Integrating ULTRA and Trip-Based Routing

Jonas Sauer
Karlsruhe Institute of Technology (KIT), Germany
jonas.sauer2@kit.edu

Dorothea Wagner
Karlsruhe Institute of Technology (KIT), Germany
dorothea.wagner@kit.edu

Tobias Zündorf
Karlsruhe Institute of Technology (KIT), Germany
zuendorf@kit.edu

Abstract

We study a bi-modal journey planning scenario consisting of a public transit network and a transfer graph representing a secondary transportation mode (e.g., walking or cycling). Given a pair of source and target locations, the objective is to find a Pareto set of journeys optimizing arrival time and the number of required transfers. For public transit networks with a restricted, transitively closed transfer graph, one of the fastest known algorithms solving this bi-criteria problem is Trip-Based Routing [26]. However, this algorithm cannot be trivially extended to unrestricted transfer graphs. In this work, we combine Trip-Based Routing with ULTRA [5], a preprocessing technique that allows any public transit algorithm that requires transitive transfers to handle an unrestricted transfer graph. Since both ULTRA and Trip-Based Routing precompute transfer shortcuts in a preprocessing phase, a naive combination of the two leads to a three-phase algorithm that performs redundant work and produces superfluous shortcuts. We therefore propose a new, integrated preprocessing phase that combines the advantages of both and reduces the number of computed shortcuts by up to a factor of 9 compared to a naive combination. The resulting query algorithm, ULTRA-Trip-Based is the fastest known algorithm for the considered problem setting, achieving a speedup of up to 4 compared to the fastest previously known approach, ULTRA-RAPTOR.

2012 ACM Subject Classification Theory of computation → Shortest paths; Mathematics of computing → Graph algorithms; Applied computing → Transportation

Keywords and phrases Algorithms, Journey Planning, Multi-Modal, Public Transportation

Digital Object Identifier 10.4230/OASIcs.ATMOS.2020.4

Supplementary Material Code is available at `https://github.com/kit-algo/ULTRA-Trip-Based`.

Funding This research was funded by the DFG under grant number WA 654123-2.

Acknowledgements We thank Sascha Witt for many fruitful discussions about Trip-Based Routing.

1 Introduction

Research on algorithms for journey planning in both road and public transit networks has seen remarkable advances in recent years [3]. Many algorithms have been developed that enable efficient journey planning in either type of network, but exceedingly few of them are capable of efficient journey planning in a combined multi-modal network. Recently, the ULTRA [5] approach was introduced, which promises to extend most public transit journey planning algorithms to handle multi-modal networks. In this work we consider the combination of ULTRA and Trip-Based Public Transit Routing [26], a very efficient algorithm for public transit networks that on its own cannot handle multi-modal networks. We demonstrate that the naive combination of these two algorithms, i.e., using the output of ULTRA as input for the Trip-Based approach, indeed results in an efficient multi-modal journey planning

algorithm. However, we observe that the two algorithms can be combined on a much deeper level, as they are both based on the precomputation of shortcuts. Through careful algorithm engineering, we develop a truly integrated version of the ULTRA-Trip-Based algorithm, which significantly reduces the number of required shortcuts. Using this approach, we are able to outperform the previously fastest multi-criteria multi-modal algorithm.

Related Work. Journey planning algorithms for public transit networks can generally be divided into graph-based approaches and algorithms that operate directly on the timetable and exploit its schedule-based structure [3]. Graph-based approaches can be further subdivided into time-dependent [17, 14, 18, 19] and time-expanded [2, 4, 22] techniques. Notable examples of timetable-based approaches are RAPTOR [10, 8], which partitions the timetable into routes, Trip-Based Routing [26, 27], which operates directly on the trips in the timetable, and CSA [11, 24], which divides the trips further into elementary connections and processes them individually. Common to all these algorithms is that they only consider walking and other forms of non-schedule-based transport in the form of a restricted transfer graph, which is often required to be transitively closed. However, experiments have shown that the availability of unrestricted walking significantly reduces travel times [25, 23, 21].

