Efficient Traffic Assignment for Public Transit Networks*

Lars Briem[1], Sebastian Buck[2], Holger Ebhart[3], Nicolai Mallig[4], Ben Strasser[5], Peter Vortisch[6], Dorothea Wagner[7], and Tobias Zündorf[8]

1 **Karlsruhe Institute of Technology (KIT), Karlsruhe, Germany**
lars.briem@kit.edu,
2 **Karlsruhe Institute of Technology (KIT), Karlsruhe, Germany**
sebastian.buck@kit.edu
3 **Karlsruhe Institute of Technology (KIT), Karlsruhe, Germany**
holger.ebhart@ira.uka.de
4 **Karlsruhe Institute of Technology (KIT), Karlsruhe, Germany**
nicolai.mallig@kit.edu
5 **Karlsruhe Institute of Technology (KIT), Karlsruhe, Germany**
strasser@kit.edu
6 **Karlsruhe Institute of Technology (KIT), Karlsruhe, Germany**
peter.vortisch@kit.edu
7 **Karlsruhe Institute of Technology (KIT), Karlsruhe, Germany**
dorothea.wagner@kit.edu
8 **Karlsruhe Institute of Technology (KIT), Karlsruhe, Germany**
zuendorf@kit.edu

Abstract

We study the problem of computing traffic assignments for public transit networks: Given a public transit network and a demand (i.e. a list of passengers, each with associated origin, destination, and departure time), the objective is to compute the utilization of every vehicle. Efficient assignment algorithms are a core component of many urban traffic planning tools. In this work, we present a novel algorithm for computing public transit assignments. Our approach is based upon a microscopic Monte Carlo simulation of individual passengers. In order to model realistic passenger behavior, we base all routing decisions on travel time, number of transfers, time spent walking or waiting, and delay robustness. We show how several passengers can be processed during a single scan of the network, based on the Connection Scan Algorithm [6], resulting in a highly efficient algorithm. We conclude with an experimental study, showing that our assignments are comparable in terms of quality to the state-of-the-art. Using the parallelized version of our algorithm, we are able to compute a traffic assignment for more than ten million passengers in well below a minute, which outperforms previous works by more than an order of magnitude.

1998 ACM Subject Classification G.2.2 Graph Theory

Keywords and phrases Algorithms, Optimization, Route planning, Public transportation

Digital Object Identifier 10.4230/LIPIcs.SEA.2017.20

* Tobias Zündorf's research was supported by DFG Research Grant WA 654/23-1.

16th International Symposium on Experimental Algorithms (SEA 2017).
Editors: Costas S. Iliopoulos, Solon P. Pissis, Simon J. Puglisi, and Rajeev Raman; Article No. 20; pp. 20:1–20:14

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

Traffic assignment problems are widely studied for the case of street networks and private transport. However, virtually no work addresses the algorithmic challenges of finding a traffic assignment for public transit networks, despite the fact that such assignments are a very important tool for planning public transportation services. When implementing new public transit lines (or redirecting existing ones), it is often desired, that the capacity of all the vehicles serving the lines is well utilized. If vehicles are overcrowded, then more or larger vehicles have to be deployed. However, if vehicles are only sparsely used, they may be dropped from the schedule. Using traffic assignments, the utilization of the vehicles can be estimated ahead of time allowing an efficient public transportation service design.

Determining a public transit assignment requires two components: The timetable of the underlying public transit network; and an estimation of the overall passenger flow, specified by individual origin, destination, and time triples, called demand. The most basic variant of the assignment problem asks only for the expectable utilization of the vehicles operating in the public transit network. A more elaborate variant of the problem requires that every individual passenger is assigned to a route from his origin to his destination. The objective of the assignment is in every case a statistical analysis of the network utilization. Thus, a passenger may even be proportionally assigned to several routes, in order to increase the overall accuracy of the assignment.

In this work we present a novel algorithm for computing public transit assignments. Our approach for assigning routes to individual passengers is based on a Monte Carlo simulation. Every passenger's movement through the network is simulated step by step until his destination is reached. We consider multiple criteria of each possible route, in order to achieve a realistic movement of the passengers: the arrival time at the destination, the time that has to be waited for connecting vehicles, the time that has to spent walking between stops, the number of transfers between vehicles, and the delay robustness. We describe an efficient approach for this simulation, based on the Connection Scanning Algorithm [6], which allows for efficient one to all queries on public transit networks.

Our paper is organized as follows: In Section 2 we formally define the public transit network and the demand, and we introduce the basic notation used throughout this paper. Next, we propose the notion of perceived arrival times, which we use to model passenger preferences in Section 3. We continue with presenting our algorithm in Section 4. In Section 5 we conduct an experimental study, showing that the quality of our assignment is comparable to state-of-the-art, while the running time is more than an order of magnitude lower.

1.1 Related Work

The problem of finding a traffic assignment comprises two important subproblems. First, achieving a high quality assignment requires a sophisticated procedure for deciding which route a passenger would choose in the real world. Second, efficient route planning algorithms are required in order to compute possible routes to choose from. There has been a lot of research addressing route planning problems in recent years. A comprehensive overview of state-of-the art route planning algorithms is given in [2]. Unlike time independent route planning, it is quite hard to accelerate routing algorithms for public transit [4]. Several speed-up techniques have been proposed, in order to increase the efficiency of public transit route planning, many of them exploit the special structure of timetables. The RAPTOR [5] algorithm is one of the first techniques solely based on an efficient timetable representation. With Transfer Patterns [1, 3] a first approach that utilizes preprocessing in order to enable fast

public transit queries was introduced. The author of [12] proposes an algorithm for fast profile queries, based on a data model focused on trips and transfers between them. The Connection Scanning Algorithm (CSA) [6, 11] relies on a particularly simple data model, namely a sorted array of connection. Nevertheless, it allows for fast one to all profile queries. Based on CSA, the MEAT [7] technique was developed, enabling delay robust journey planning.

An overview over traffic assignment techniques and models can be found in [10]. An important concept for achieving realistic traffic assignments are equilibrium models, which enable that the assignment adapts to congested parts of the network. Various variants of equilibrium models are discussed in [8]. Just like route planning, traffic assignment problems become more difficult when applied to public transit networks. An implementation of state-of-the-art traffic assignment algorithms is available in VISUM from PTV AG[1].

2 Preliminaries

Our algorithm operates on a public transit network $(\mathcal{C}, G)$ consisting of a finite set of elementary connections $\mathcal{C}$ and a directed, weighted *transfer graph* $G = (\mathcal{V}, \mathcal{E}, \tau_{\text{trans}})$.