Multi-modal journey planning algorithms remove this limitation, allowing the combination of public transit with arbitrary, unrestricted transfer graphs. These algorithms are usually based on an existing public transit journey planning algorithm that is interleaved with an exploration of the unrestricted transfer graph. UCCH [12] combines a time-dependent graph-based approach with Dijkstra [13] searches on a contracted transfer graph. Similarly, MCR [7] combines RAPTOR [10] with Dijkstra [13] searches on a contracted transfer graph. HLRaptor and HLCSA [21], which are based on CSA [11] and RAPTOR [10], respectively, explore the transfer graph with two-hop searches based on Hub Labeling [1]. The most recent approach is ULTRA [5], which utilizes a preprocessing step that creates shortcuts for all *intermediate* transfers, i.e., transfers between two public transit vehicles. Using these shortcuts, only initial and final transfers have to be computed at query time, which can be done very efficiently by using Bucket-CH [20, 15, 16], a technique for fast one-to-many searches on road networks. This approach can be combined with many public transit algorithms. In combination with RAPTOR, it yields the currently fastest multi-modal journey planning algorithm that can optimize travel time and number of used trips.

2 Preliminaries

Following the notation in [5], we define a public transit network as a 4-tuple $(\mathcal{S}, \mathcal{T}, \mathcal{R}, G)$ consisting of a set of *stops* $\mathcal{S}$, a set of *trips* $\mathcal{T}$, a set of *routes* $\mathcal{R}$ and a directed, weighted *transfer graph* $G = (\mathcal{V}, \mathcal{E})$. A stop $v \in \mathcal{S}$ is a location in the network where passengers can enter or exit a vehicle. A trip $T = \langle \epsilon_0, \ldots, \epsilon_k \rangle \in \mathcal{T}$ is a sequence of *stop events* performed by the same vehicle. Each of these events ϵ_i represents the vehicle of the trip stopping at a stop $v(\epsilon_i) \in \mathcal{S}$. The *arrival time* of the vehicle at this stop is denoted as $\tau_{\text{arr}}(\epsilon_i)$ and the corresponding *departure time* is $\tau_{\text{dep}}(\epsilon_i)$. We use $T[i]$ to refer to the i-th stop event in a trip T. The trips are partitioned into a set of routes $\mathcal{R}$ such that all trips of a route share the same stop sequence and no trip overtakes another along the stop event sequence. The transfer graph $G = (\mathcal{V}, \mathcal{E})$ consists of a set of *vertices* $\mathcal{V}$ with $\mathcal{S} \subseteq \mathcal{V}$, and a set of *edges* $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$. Associated with each edge $e = (v, w)$ is a *transfer time* $\tau_{\text{t}}(e)$, which denotes the time required to travel from v to w along e. The transfer graph is not required to be transitively closed, and may represent any non-schedule-based mode of transportation, such as walking or cycling.

Given a source vertex $s \in \mathcal{V}$ and a target vertex $t \in \mathcal{V}$, an *s-t-journey* represents the movement of a passenger from s to t through the public transit network. It consists of an alternating sequence of *trip legs* (i.e., subsequences of trips) and *transfers* (i.e., paths in the transfer graph). A *departure buffer time* has to be observed between consecutive transfers and trip legs. For the sake of simplicity, we do not consider them explicitly in this work. However, they are implemented as in [5]. The transfer connecting s to the first trip leg is called the *initial transfer*, whereas the *final transfer* connects the final trip leg to t. The remaining transfers, which occur between trip legs, are called *intermediate transfers*. Note that transfers may consist of empty paths.

Problem Statement. To evaluate the usefulness of an s-t-journey J, we consider its arrival time at t and the number of used trips (i.e., the number of trip legs). We say that a journey J *weakly dominates* another journey J' if J arrives no later than J' and does not use more trips than J'. Moreover, J *strongly dominates* J' if J weakly dominates J' and J has an earlier arrival time or uses fewer trips than J' (i.e., J is strictly better than J' according to at least one criterion, and no worse according to the other). A *Pareto set* is a set containing a minimal number of journeys such that every valid journey is weakly dominated by a journey in the set. Given a source vertex $s \in \mathcal{V}$, a target vertex $t \in \mathcal{V}$ and a departure time τ_{dep}, we want to compute a Pareto set of s-t-journeys that depart no later than τ_{dep}.

Algorithms. The main algorithms discussed in this work are Trip-Based Routing and ULTRA, which we briefly outline in the following. Trip-Based Routing [26] is a routing algorithm for public transit networks with a transitively closed transfer graph. It optimizes both arrival time and number of trips in a Pareto sense, as required by our problem statement. The algorithm explores the reachable trips of the network in *rounds*, where each round extends the journeys found in the previous round by another trip. Unlike RAPTOR [10], which also works in rounds, Trip-Based Routing does not maintain arrival times at stops. Instead, each round consists of scanning reachable trips in order to find transfers to the target or to other trips, which are then processed in the next round. The transfers to other trips are precomputed in a preprocessing phase by first generating all potentially relevant transfers between stop events, and then pruning unnecessary transfers in a "transfer reduction" phase.

ULTRA [5] is a preprocessing technique which enables any public transit journey planning algorithm designed for transitively closed transfer graphs to handle unlimited transfers instead. This is achieved by precomputing a small number of *transfer shortcuts* representing all intermediate transfers that are required to answer any query correctly. To this end, the preprocessing phase enumerates journeys using at most two trips, distinguishing between *candidate journeys*, which contain a potential shortcut, and *witness journeys*, which can prove irrelevance of candidates. If a witness journey is found that weakly dominates a candidate journey, the corresponding shortcut is not needed. An ULTRA query is performed by first exploring initial and final transfers via Bucket-CH [20, 15, 16], a fast one-to-many technique for road networks. Afterwards, a public transit algorithm of choice can be run on the public transit network, using the precomputed shortcuts as the transfer graph.

3 Algorithms

The Trip-Based Routing algorithm can be integrated into the generic ULTRA query framework, without any modification. However, as Trip-Based Routing on its own already requires a preprocessing step, unlike RAPTOR and CSA, this yields a three-phase algorithm: The first

phase is the ULTRA preprocessing, the second phase is the Trip-Based preprocessing, which uses the ULTRA transfer shortcuts as input, and the third phase is the ULTRA-Trip-Based query. Of these three phases, the two preprocessing steps have some parts in common. Therefore, integrating them and removing redundant parts yields a single, more elegant preprocessing step that produces fewer shortcuts.

Furthermore, the original Trip-Based query, as introduced in [26], is optimized for a use case where only a small number of stops is reachable with transfers from the source or the target. However, with unlimited transfers, we expect that almost every stop is reachable from the source and the target. We therefore restructure the query to process the huge number of possible initial and final transfers more efficiently.

3.1 Integrated Preprocessing

The preprocessing phases of ULTRA and Trip-Based Routing have many similarities, despite the fact that Trip-Based Routing requires transitively closed transfers, which ULTRA does not. Both of them compute shortcuts, which are later used to accelerate the query. However, ULTRA computes time-independent shortcuts (connecting pairs of stops), while the Trip-Based shortcuts are time-dependent (connecting pairs of stop events). This means that a shortcut which is needed at one time during the day is available at all times when using ULTRA, while Trip-Based Routing is aware that the shortcut is only needed at a certain time.

Both approaches identify unnecessary shortcuts by enumerating journeys with at most two trips in order to find witness journeys which prove that a potential shortcut is not necessary. The Trip-Based preprocessing does this in a "transfer reduction" step, after all potential shortcuts have been generated. Since this is no longer feasible with unlimited transfers, ULTRA interleaves the generation and pruning of shortcuts. Another difference is the type of journeys that are considered as witnesses. In the Trip-Based preprocessing, witness journeys must start with the same trip from which the shortcut originated, whereas the ULTRA preprocessing also considers witness journeys that start with an initial transfer. Furthermore, the Trip-Based preprocessing does not guarantee that a witness journey is found before the shortcut it could prune has already been added to the output, since this depends on the order in which the shortcuts are explored. Overall, ULTRA has more options for pruning candidate journeys, and thus produces fewer shortcuts.

Since both preprocessing phases enumerate journeys for similar purposes, we propose to integrate them and remove redundant parts. We implement this by keeping the general approach of the ULTRA journey enumeration, which can handle unlimited transfer graphs and prunes more shortcuts overall. In order to produce time-dependent shortcuts, we switch from computing shortcuts between stops to computing shortcuts between stop events. This makes the "transfer reduction" phase of the Trip-Based preprocessing obsolete. Achieving this requires some alterations to the original ULTRA preprocessing phase, which we describe in detail in the remainder of this section.

Candidate Journeys. The original ULTRA preprocessing includes an optimization that dismisses candidate journeys if their intermediate transfer was already added as a shortcut before. In the context of ULTRA, this has a significant impact because time-independent shortcuts are likely to be used multiple times during the day. However, when switching to time-dependent shortcuts, it becomes much less likely for a new candidate journey to use a previously found shortcut. Thus, the expected benefit of potentially dismissing the candidate no longer outweighs the work required to look up the shortcut. We therefore do not prune candidate journeys with already found shortcuts.