A *connection* $c \in \mathcal{C}$ is a tuple $(v_{\text{dep}}(c), v_{\text{arr}}(c), \tau_{\text{dep}}(c), \tau_{\text{arr}}(c), \text{trip}(c))$ representing a vehicle driving from a *departure stop* $v_{\text{dep}}(c) \in \mathcal{V}$ to an *arrival stop* $v_{\text{arr}}(c) \in \mathcal{V}$ without any intermediate stops. The vehicle is scheduled to depart from $v_{\text{dep}}(c)$ at the *departure time* $\tau_{\text{dep}}(c)$ and arrives at $v_{\text{arr}}(c)$ at the *arrival time* $\tau_{\text{arr}}(c)$ which we require to be greater than $\tau_{\text{dep}}(c)$. We assume that connections departing from the same stop have a well defined order of departure, i.e. $v_{\text{dep}}(c) = v_{\text{dep}}(c') \Rightarrow \tau_{\text{dep}}(c) \neq \tau_{\text{dep}}(c')$. However, this is not a real restriction, as such a scenario is unrealistic, and a unique order could be established by perturbing the departure times by some $\varepsilon > 0$. Consecutive connections c_1 and c_2 are part of a *trip* $\text{trip}(c_1) = \text{trip}(c_2)$ if they are served by the same vehicle. Using two successive connections of different trips requires a transfer in between.

Valid *transfers* are defined using the transfer graph $G = (\mathcal{V}, \mathcal{E}, \tau_{\text{trans}})$, where $\mathcal{V}$ is a set of *vertices*, $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ is a set of *directed edges*, and $\tau_{\text{trans}} \colon \mathcal{E} \to \mathbb{N}_0$ is an edge weight representing the minimal required *transfer time* for using an edge. We expand τ_{trans} for arbitrary pairs of vertices $u, v \in \mathcal{V}$, by setting $\tau_{\text{trans}}(u, v) := \tau_{\text{trans}}(e)$ if there exists an edge $e = (u, v) \in \mathcal{E}$, otherwise we define $\tau_{\text{trans}}(u, v) := \infty$. Transferring between connections c_1 and c_2 of different trips is only possible if there exists an edge $e = (v_{\text{arr}}(c_1), v_{\text{dep}}(c_2))$ connecting the arrival stop of the first connection c_1 with the departure stop of the second connection c_2, such that $\tau_{\text{arr}}(c_1) + \tau_{\text{trans}}(e) \leq \tau_{\text{dep}}(c_2)$. Note that by our definition, a transfer edge has to be used even if $v_{\text{arr}}(c_1) = v_{\text{dep}}(c_2)$. This enables us to model minimum transfer times for changing between trips stopping at the same vertex. We define by $\tau_{\text{wait}}(v, \tau, c) := \tau_{\text{dep}}(c) - \tau - \tau_{\text{trans}}(v, v_{\text{dep}}(c))$ the additional waiting time after transferring from vertex v at time τ to $v_{\text{dep}}(c)$, before c departs. Given two connections $c, c' \in \mathcal{C}$, the waiting time for transferring from c to c' is given by $\tau_{\text{wait}}(c, c') := \tau_{\text{wait}}(v_{\text{arr}}(c), \tau_{\text{arr}}(c), c')$. Thus, a transfer between connections c and c' is valid if and only if $\tau_{\text{wait}}(c, c') \geq 0$ holds. Following [5, 11], we require that the transfer graph is transitively closed and fulfills the triangle inequality. This ensures that an edge $e = (u, v)$ is always a shortest path in G from u to v. We call a vertex $v \in \mathcal{V}$ *stop* if there exists at least one connection c such that $v_{\text{dep}}(c) = v$ or $v_{\text{arr}}(c) = v$, the set of all stops is denoted by $\mathcal{S} \subseteq \mathcal{V}$.

A *journey* $j = \langle c_1, \ldots, c_k \rangle$ is a sequence of connections, such that transferring between subsequent connections in j is valid. Formally, this means that the connections in j have to be

[1] `http://vision-traffic.ptvgroup.com/en-us/products/ptv-visum/`

sorted chronologically. Furthermore, we require that subsequent connections c_i and c_{i+1} are either part of the same trip (i.e. $\text{trip}(c_i) = \text{trip}(c_{i+1})$), or there exits a transfer connecting them (i.e. $\tau_{\text{wait}}(c_i, c_{i+1}) \geq 0$), for every $i \in [1, k-1]$. Analogous to connections, we define a departure stop $v_{\text{dep}}(j) := v_{\text{dep}}(c_1)$, a departure time $\tau_{\text{dep}}(j) := \tau_{\text{dep}}(c_1)$, an arrival stop $v_{\text{arr}}(j) := v_{\text{arr}}(c_k)$, and an arrival time $\tau_{\text{arr}}(j) := \tau_{\text{arr}}(c_k)$, for a journey $j = \langle c_1, \ldots, c_k \rangle$.

In addition to a public transit network, the input of a traffic assignment instance also contains a set of demands $\mathcal{D}$. A demand $D \in \mathcal{D}$ is a triple $(o(D), d(D), \tau_{\text{dep}}(D))$ representing a passenger who wishes to travel from his *origin* $o(D) \in \mathcal{V}$ to his *destination* $d(D) \in \mathcal{V}$, starting at his departure time $\tau_{\text{dep}}(D)$. The objective of the traffic assignment is to compute for every passenger represented by $\mathcal{D}$ a journey that satisfies his demand. A journey j *satisfies* a demand D, if it can be used to travel from $o(D)$ to $d(D)$, with a departure time of at least $\tau_{\text{dep}}(D)$. More precisely, the journey must be reachable from the demanded origin when departing after $\tau_{\text{dep}}(D)$. This is the case, if either the journey departs not earlier than $\tau_{\text{dep}}(D)$ directly from $o(D)$ (i.e. $o(D) = o(j) \wedge \tau_{\text{dep}}(D) \leq \tau_{\text{dep}}(j)$) or if transferring from the origin to the journeys departure is valid (i.e. $\tau_{\text{dep}}(D) + \tau_{\text{trans}}(o(D), v_{\text{dep}}(j)) \leq \tau_{\text{dep}}(j)$). Furthermore, the journey has to end at the demanded destination, either directly or by using an additional transfer. Formally, we assume that either $v_{\text{arr}}(j) = d(D)$ or $\tau_{\text{trans}}(v_{\text{arr}}(j), d(D)) < \infty$ holds. An empty journey $j = \langle\rangle$ satisfies a demand D, if $o(D) = d(D)$ or $\tau_{\text{trans}}(o(D), d(D)) < \infty$ holds. Note that we do not require a journey to be optimal with respect to any metric, in order to satisfy a demand.

Given two vertices $u, v \in \mathcal{V}$, a *u-v-profile* is a function $f^{u,v}(\tau)$ mapping departure times τ onto the minimal costs for traveling from u at time τ to v, with respect to some cost function. If no such journey exists we define $f^{u,v}(\tau)$ as ∞. Profile functions are piecewise linear, since for every departure time τ the value $f^{u,v}(\tau)$ corresponds to a unique journey. Each journey contributing to $f^{u,v}(\tau)$ is either empty or has a fixed departure time, depending solely on the journeys first connection. Since there exists only a finite number of connections, every profile function can be described using a finite number of supporting points.