Figure 1 An example network that demonstrates how using weak domination in the ULTRA-Trip-Based preprocessing leads to missing shortcuts. Transfer edges (gray) are labeled with their travel time, while trips (colored) are labeled with $\tau_{\text{dep}} \to \tau_{\text{arr}}$. With weak domination of candidates, the preprocessing only finds two shortcuts: $(0 \to 1, 3 \to 4)$ and $(5 \to 6, 8 \to 9)$. These two shortcuts are not sufficient for finding an s-t-journey. If candidate journeys are only dismissed if they are strictly dominated by a witness, then an additional shortcut $(3 \to 4, 8 \to 9)$ is found during preprocessing. Using this shortcut, the s-t-journey $\langle\langle s\rangle, \langle 0 \to 1\rangle, \langle v, w\rangle, \langle 3 \to 4\rangle, \langle x, y\rangle, \langle 8 \to 9\rangle, \langle t\rangle\rangle$ can be computed.

Parent Pointers. In order to determine the shortcut that corresponds to a candidate journey, the ULTRA preprocessing algorithm maintains parent pointers for the stops of the candidate journeys. These parent pointers point to earlier stops within the same journey and can thus be used to find the intermediate transfer of a journey by tracing them back, starting from the last stop of the journey. Since we want to compute shortcuts between stop events instead of stops, we also change the parent pointers from stops to stop events. As in the original ULTRA preprocessing, we propagate parent pointers by assigning $\text{parent}[w] \leftarrow \text{parent}[v]$ whenever relaxing an edge (v, w) leads to an improved arrival time at w. Doing this enables an efficient retrieval of the shortcut corresponding to the intermediate transfer of a candidate journey. Assume that a candidate journey J ends at the stop t. In this case, the shortcut corresponding to the intermediate transfer of J is $(\text{parent}_1[v(\text{parent}_2[t])], \text{parent}_2[t])$, where $\text{parent}_k[v]$ is the parent for reaching v using k trips (i.e., within the k-th RAPTOR round). As before, witness journeys are distinguished from candidate journeys by assigning a special value to the parent pointers of witness journeys.

Initial Transfer and Strict Dominance. The most important modification of the algorithm is required due to the fact that the ULTRA preprocessing allows witness journeys with initial transfers (unlike Trip-Based). In combination with weak domination of candidates, this can lead to missed shortcuts between stop events, as demonstrated in Figure 1. In this example, only two shortcuts will be found: $(0 \to 1, 3 \to 4)$ and $(5 \to 6, 8 \to 9)$. However, these two shortcuts are not sufficient for finding a journey from s to t with the Trip-Based query algorithm. The algorithm will only find journeys starting at s that reach the only trip of the blue route $(0 \to 1)$ and the first trip of the yellow route $(3 \to 4)$. No further journeys can be found, since there is no transfer shortcut from the blue route to the second trip of the yellow route $(0 \to 1, 5 \to 6)$ and no transfer from the first trip of the yellow route to the red route $(3 \to 4, 8 \to 9)$. Either one of these shortcuts would be sufficient for finding a journey from s to t. We argue that adding $(3 \to 4, 8 \to 9)$ as a shortcut is preferable, since passengers using the blue route would have no reason to wait for the second trip of the yellow route if they can also continue with the first trip of the yellow route.

Before explaining the modifications that are necessary in order to find the shortcut $(3 \to 4, 8 \to 9)$, we briefly examine why this shortcut is not found by a naive combination of the ULTRA preprocessing and the Trip-Based preprocessing. For this, we consider the candidate journey $J^{\mathsf{c}} = \langle\langle w\rangle, \langle 3 \to 4\rangle, \langle x, y\rangle, \langle 8 \to 9\rangle, \langle t\rangle\rangle$, which contains the missing shortcut. During the ULTRA preprocessing, this journey is dominated by the witness journey $J = \langle\langle w\rangle, \langle 5 \to 6\rangle, \langle x, y\rangle, \langle 8 \to 9\rangle, \langle t\rangle\rangle$, hence no shortcut is added. Note that this problem

only arises when ULTRA and Trip-Based Routing are combined. When using ULTRA on its own, shortcuts connect pairs of stops instead of stop events. This means that the two shortcuts $(3 \to 4, 8 \to 9)$ and $(5 \to 6, 8 \to 9)$ between stop events are both represented with the single shortcut (x, y) between stops. Therefore, finding only one of them is sufficient. On the other hand, when using Trip-Based Routing on its own, the problem does not arise, as the Trip-Based preprocessing does not consider journeys with initial transfers. This means that the candidate journey J^c is not dominated by the witness journey J, since J requires waiting at w, which is considered to be an initial transfer. Therefore the shortcut $(5 \to 6, 8 \to 9)$ is found by the standard Trip-Based preprocessing.