3 Perceived Arrival Time (PAT)

The core problem of computing a public transit assignment, is to decide for each passenger which connections he takes in order to reach his destination. The quality of the resulting assignment highly depends on these choices. Thus it is important to model the behavior and preferences of the passengers in a realistic way. This means that we cannot assign connections to the passengers solely based on the travel time of the resulting journey. For example, a passenger might prefer a journey with a slightly longer travel time, if this reduces the number of changes between vehicles.

As a means of reflecting the passengers preferences, we introduce the notion of *perceived arrival time* (short PAT). Given a connection $c \in \mathcal{C}$ and a destination $d \in \mathcal{V}$, the perceived arrival time $\tau^{\text{P}}(c, d)$ is a measurement for how useful c is in order do reach d. The PAT $\tau^{\text{P}}(c, d)$ depends on the possible journeys that end at d and contain c. We consider five properties of these journeys that influence the perceived arrival time: the actual arrival time at d, the number of transfers, the time spend walking, the time spend waiting, and the delay robustness. We account for walking and waiting time by weighting the corresponding times with factors $\lambda_{\text{walk}}, \lambda_{\text{wait}} \in \mathbb{R}$. For every transfer during the journey we add an additional cost of $\lambda_{\text{trans}} \in \mathbb{R}$. Finally, we incorporate delay robustness by computing the expected arrival time under the assumption that each connection has a random delay of at most $\Delta_\tau^{\max}$.

We adapt the concept of minimum expected arrival time (MEAT) introduced by Dibbelt et al. [7], in order to model delay robustness. Following their approach, we introduce a

random variable $\Delta_\tau^c \in \mathbb{R}_0^+$ for every connection $c \in \mathcal{C}$, that represents the delay of the connection. This means that the arrival stop $v_{\text{arr}}(c)$ will be reached at $\tau_{\text{arr}}(c) + \Delta_\tau^c$. Thus, transferring to another connection can become invalid, if the delay exceeds the waiting time of the transfer. The probability that the delay is at most x, is given by the cumulative distribution function $P[\Delta_\tau^c \leq x]$. We define $P[\Delta_\tau^c \leq x]$ as follows: $P[\Delta_\tau^c \leq x] := 0$ for $x \leq 0$, $P[\Delta_\tau^c \leq x] := 1$ for $x \geq \Delta_\tau^{\max}$, and $P[\Delta_\tau^c \leq x] := 31/30 - (11\Delta_\tau^{\max})/(300x + 30\Delta_\tau^{\max})$ for $0 < x < 1$, where $\Delta_\tau^{\max}$ is the maximal delay that can occur. Based on this, the probability that a transfer between two connections $c, c' \in \mathcal{C}$ is valid, is given by $P[\Delta_\tau^c \leq \tau_{\text{wait}}(c, c')]$. Additionally, we define the probability $P[y < \Delta_\tau^c \leq x] := P[\Delta_\tau^c \leq x] - P[\Delta_\tau^c \leq y]$ that the delay of c is between y and x. For more details on the delay model see [7].

We now proceed with defining the perceived arrival time $\tau^{\text{p}}(c, d)$ in a recursive way, which allows us to take all journeys containing c into account. There exist three distinct cases for continuing a journey after using the connection c. If it is possible to use a transfer from the arrival stop of c to the destination, then the journey can be completed by walking. Otherwise, the journey continues either with the next connection of the same trip as $\text{trip}(c)$ or the vehicle serving c is left at $v_{\text{arr}}(c)$. Therefore we define

$$\tau_{\text{arr}}^{\text{p}}(c, d) := \min\{\tau_{\text{arr}}^{\text{p}}(c, d \mid \text{walk}),\ \tau_{\text{arr}}^{\text{p}}(c, d \mid \text{trip}),\ \tau_{\text{arr}}^{\text{p}}(c, d \mid \text{trans})\},$$

where $\tau_{\text{arr}}^{\text{p}}(c, d \mid \text{walk})$ is the PAT under the constraint that the journey is completed by walking from c to d, $\tau_{\text{arr}}^{\text{p}}(c, d \mid \text{trip})$ is the PAT under the constraint that the journey continues with the same trip as $\text{trip}(c)$, and $\tau_{\text{arr}}^{\text{p}}(c, d \mid \text{trans})$ is the PAT under the constraint for that the journey continues with a transfer to another connection after c. The perceived arrival time for walking to the destination is defined as:

$$\tau_{\text{arr}}^{\text{p}}(c, d \mid \text{walk}) := \begin{cases} \tau_{\text{arr}}(c) & \text{if } v_{\text{arr}}(c) = d \\ \tau_{\text{arr}}(c) + \lambda_{\text{walk}} \cdot \tau_{\text{trans}}\big(v_{\text{arr}}(c), d\big) & \text{otherwise.} \end{cases}$$

This means that the PAT is the actual arrival time, if the destination is reached directly by using c. If this is not the case, the time needed for walking to the destination is multiplied with the cost factor λ_{walk} and added to the arrival time. For the definition of $\tau_{\text{arr}}^{\text{p}}(c, d \mid \text{trip})$ let $\mathcal{T}(c) := \{c' \in \mathcal{C} \mid \text{trip}(c') = \text{trip}(c) \wedge \tau_{\text{dep}}(c') \geq \tau_{\text{arr}}(c)\}$ be the set of all connections following after c in the trip of c. We then define the PAT for continuing with the same trip as the minimum over the perceived arrival times of all subsequent connections in the trip:

$$\tau_{\text{arr}}^{\text{p}}(c, d \mid \text{trip}) := \begin{cases} \min\{\tau^{\text{p}}(c', d) \mid c' \in \mathcal{T}(c)\} & \text{if } \mathcal{T}(c) \neq \emptyset \\ \infty & \text{otherwise.} \end{cases}$$

Finally, we proceed with defining the PAT $\tau_{\text{arr}}^{\text{p}}(c, d \mid \text{trans})$ for transferring from c to a connection c' of another trip. For this purpose, we first introduce the perceived time $\tau_{\text{trans}}^{\text{p}}(u, v)$ for transferring from u to v as a weighted sum of walking respectively waiting time and transfer costs:

$$\tau_{\text{trans}}^{\text{p}}(u, v) := \begin{cases} \lambda_{\text{trans}} + \lambda_{\text{wait}} \cdot \tau_{\text{trans}}(u, u) & \text{if } u = v \\ \lambda_{\text{trans}} + \lambda_{\text{walk}} \cdot \tau_{\text{trans}}(u, v) & \text{otherwise.} \end{cases}$$

Additionally, we define $\tau_{\text{trans}}^{\text{p}}(c, c') := \tau_{\text{trans}}^{\text{p}}(v_{\text{arr}}(c), v_{\text{dep}}(c'))$, in order to reflect the perceived time for transferring from a connection c to another connection c'. Transferring between connections $c, c' \in \mathcal{C}$ may include some additional waiting time $\tau_{\text{wait}}(c, c')$ at the departure stop of c', after the actual transfer took place. We account for this by introducing the perceived

waiting times $\tau^{\mathrm{p}}_{\mathrm{wait}}(v,\tau,c) := \lambda_{\mathrm{wait}} \cdot \tau_{\mathrm{wait}}(v,\tau,c)$, respectively $\tau^{\mathrm{p}}_{\mathrm{wait}}(c,c') := \lambda_{\mathrm{wait}} \cdot \tau_{\mathrm{wait}}(c,c')$. Using this we define the perceived arrival time $\tau^{\mathrm{p}}_{\mathrm{arr}}(c,c',d) := \tau^{\mathrm{p}}_{\mathrm{trans}}(c,c') + \tau^{\mathrm{p}}_{\mathrm{wait}}(c,c') + \tau^{\mathrm{p}}_{\mathrm{arr}}(c',d)$ of journeys starting with the connection c, followed by a transfer to the connection c', and ending at the destination d. In order to define $\tau^{\mathrm{p}}_{\mathrm{arr}}(c,d \mid \mathrm{trans})$, we only need to specify, which connection c' is used after c. Here, we take not only the perceived arrival time $\tau^{\mathrm{p}}(c',d)$ into account, but also the possibility that the transfer from c to c' might become invalid due to a delay of c. We achieve this by considering all connections that are Pareto-optimal with respect to their PAT and their delay robustness as possible candidates. Based on the set $\mathcal{R}(c) := \{c' \in \mathcal{C} \mid \tau_{\mathrm{wait}}(c,c') \geq 0\}$ of all connections that are reachable from c, the set $\mathcal{R}_{\mathrm{opt}}(c)$ of Pareto-optimal connections, reachable from c can be defined as:

$$\mathcal{R}_{\mathrm{opt}}(c) := \{c' \in \mathcal{R}(c) \mid \forall \bar{c} \in \mathcal{R}(c) : \tau_{\mathrm{wait}}(c,\bar{c}) \geq \tau_{\mathrm{wait}}(c,c') \Rightarrow \tau^{\mathrm{p}}_{\mathrm{arr}}(c,\bar{c},d) \geq \tau^{\mathrm{p}}_{\mathrm{arr}}(c,c',d)\}.$$

Let $\langle c_1,\ldots,c_k\rangle$ be the sequence of connections from $\mathcal{R}_{\mathrm{opt}}(c)$ sorted by their waiting time in increasing order, that is $\tau_{\mathrm{wait}}(c,c_i) \geq \tau_{\mathrm{wait}}(c,c_{i-1})$ for $i \in [2,k]$. This means transferring from c to c_1 results in the minimum PAT. If however, transferring to c_1 is not possible, due to delay, c_2 is the next best option, and so on. We define the waiting time when transferring to the i-th connection of the sequence as $\tau^c_{\mathrm{wait}}(i) := \tau_{\mathrm{wait}}(c,c_i)$ for $i \in [1,k]$. For $i \neq [1,k]$ we set $\tau^c_{\mathrm{wait}}(i) := -\infty$. Finally we define $\tau^{\mathrm{p}}_{\mathrm{trans}}(c,d)$ as the sum of the perceived arrival times of all c_i, weighted by the probability that transfer to c_i is valid, while the transfer to c_{i-1} is invalid:

$$\tau^{\mathrm{p}}_{\mathrm{arr}}(c,d \mid \mathrm{trans}) := \begin{cases} \displaystyle\sum_{i=1}^{k} \left(\frac{P[\tau^c_{\mathrm{wait}}(i-1) < \Delta^c_\tau \leq \tau^c_{\mathrm{wait}}(i)]}{P[\Delta^c_\tau \leq \tau^c_{\mathrm{wait}}(k)]} \cdot \tau^{\mathrm{p}}_{\mathrm{arr}}(c,c_i,d) \right) & \text{if } k > 0 \\ \infty & \text{otherwise.} \end{cases}$$

Note that our recursive definition of $\tau^{\mathrm{p}}(c,d)$ is well-defined, since it only depends on the perceived arrival times of connections c' with $\tau_{\mathrm{dep}}(c') > \tau_{\mathrm{dep}}(c)$.

4 Our Approach

Our algorithm is based on a microscopic Monte Carlo simulation of individual passengers represented by a unique integer identifier. For each vertex of the network we maintain a list containing all the passengers, who currently reside at the vertex. Passengers are gradually moved from one vertex to the next, until they reach their destination. We decide for every passenger which vertex he visits next, based on perceived arrival times. The vertex chosen as next vertex is not necessarily the one that minimizes the PAT. We assign a probability based on the perceived arrival times to every possible option. Next, we choose randomly an option for every passenger. In order to increase the accuracy of the simulation we generate λ_{mul} times as many passengers as specified by the demand. After the simulation finished, the results are divided by λ_{mul}, in order to obtain a stochastic distribution of the passengers specified by the demand.

We observe that passengers with the same destination d and roughly the same time of travel, will eventually encounter each other on their journeys (at least at d). If they meet before d, then there exists a vertex v where they have the same options for continuing their journey to d. Our algorithm exploits this observation by evaluating the options and computing the decisions for all passengers at v at once. In order to achieve this, we partition the passengers based on their destination. We proceed with showing how the traffic assignment for passenger with a common destination can be computed. A complete traffic assignment

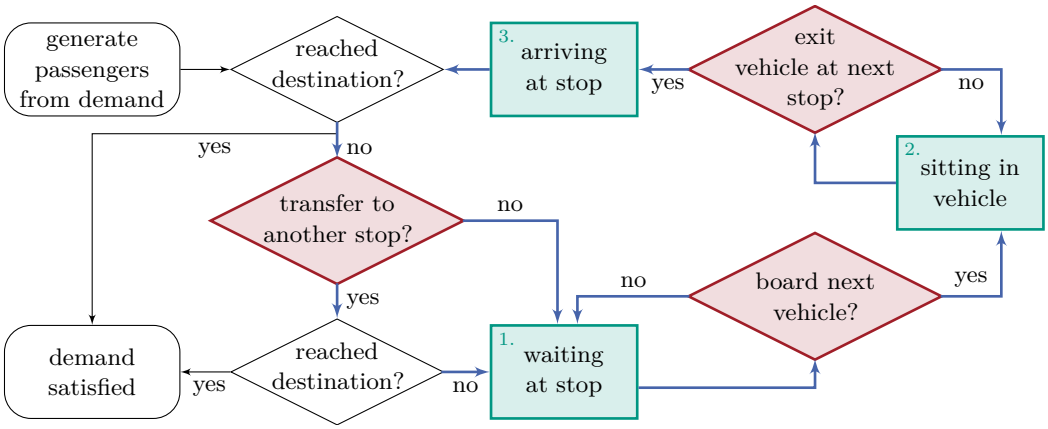

Figure 1 A flowchart describing the movement of a passenger through the network. Passengers are generated according to the demand. If their destination differs from their origin, then they enter the main cycle (colored part) of the simulation. Passengers traverse the main cycle until they reach their destination. During this they can be in one of three situations (green). The situation a passenger is in changes depending on his decisions (red).

can be obtained by doing this for every destination and aggregating the results. For the remainder of this chapter we assume d to be a fixed destination vertex.