We observe that the problem of missing shortcuts only occurs if a candidate journey and the corresponding witness journey are equivalent with respect to their arrival time and their number of used trips. Thus the problem can be solved by only dismissing candidate journeys that are strictly dominated by a witness (instead of being weakly dominated as in standard ULTRA). We now continue with describing how this change can be implemented within our preprocessing algorithm. Using strict dominance instead of weak dominance affects all parts of the algorithm where a new arrival time at a vertex v is discovered (i.e., during the relaxation of edges and during route scanning). Normally the label of v is only updated if the newly discovered arrival time is strictly better (earlier) than the previously found arrival time. Instead, we now also update the label of v if the following three conditions hold: First, the new arrival time at v is equivalent to the previous arrival time. Secondly, the current label of v does not correspond to a candidate journey. Thirdly, the journey that corresponds to the new arrival time is a candidate journey. These new rules for updating a label ensure that a newly found candidate journey is not implicitly dominated by a previously found journey with the same arrival time. In the case of equal arrival times, we allow that candidate journeys replace non-candidate journeys, but not vice versa. This is necessary to prevent cyclic label updates, which would otherwise lead to infinite loops.

3.2 Improved Query

We use the shortcuts computed by the combined ULTRA-Trip-Based preprocessing within a modified version of the Trip-Based query algorithm. As with the original ULTRA query, initial and final transfers are handled by performing two Bucket-CH queries. However, in contrast to the general ULTRA query, efficiently integrating the results of the Bucket-CH queries into the Trip-Based query is more involved. We provide an overview that shows how initial and final transfers are processed in our ULTRA-Trip-Based query algorithm in Algorithm 1. In the following, we describe this query algorithm in detail.

Bucket-CH Query. The first step of the algorithm (lines 1–4) is the execution of the Bucket-CH queries, which is done in the same manner as in the generic ULTRA query. In order to improve efficiency, the Bucket-CH queries are split into three parts. First, a standard CH query from s to t with departure time τ_{dep} is performed. This yields the minimal arrival time τ_{min} at the target via a direct transfer, the forward CH search space $\mathcal{V}_s$ of s, and the backward CH search space $\mathcal{V}_t$ of t. The minimal arrival time τ_{min} is ∞ if no path from s to t exists in the transfer graph. If, on the other hand, $\tau_{\mathsf{min}} < \infty$ holds, then we have found an s-t-journey with arrival time τ_{min} that uses zero trips, which we add to the result set in line 2. Afterwards, we evaluate the buckets containing vertex-to-stop transfer times for vertices in $\mathcal{V}_s$, which provides us with the arrival time $\tau_{\text{arr}}(s, v)$ for each stop v with $\tau_{\text{arr}}(s, v) \leq \tau_{\mathsf{min}}$. Similarly, we evaluate the buckets containing stop-to-vertex transfer times for vertices in $\mathcal{V}_t$, in order to obtain transfer times $\tau_{\mathsf{t}}(v, t)$ for all stops v with $\tau_{\mathsf{t}}(v, t) \leq \tau_{\mathsf{min}} - \tau_{\text{dep}}$.

Algorithm 1 ULTRA-Trip-Based query.

Input: Public transit network $(\mathcal{S}, \mathcal{T}, \mathcal{R})$, transfer shortcut graph $G^{\mathsf{t}} = (\mathcal{V}^{\mathsf{t}}, \mathcal{E}^{\mathsf{t}})$, Bucket-CH of the original transfer graph G, source vertex s, departure time τ_{dep}, and target vertex t
Output: All Pareto-optimal journeys from s to t for departure time τ_{dep}