We compute the traffic assignment for passengers with destination d in three phases. First, we compute for every connection the minimum perceived arrival times for taking and avoiding that connection. Next, we simulate the movement of the passengers through the network. Every time a passenger could use a connection c without producing an invalid transfer, we decide based on the previously computed perceived transfer times whether the passenger takes the connection c. Connections that are used by the passenger are added to the passengers journey. Finally, we simplify the journeys by removing unwanted cycles.

4.1 Perceived Arrival Time Computation

In the first phase we compute all information required to build journeys one connection at a time. We identified three situations that can occur during the simulation of a passenger's movement, that require a decision about the journey's continuation (see Figure 1).

The first situation arises when a passenger waits at a stop s, while a connection c departs from s. In this case, it has to be decided if the passenger boards the vehicle serving c, or keeps waiting at the stop. In order to make this decision, we need the perceived arrival times for both alternatives. The PAT for using the connection c (i.e. boarding the vehicle) is given by $\tau^{\mathrm{p}}(c, d)$. On the other hand, skipping the connection c and waiting at the stop, implies that some later connection departing from the stop has to be taken. Transferring to another stop is not an option, as the passenger transferred to his current stop s, with the intention to board some vehicle at s. The set of all alternative connections departing from the same stop is given by $\mathcal{A}(c) := \{c' \in \mathcal{C} \mid v_{\mathrm{dep}}(c') = v_{\mathrm{dep}}(c),\ \tau_{\mathrm{dep}}(c') > \tau_{\mathrm{dep}}(c)\}$. We use these alternative connections to obtain the PAT $\tau^{\mathrm{p}}_{\mathrm{arr}}(c, d \mid \mathrm{skip}\, c)$ for skipping the connection c as the sum of the additional waiting time and the perceived arrival time of the best alternative connection. Formally, we define: $\tau^{\mathrm{p}}_{\mathrm{arr}}(c, d \mid \mathrm{skip}\, c) := \min\{\tau^{\mathrm{p}}_{\mathrm{wait}}(v_{\mathrm{dep}}(c), \tau_{\mathrm{dep}}(c), c') + \tau^{\mathrm{p}}_{\mathrm{arr}}(c', d) \mid c' \in \mathcal{A}(c)\}$.

The second situation affects passengers using a connection that is not the last connection of its trip. These passengers again have to make a binary decision. Either they leave the vehicle at the arrival stop of the current connection, or they use another connection

of the trip. As before, making this decision requires the perceived arrival times of both alternatives. The PAT for continuing with the same trip is given by $\tau^{\mathrm{p}}_{\mathrm{arr}}(c, d \mid \mathrm{trip})$. When disembarking the vehicle, a passenger can continue his journey by either walking to his destination or transferring to another vehicle. Therefore, the PAT for disembarking is given by $\tau^{\mathrm{p}}_{\mathrm{arr}}(c, d \mid \mathrm{disembark}) := \min\{\tau^{\mathrm{p}}_{\mathrm{arr}}(c, d \mid \mathrm{walk}), \tau^{\mathrm{p}}_{\mathrm{arr}}(c, d \mid \mathrm{trans})\}$.

The last situation where a decision has to be made occurs when a passenger leaves a vehicle, but has not yet reached his destination. In this case, it has to be decided to which stop the passenger transfers, in order to wait for another connection. This decision requires a perceived arrival times for every stop v that can be reached by a transfer. Similar to the definition of $\tau^{\mathrm{p}}_{\mathrm{arr}}(c, d \mid \mathrm{skip}\, c)$, the PAT for the stop v is given by the PAT of the best connection c departing from v plus the additional waiting time between the arrival time τ at v and the departure of c. As this value is required for every possible arrival time τ at v, we simply compute a profile function $f^{v,d}_{\mathrm{wait}}(\tau)$ for every vertex, which we define as:

$$f^{v,d}_{\mathrm{wait}}(\tau) := \min\{\tau^{\mathrm{p}}_{\mathrm{wait}}(v, \tau, c) + \tau^{\mathrm{p}}_{\mathrm{arr}}(c, d) \mid c \in \mathcal{C},\, \tau_{\mathrm{dep}}(c) \geq \tau,\, v_{\mathrm{dep}}(c) = v\}.$$

In summary, we require for decision making three values per connection: $\tau^{\mathrm{p}}_{\mathrm{arr}}(c, d \mid \mathrm{trip})$, $\tau^{\mathrm{p}}_{\mathrm{arr}}(c, d \mid \mathrm{skip}\, c)$, and $\tau^{\mathrm{p}}_{\mathrm{arr}}(c, d \mid \mathrm{disembark})$, as well as a profile function $f^{v,d}_{\mathrm{wait}}(\tau)$ per vertex. We now show how these values can be computed in a single sweep over the connection array. As basis for our algorithm, we use the MEAT algorithm [7], which allows for efficient all-to-one profile queries. Instead of computing minimum expected arrival time profiles, as the original MEAT algorithm does, we compute minimum perceived arrival time profiles. In addition to the profile $f^{v,d}_{\mathrm{wait}}(\tau)$, we compute a second profile $f^{v,d}_{\mathrm{trans}}(\tau)$, which we use in order to determine the three PAT values needed per connection. The difference between the two profile is that $f^{v,d}_{\mathrm{trans}}(\tau)$ requires an initial transfer to another stop. Formally, we define:

$$f^{v,d}_{\mathrm{trans}}(\tau) := \min\{\tau^{\mathrm{p}}_{\mathrm{trans}}(v, v_{\mathrm{dep}}(c)) + \tau^{\mathrm{p}}_{\mathrm{wait}}(v, \tau, c) + \tau^{\mathrm{p}}_{\mathrm{arr}}(c, d) \mid c \in \mathcal{C},\, \tau_{\mathrm{wait}}(v, \tau, c) \geq 0\}.$$