1 $(\tau_{\mathsf{min}}, \mathcal{V}_s, \mathcal{V}_t) \leftarrow$ Run a CH query from s to t with departure time τ_{dep}
2 **if** $\tau_{\mathsf{min}} < \infty$ **then** $\mathcal{J} \leftarrow \{(\tau_{\text{arr}}(s,t), 0)\}$
3 $\tau_{\text{arr}}(s, \cdot) \leftarrow$ Evaluate the vertex-to-stop buckets for vertices in $\mathcal{V}_s$
4 $\tau_{\mathsf{t}}(\cdot, t) \leftarrow$ Evaluate the stop-to-vertex buckets for vertices in $\mathcal{V}_t$
5 **for each** $v \in \mathcal{V}_s$ **do**
6 $\quad \mathcal{R}' \leftarrow \mathcal{R}' \cup \{$Routes from $\mathcal{R}$ that contain $v\}$
7 **for each** $R \in \mathcal{R}'$ **do**
8 $\quad T_{\mathsf{min}} \leftarrow \infty$
9 $\quad$ **for** i from 0 to $|R|$ **do**
10 $\quad\quad v \leftarrow i$-th stop of R
11 $\quad\quad$ **if** $\tau_{\text{arr}}(s,v) \geq \tau_{\mathsf{min}}$ **then continue**
12 $\quad\quad$ **if** $T_{\mathsf{min}} = \infty$ **then**
13 $\quad\quad\quad T_{\mathsf{min}} \leftarrow$ Binary search: First $T \in R$ departing from v after $\tau_{\text{arr}}(s,v)$
14 $\quad\quad$ **else**
15 $\quad\quad\quad$ **while** the trip before T_{min} in R departs from v after $\tau_{\text{arr}}(s,v)$ **do**
16 $\quad\quad\quad\quad T_{\mathsf{min}} \leftarrow$ The trip before T_{min} in R
17 $\quad\quad\quad\quad$ **if** T_{min} is the first trip in R **then break**
18 $\quad\quad$ **if** $T_{\mathsf{min}} \neq \infty$ **and** $\tau_{\text{dep}}(T_{\mathsf{min}}[i]) \geq \tau_{\text{arr}}(s,v)$ **then** Enqueue$(T_{\mathsf{min}}, i, Q_1)$
19 $\quad\quad$ **if** T_{min} is the first trip in R **then break**
20 $n \leftarrow 1$
21 **while** Q_n is not empty **do**
22 $\quad$ **for each** $(T, j, k) \in Q_n$ **do**
23 $\quad\quad$ **for** i from j to k **do**
24 $\quad\quad\quad$ **if** $\tau_{\text{arr}}(T[i]) \geq \tau_{\mathsf{min}}$ **then break**
25 $\quad\quad\quad$ **if** $\tau_{\text{arr}}(T[i]) + \tau_{\mathsf{t}}(v(T[i]), t) < \tau_{\mathsf{min}}$ **then**
26 $\quad\quad\quad\quad \tau_{\mathsf{min}} \leftarrow \tau_{\text{arr}}(T[i]) + \tau_{\mathsf{t}}(v(T[i]), t)$
27 $\quad\quad\quad\quad \mathcal{J} \leftarrow$ Pareto set of $\mathcal{J} \cup \{(\tau_{\mathsf{min}}, n)\}$
28 $\quad$ **for each** $(T, j, k) \in Q_n$ **do**
29 $\quad\quad$ **for** i from j to k **do**
30 $\quad\quad\quad$ **if** $\tau_{\text{arr}}(T[i]) \geq \tau_{\mathsf{min}}$ **then break**
31 $\quad\quad\quad$ **for each** $(T[i], T'[i']) \in \mathcal{E}^{\mathsf{t}}$ **do**
32 $\quad\quad\quad\quad$ Enqueue(T', i', Q_{n+1})
33 $\quad n \leftarrow n + 1$

Initial Transfer Evaluation. In the second step of the algorithm (lines 5–19), we evaluate which trips of the public transit network are reachable by an initial transfer. In the original Trip-Based query [26], this is done by iterating over all stops that are reachable via an initial transfer. For each such stop v and each route R visiting v, the algorithm identifies the earliest trip of R that can be entered at v after taking the initial transfer. This approach is efficient as long as the number of stops reachable via an initial transfer is small. However, in a scenario with unlimited transfers, where almost all stops are reachable by initial transfers, consecutive stops of a route often share the same earliest reachable trip. This can cause

cause the same trip to be found multiple times, leading to redundant work. To avoid this, we propose a new approach for evaluating the initial transfers, which is based on two steps of the RAPTOR algorithm: collecting updated routes and scanning routes.

We start by collecting all routes which contain a stop that is reachable by an initial transfer from the source in lines 5 and 6. This is analogous to collecting routes that contain updated stops at the beginning of a RAPTOR round. We proceed by scanning the routes we have collected. The goal of this step is to find for each stop v within a route R the first trip T_{min} of the route R that can be boarded at v. We achieve this by processing the stops v in the order they appear in R, while gradually updating T_{min} at the same time.

Let v be the next stop to be processed while scanning the route R. If we have not found a reachable trip for any of the previous stops in R (i.e., $T_{\mathsf{min}} = \infty$), then we use a binary search to find the first trip in R that can be boarded at v (line 13). Otherwise, we assume that the earliest reachable trip at v is probably not much earlier than the previously found trip T_{min}. Thus, we perform a linear search, starting from T_{min}, to find this trip (lines 15–17). Note that in cases where the earliest reachable trip at v departs after T_{min}, the linear search will not find it. However, this is not a problem, since it is preferable to enter T_{min} at a previous stop, in this case. After we have found the earliest trip reachable at v, we add it to the queue of trips that have to be scanned in line 18. Finally, we can stop searching for earlier trips if T_{min} is already the earliest trip in the route R.