Our algorithm maintains for every vertex the two initially incomplete profiles $f^{v,d}_{\mathrm{wait}}(\cdot)$, and $f^{v,d}_{\mathrm{trans}}(\cdot)$. Additionally we store for every trip t a value $\tau^{\mathrm{p}}_{\mathrm{arr}}(t)$, that keeps track of the current value for $\tau^{\mathrm{p}}_{\mathrm{arr}}(c, d \mid \mathrm{trip})$ with $\mathrm{trip}(c) = t$, and is initially ∞. We scan the connection array in decreasing order by departure time. For every connection c we can directly determine the three required values. Since we store the arrival time for continuing with the same trip separately we can set $\tau^{\mathrm{p}}_{\mathrm{arr}}(c, d \mid \mathrm{trip}) \leftarrow \tau^{\mathrm{p}}_{\mathrm{arr}}(\mathrm{trip}(c))$. The PAT for ignoring the connection c is given by the profile that describes waiting at the departure stop of c. Thus, we set $\tau^{\mathrm{p}}_{\mathrm{arr}}(c, d \mid \mathrm{skip}\, c) \leftarrow f^{v_{\mathrm{dep}}(c),d}_{\mathrm{wait}}(\tau_{\mathrm{dep}}(c))$. Similarly, the PAT for disembarking at the arrival stop of c is given by the profile that requires an initial transfer. Accordingly, we set $\tau^{\mathrm{p}}_{\mathrm{arr}}(c, d \mid \mathrm{disembark}) \leftarrow f^{v_{\mathrm{arr}}(c),d}_{\mathrm{trans}}(\tau_{\mathrm{arr}}(c))$. For the special case that a transfer edge from the arrival stop of c to the destination exists, we have to consider the possibility of walking to the destination. Therefore, we set $\tau^{\mathrm{p}}_{\mathrm{arr}}(c, d \mid \mathrm{disembark}) \leftarrow \tau_{\mathrm{arr}}(c) + \tau_{\mathrm{trans}}(v_{\mathrm{arr}}(c), d)$, if this is smaller than the previous value of $\tau^{\mathrm{p}}_{\mathrm{arr}}(c, d \mid \mathrm{disembark})$. Afterwards, we temporarily compute the PAT of the connection c: $\tau^{\mathrm{p}}_{\mathrm{arr}}(c, d) \leftarrow \min(\tau^{\mathrm{p}}_{\mathrm{arr}}(c, d \mid \mathrm{trip}), \tau^{\mathrm{p}}_{\mathrm{arr}}(c, d \mid \mathrm{disembark}))$. We use this value in order to update the profiles and the value $\tau^{\mathrm{p}}_{\mathrm{arr}}(\mathrm{trip}(c))$. First we set $\tau^{\mathrm{p}}_{\mathrm{arr}}(\mathrm{trip}(c)) \leftarrow \tau^{\mathrm{p}}_{\mathrm{arr}}(c, d)$. Next, we add the point $(\tau_{\mathrm{dep}}(c), \tau^{\mathrm{p}}_{\mathrm{arr}}(c, d))$ as a break point to the profile $f^{v_{\mathrm{dep}}(c),d}_{\mathrm{wait}}(\cdot)$, unless this profile already contains a breakpoint with smaller PAT. Finally, we iterate over all vertices v with $(v, v_{\mathrm{dep}}(c)) \in \mathcal{E}$. For each such vertex v we add the point $(\tau_{\mathrm{dep}}(c) - \tau_{\mathrm{trans}}(v, v_{\mathrm{dep}}(c)), \tau^{\mathrm{p}}_{\mathrm{arr}}(c, d))$ as a break point to the profile $f^{v,d}_{\mathrm{trans}}(\cdot)$, unless the profile already contains a breakpoint with smaller PAT. We repeat this process for every connection. Afterwards, we have computed all values required for decision making and can continue with the actual assignment.

4.2 Assignment

The second phase of our algorithm uses the previously computed perceived arrival times to compute the journeys for all the passengers with destination d. To this intent, we maintain for every passenger a list of connections used by the passenger. Additionally, we maintain a list of passengers for every vertex and trip, representing the passengers currently waiting at the vertex, respectively sitting in the vehicle serving the trip. Furthermore, we use a queue sorted by arrival time for every vertex, containing the passengers that are currently transferring to the stop. The transfer queue of each vertex v is initialized with passengers created from the demand with origin v, using their desired departure time as keys.

We now describe how the passengers movement through the network is simulated, using a single scan over the connection array in ascending order by departure time. During this scan, we decide for each connection, which passengers use the connection. When scanning a connection c we first determine the set of passengers that could enter the vehicle. We establish this by removing all the passengers from the transfer queue of $v_{\text{dep}}(c)$ that arrive at $v_{\text{dep}}(c)$ before $\tau_{\text{dep}}(c)$ These passengers are then added to the list of passengers waiting at $v_{\text{dep}}(c)$. Afterwards, the list of passengers waiting at $v_{\text{dep}}(c)$ comprises exactly the passengers that could enter c. We decide whether the passengers take c or not, based on the two PATs $\tau_1 = \min(\tau^{\text{p}}_{\text{arr}}(c, d \mid \text{trip}), \tau^{\text{p}}_{\text{arr}}(c, d \mid \text{disembark}))$ and $\tau_2 = \tau^{\text{p}}_{\text{arr}}(c, d \mid \text{skip}\, c)$. We do so by assigning a probability $P[i]$, that describes the likelihood of a passenger using the option associated with τ_i, to each of the alternatives.

In general, given k options, with perceived arrival times $\tau_1, \ldots, \tau_k$, we define the probability $P[i]$ for choosing option i as follows. First, we compute the *gain* of each option, which we define as $g(i) := \max(0, \min_{j \neq i}(\tau^{\text{p}}(j)) - \tau^{\text{p}}(i) + \lambda_{\Delta\max})$. Doing so results in a gain of zero, for options that differ from the optimum by more than $\lambda_{\Delta\max}$. For all other options τ_i, the gain correlates linearly to the difference between τ_i and the optimal, respectively nest best option. The probability that a passenger uses option i is equivalent to the gain of option i, divided by the sum of the gain of all other options. Formally we define: $P[i] := g(i) / \sum_{j=1}^{k} g(j))$.

Using this, the probabilities of the two options τ_1 (for using the connection c), and τ_2 (for skipping the connection), are given by $P[1] := (\tau_2 - \tau_1 + \lambda_{\Delta\max})/2\lambda_{\Delta\max}$, and $P[1] := 1 - P[2]$. Based on these probabilities, we make a random decision for every passenger waiting at the departure stop of c. If a passenger happens to enter the connection, then he is removed from the list of passengers waiting at $v_{\text{dep}}(c)$, and added to the list of passengers sitting the trip trip(c). Furthermore, the connection c is added to the journey of the passenger.

Next, we decide for every passenger sitting in the trip, if he disembarks at the arrival stop of the connection. In this case, the two options are given by $\tau_1 = \tau^{\text{p}}_{\text{arr}}(c, d \mid \text{disembark})$ for leaving the vehicle, and $\tau_2 = \tau^{\text{p}}_{\text{arr}}(c, d \mid \text{trip})$ for continuing with the same trip. As before we compute the probabilities of both options, and make a random decision for every passenger sitting in the trip, based on these probabilities. Passengers disembarking the vehicle are collected in a temporary list. If the arrival stop of the connections happens to be the destination vertex, then the journeys of all passengers in the temporary list are complete, and we simply continue with the next connection. Otherwise, we have to decide for all passenger in the temporary list, to which vertex they transfers. Let $v_1, \ldots, v_k$ be all vertices for which $\tau_{\text{trans}}(v_{\text{arr}}(c), v_i) < \infty$ holds. The perceived arrival time for transferring to v_i is given by $\tau_i = \tau^{\text{p}}_{\text{trans}}(v_{\text{arr}}(c), v_i) + f^{v_i,d}_{\text{wait}}(\tau_{\text{arr}}(c) + \tau_{\text{trans}}(v_{\text{arr}}(c), v_i))$. Based on these perceived arrival times, we compute the probability of a passenger transferring to vertex v_i, for $i \in [1, k]$. As before, we determine for every passenger randomly, which option he chooses. Finally, passengers transferring to vertex v are added to the queue of transferring passengers of the vertex v, their arrival time at v is $\tau_{\text{arr}}(c) + \tau_{\text{trans}}(v_{\text{arr}}(c), v)$. We repeat this process

for every connection$c \in \mathcal{C}$. After processing every connection, we have assigned journeys to all passengers except the ones where no valid journey exists.

4.3 Cycle Elimination

During the second phase, we assigned a journey to every passenger which might not necessarily be an optimal journey. Therefore it is possible that the assigned journey contains cycles. In fact, it is even possible that a journey that is optimal with respect to perceived arrival time can contain cycles. This could be the case if the waiting cost λ_{wait} is very high, such that driving in a circle instead of waiting reduces the perceived arrival time. However, for some applications it might be undesirable or inadmissible to assign journeys containing cycles. Thus, we now describe an optional third phase of our algorithm, that removes all cycles from the assigned journeys.

In order to detect and remove cycles from a journey $j = \langle c_1, \dots, c_k \rangle$ satisfying a demand D, we first convert it into a sequence $\langle (v_1, \tau_1), \dots, (v_{2k+2}, \tau_{2k+2}) \rangle$ of vertex, time pairs. We do so by setting $v_{2i} := v_{\text{dep}}(c_i)$, $\tau_{2i} := \tau_{\text{dep}}(c_i)$, $v_{2i+1} := v_{\text{arr}}(c_i)$, and $\tau_{2i+1} := \tau_{\text{arr}}(c_i)$, for $i \in [1, k]$. Furthermore we define the first and last pair as $v_1 := o(D)$, $\tau_1 := \tau_{\text{dep}}(D)$, $v_{2k+2} := d(D)$, and $\tau_{2k+2} := \infty$. Given this, we say that the journey contains a cycle, if their exist indices i and $j > i + 1$, such that the part of the journey between vertices v_i and v_j can be replaced by a transfer. This is possible if $\tau_i + \tau_{\text{trans}}(v_i, v_j) \leq \tau_j$ holds. Since the transfer graph is transitively closed, it consist of disjoint cliques. Thus a journey can only contain a cycle if it contains two vertices v_i, v_j of the same clique. We can check this efficiently while iterating through the sequence of vertex, time pairs. For every $i \in [1, 2k + 2]$ add i to a set associated with the clique that contains v_i Afterwards we check for each of these sets, if it contains indices i, j such that $\tau_i + \tau_{\text{trans}}(v_i, v_j) \leq \tau_j$ holds. If we found such indices i, j, then we have also found a cycle that can be replaced with a transfer. We remove this cycle be removing the connections $c_{\lfloor i/2 \rfloor}, \dots, c_{\lceil j/2 \rceil}$ from the journey.

4.4 Parallelization

Our algorithm begins with a short setup phase, during which the connections get sorted, and the passengers get divided by their destination. Afterwards, a separate assignment is computed for every destination. Finally, the results are aggregated and the algorithm terminates. The assignment computation for the different destinations is by far the most complex part of the algorithm and can be performed for every destination independently. Therefore, it is quite easy to parallelize this part of the algorithm. First, the destinations a distributed among the available processors. Afterwards each processor computes an independent assignment for the corresponding destinations.

5 Evaluation

We implemented our algorithm in C++ compiled with GCC version 5.3.1 and optimization flag -O3. Experiments were conducted on a quad core Intel Xeon E5-1630v3 clocked at 3.7 GHz, with 128 GiB of DDR4-2133 RAM, 10 MiB of L3 cach, and 256 KiB of L2 cache.

5.1 Instance

We tested our algorithm on a public transit network covering the greater region of Stuttgart, as well as some long distance routes, reaching as far as Mannheim, Basel or Munich. This

Table 1 Instance size.

#Vertices	15 115
#Stops	13 941
#Edges	33 890
#Edges − #Loops	18 775
#Connections	780 042
#Trips	47 844
#Passenger	1 249 910

Table 2 Running time of our algorithm depending on the maximum delay $\Delta_\tau^{\max}$.

$\Delta_\tau^{\max}$	time
1 min	108.57 sec
2 min	109.92 sec
4 min	111.49 sec
8 min	117.32 sec
16 min	125.26 sec
32 min	136.25 sec
64 min	149.61 sec

network as well as the associated model for demand was introduced in [9]. The timetable covers roughly the traffic of one day, the earliest connection departs at 0:39 am and the last connection arrives at 2:37 am on the second day. Some key figures of the network are listed in Table 1. Since we require the transfer graph to be transitively closed, it contains a loop edge at every vertex, which increases the number of total edges significantly. Our algorithm depends on several tuning parameters that are used to adequately model passenger behavior. For our experiments, we chose the following values: the walking cost is set to $\lambda_{\text{walk}} = 2.0$, the waiting cost is set to $\lambda_{\text{wait}} = 0.5$, the transfer cost as well as the delay tolerance are set to $\lambda_{\text{trans}} = \lambda_{\Delta\max} = 300\,\text{sec}$, and the maximum delay is set to $\Delta_\tau^{\max} = 60\,\text{sec}$.

5.2 Experiments

Our first experiment evaluates the performance of our algorithm, depending on the various tuning parameters. These parameters are primarily intended to model different passenger preferences. As such they do not directly influence the computational complexity of the algorithm. In fact there is no measurable difference in running times when the parameters λ_{walk}, λ_{wait}, λ_{trans}, or $\lambda_{\Delta\max}$ are changed. However, increasing the maximum delay $\Delta_\tau^{\max}$ of the connections slightly increases the running time, as stated in Table 2. For every row in the table we repeated the assignment computation ten times and report the mean of the resulting running times. The increase in running time is caused by the computation of $\tau_{\text{arr}}^{\text{p}}(c, d \mid \text{trans})$, since more connections have a non zero probability of being the successor connection for c.

Another important tuning parameter, is the passenger multiplier λ_{mul}. Changing λ_{mul} directly influences the amount of work that has to be done, since more passengers have to be simulated. Figure 2 shows the running time of our algorithm dependent on λ_{mul}, differentiated by the phases of the algorithm. As expected the running time increases with an increasing passenger multiplier. The additional running time is mostly due to the assignment

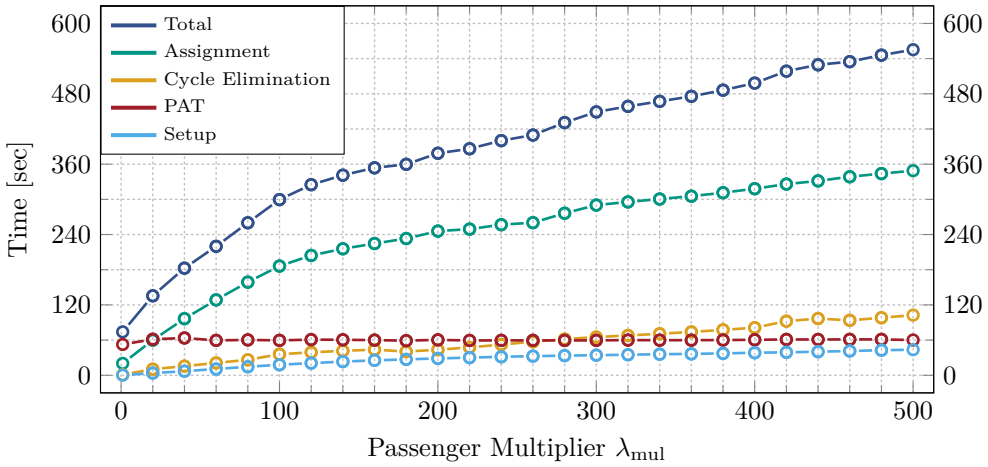

Figure 2 The running time of our algorithm depending on the passenger multiplier, differentiated by the phases of the algorithm. Changing the passenger multiplier primarily affects the assignment phase. Every measurement is the mean over the running times of ten repetitions of our algorithm.