The original Trip-Based query also collects final transfers to the target before continuing with the trip scanning step. These are used in the trip scanning step to efficiently identify the stops in the trip from which the target can be reached. In the presence of unlimited transfers, this is no longer worth the effort, since the target can be reached from almost all stops. We therefore skip this step and evaluate final transfers on the fly while scanning trips. Unfortunately, skipping the evaluation of initial transfers is not an option, as we need to evaluate them in order to know which trips have to be scanned.

Trip Scanning. The third and last step of the query algorithm (lines 20–33) is the trip scanning phase, which is mostly identical to the original Trip-Based query algorithm. It is organized in rounds, where the n-th round scans the trips that have previously been collected in Q_n, which correspond to journeys that start at s and contain n trips. For each of these trips, the queue also contains indices i and j, which indicate the first and last stop event of the trip that have to be scanned, respectively. While scanning the i-th stop event of the trip T, the algorithm checks whether a final transfer from the i-th stop of the trip T to the target exists in line 24. If such a transfer exists and if this transfer can be used to improve the earliest known arrival time τ_{min} at the target, then the algorithm has found a new Pareto-optimal journey. In this case, τ_{min} is updated and the newly found journey is added to the result set $\mathcal{J}$. If $\mathcal{J}$ already contains a journey with n trips (note that a Pareto set can only contain one such journey), this journey is replaced.

After the final transfers have been evaluated, we continue with relaxing the precomputed transfer edges in $\mathcal{E}^{\mathsf{t}}$ that start at the stop event $T[i]$. Each of these edges provides us with a new trip T' that can be used to extend the current journey. Thus, the trip T' (together with the index i' of the first stop event in T' that can be reached) is added to the queue Q_{n+1} of trips that have to be scanned in the next round.

Note that we scan the trips in Q_n twice. We only evaluate final transfers during the first scan and use a separate second scan to relax transfer shortcuts. We do this for two reasons: First, separating the two scans improves memory locality. Secondly, we improve τ_{min} throughout the first scan, which enables better pruning in line 30 of the second scan.

Table 1 Sizes of the public transit networks and the accompanying transfer graphs which we consider in this work. Additionally, we report the number of edges in a transitively closed transfer graph that we use to compare our multi-modal algorithm with pre-existing uni-modal algorithms.

	Stuttgart	London	Switzerland	Germany
Stops	13 583	20 595	25 125	244 055
Routes	12 350	2 107	13 785	231 084
Trips	91 298	125 436	350 006	2 387 292
Stop events	1 561 912	4 970 428	4 686 865	48 495 066
Transfer graph vertices	1 166 593	183 025	603 691	6 872 105
Transfer graph edges	3 680 930	579 888	1 853 260	21 372 360
Transitive graph edges	945 514	3 755 200	2 639 402	23 880 322

Enqueueing Trips. The enqueue operation, which is used to add trips to the queues in lines 18 and 32, is identical to the enqueue operation of the original Trip-Based query [26]. Internally, it maintains an index k for every trip T in the network. This index marks the first stop event of the trip that has already been scanned and is initialized as $|T|$. When invoking Enqueue(T, i, Q_n), this index is used to add the triple (T, i, k) to the queue Q_n. Afterwards, k is decreased to $i - 1$ for this trip and all later trips of the route.

Data Structures and Memory Layout. In order to achieve the optimal performance possible for the query algorithm, it is quite important that a streamlined memory layout is used. To this end, we implement the FIFO queues Q_n using dynamic arrays. This enables an efficient enqueue operation and efficient scanning of the entries in Q_n. The edges $\mathcal{E}^{\mathsf{t}}$ are also stored in an array, such that edges $(T[i], T_a[j])$ and $(T[i], T_b[k])$, which start at the same stop event $T[i]$, occupy consecutive memory locations. Moreover, the section of this array that contains edges starting with the stop event $T[i]$ is directly in front of the section that contains edges starting with the stop event $T[i + 1]$. Finally, we observe that we only need access to the arrival time $\tau_{\text{arr}}(T[i])$ and the stop $v(T[i])$ of the stop events $T[i]$ during the trip scanning step. Thus we store these values separately from the departure time $\tau_{\text{dep}}(T[i])$ of the stop event, which improves memory locality.

4 Experiments

All algorithms were implemented in C++17 compiled with GCC version 8.2.1 and optimization flag -O3. All experiments were conducted on a machine with two 8-core Intel Xeon Skylake SP Gold 6144 CPUs clocked at 3.5 GHz, with a turbo frequency of 4.2 GHz, 192 GiB of DDR4-2666 RAM, and 24.75 MiB of L3 cache.