phase. However, the running time of the assignment phase is not doubled when the number of passengers is doubled. This is the case, because an increased number of passengers leads to more passengers making the same decisions. Thus synergy effects can be used during the computation. The PAT computation is completely independent from the number of passengers, which leads to a constant running time as seen in Figure 2. The time required for the cycle elimination and the setup phase (i.e. sorting the connections and distributing the passengers by destination) increases only slightly with an increased passenger multiplier.

Next, we evaluate the performance of the parallelized version of our algorithm. For the following experiment we use a passenger multiplier of $\lambda_{\text{mul}} = 10$, since this is in most cases sufficient for an accurate result. The serial version of the algorithm has a running time of 108.57 sec. Using the parallelized version with only one thread results in a slightly increased running time of 108.92 sec. Using two threads we achieve a running time of 65.57 sec, four threads achieve 38.41 sec. As before all measurements are the mean over ten executions. Using four threads we only achieve a speed-up of 2.83, despite the fact that the computations are complete independent of each other. This could be the case, because our algorithm primarily scans through the memory. Thus, memory bandwidth could be a limiting factor.

Additionally, we compare the running time of our algorithm to VISUM, which is a commercial tool from PTV AG. On the same instance the VISUM computation took just above 30 minutes, and was parallelized using 8 threads. The VISUM assignment was computed on an Intel Core i7-6700 clocked at 3.4 GHz with 64 GiB of RAM, running Windows 10. Thus our algorithm outperforms the state-of-the-art by a factor of about 50.

Finally, we compare the quality of the assignment computed by our algorithm to the one computed by VISUM. Table 3 summarizes the results. Overall, the assignments computed by our algorithm and VISUM are quite similar. Our algorithm assigns journeys with slightly longer mean travel time, in favor of a slightly decreased number of transfers. At the same time, our algorithm assigns journeys with a higher maximum number of trips. The reason for this is that VISUM prunes all journeys with more than 6 trips, while our algorithm has no hard limit on the number of transfers. It is noticeable, that both techniques assign about 1200 passengers to a single vehicle, since both are not able to handle vehicle capacities.

Table 3 Comparison between an assignment computed by VISUM and our algorithm. We report for every quantity the minimum (min), mean, standard deviation (sd), and maximum (max) over all journeys. The figures for both assignments are quite similar. However, our assignment slightly favors journeys with fewer trips (transfers), at the disadvantage of marginal increased travel time.

	VISUM				Our Algorithm			
Quantity	min	mean	sd	max	min	mean	sd	max
Total travel time [min]	2.98	46.885	23.753	429.00	2.98	47.199	23.443	429.00
Time spent in vehicle [min]	0.02	21.059	18.796	380.00	0.02	21.231	18.749	323.97
Time spent walking [min]	2.00	22.394	5.200	149.00	2.00	22.476	5.265	149.00
Time spent waiting [min]	0.00	3.432	5.722	217.02	0.00	3.492	5.677	217.02
Trips per passenger	1.00	1.771	0.833	6.00	1.00	1.746	0.843	8.00
Connections per passenger	1.00	9.396	7.435	109.00	1.00	9.474	7.331	97.00
Passengers per connection	0.00	12.740	37.795	1 290.10	0.00	12.847	37.584	1 233.60

6 Conclusion and Future Work

In this work we presented a novel algorithmic approach to compute public transit traffic assignments. As a means of modeling realistic passenger behavior we introduced perceived arrival times. This allowed us to consider several important criteria of a journey while developing an efficient algorithm. We showed that the resulting assignment is comparable to the state-of-the-art in terms of quality. Concerning running time, our algorithm is more than an order of magnitude faster then state-of-the-art.

For future work, it would be interesting to incorporate vehicle capacities and an equilibrium model in our approach, since our experiments showed that a lack of those results in some vehicles having a very high utilization. Moreover, we would like to incorporate the cycle elimination phase into the assignment phase, such that journeys containing cycles are not assigned in the first place.

References

1 Hannah Bast, Erik Carlsson, Arno Eigenwillig, Robert Geisberger, Chris Harrelson, Veselin Raychev, and Fabien Viger. Fast routing in very large public transportation networks using transfer patterns. In *Proceedings of the 18th Annual European Symposium on Algorithms (ESA'10)*, volume 6346 of *Lecture Notes in Computer Science*, pages 290–301. Springer, 2010.

2 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. In *Algorithm Engineering – Selected Results and Surveys*, volume 9220 of *Lecture Notes in Computer Science*, pages 19–80. Springer, 2016.

3 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)*, Open-Access Series in Informatics (OASIcs), pages 42–54, 2013. doi:10.4230/OASIcs.ATMOS.2013.42.

4 Annabell Berger, Daniel Delling, Andreas Gebhardt, and Matthias Müller–Hannemann. Accelerating time-dependent multi-criteria timetable information is harder than expected. In *Proceedings of the 9th Workshop on Algorithmic Approaches for Transportation Modeling,*

Optimization, and Systems (ATMOS'09), OpenAccess Series in Informatics (OASIcs), 2009. doi:10.4230/OASIcs.ATMOS.2009.2148.

5 Daniel Delling, Thomas Pajor, and Renato F. Werneck. Round-based public transit routing. *Transportation Science*, 2014. doi:10.1287/trsc.2014.0534.

6 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.

7 Julian Dibbelt, Ben Strasser, and Dorothea Wagner. Delay-robust journeys in timetable networks with minimum expected arrival time. In *Proceedings of the 14th Workshop on Algorithmic Approaches for Transportation Modeling, Optimization, and Systems (ATMOS'14)*, OpenAccess Series in Informatics (OASIcs), 2014. doi:10.4230/OASIcs.ATMOS.2014.1.

8 Michael Patriksson. *The Traffic Assignment Problem: Models and Methods*. Courier Dover Publications, 2015.

9 Johannes Schlaich, Udo Heidl, and Regine Pohlner. Verkehrsmodellierung für die Region Stuttgart – Schlussbericht. Unpublished manuscript, 2011.

10 Yosef Sheffi. *Urban Transportation Networks*. Prentice-Hall, Englewood Cliffs, NJ, 1985.

11 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.

12 Sascha Witt. Trip-based public transit routing. In *Proceedings of the 23rd Annual European Symposium on Algorithms (ESA'15)*, Lecture Notes in Computer Science. Springer, 2015. Accepted for publication.