Benchmark Data. We evaluated our algorithms on the transportation networks of Stuttgart, London, Switzerland, and Germany. The Stuttgart network was previously used in [6]. The public transit timetable of London has been sourced from Transport for London[1] and was previously used to evaluate the original Trip-Based query algorithm [26] as well as in [9, 7].

[1] `https://data.london.gov.uk`

Table 2 An overview of the ULTRA-Trip-Based preprocessing results. We compare a basic sequential preprocessing approach with our improved integrated preprocessing. All computations were performed in parallel using 16 threads. Times are formated as h:m:s.

	Stuttgart	London	Switzerland	Germany
Shortcuts (sequential)	25 865 892	58 301 120	58 807 528	1 072 750 574
Shortcuts (integrated)	3 900 258	19 856 062	11 646 572	121 676 520
Time (sequential)	4:40	19:15	9:16	7:54:13
Time (integrated)	5:11	22:24	10:04	9:16:15

The Switzerland network was extracted from a publicly available GTFS feed[2]. Besides other works, it was previously used to evaluate ULTRA-RAPTOR, which is currently the fastest multi-modal query algorithm [5]. Lastly, the Germany network, which is the largest of our four networks, has previously been used to evaluate both Trip-Based Routing [26] and ULTRA [5]. For all four instances, we combined the public transit networks with transfer graphs representing walking and cycling that were extracted from OpenStreetMap[3]. In order to obtain travel times, we assumed an average walking speed of 4.5 km/h and an average cycling speed of 20 km/h. An overview of the resulting network sizes is given in Table 1.

4.1 Preprocessing

In this section we evaluate our novel integrated ULTRA-Trip-Based preprocessing. For this, we compare it to the naive sequential combination of ULTRA and the Trip-Based preprocessing. Furthermore, we analyze the structure of the computed shortcuts.

Comparing Sequential and Integrated Preprocessing. An overview of the results obtained by both preprocessing variants is given in Table 2. Here, rows labeled with *(integrated)* refer to our new integrated preprocessing approach, while rows labeled with *(sequential)* refer to the naive sequential approach, i.e., using the output of the standard ULTRA preprocessing as input for the Trip-Based preprocessing algorithm. The results show that using our novel integrated preprocessing leads to a significant reduction in the amount of computed shortcuts. This effect is weakest for the London network, where the number of shortcuts decreases only by a factor of 3. For our largest network (i.e., the Germany network) the sequential approach produces over 1 billion shortcuts while the integrated approach only leads to 121 million shortcuts, which corresponds to a reduction factor of almost 9. The cost for this reduction in the number of shortcuts is an increased running time of the preprocessing algorithm. However, in comparison to the significantly decreased number of shortcuts, the running time overhead is only minor. For our four test networks, the increase in preprocessing time ranges from 8% for the Switzerland network to 17% for the Germany network.

Note that all time measurements reported in Table 2 were obtained by parallel execution with 16 threads. It has been shown before that both the ULTRA preprocessing and the Trip-Based preprocessing are well suited for parallel execution [5, 26]. This also applies to our new preprocessing algorithm. As an example, we have performed the single-threaded preprocessing on the Switzerland network, where we measured running times of 1:48:55 for

[2] `https://gtfs.geops.ch/`
[3] `https://download.geofabrik.de/`

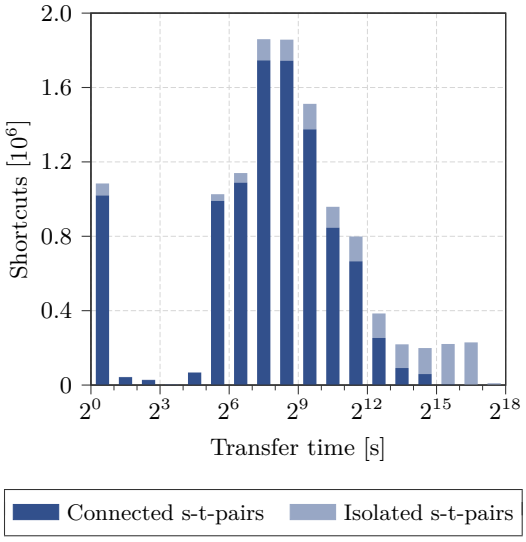

Figure 2 Histogram of the shortcuts computed for the Switzerland network by the integrated ULTRA-Trip-Based variant. The bar between 2^i and 2^{i-1} depicts the number of shortcuts with a transfer time in the interval $[2^i, 2^{i-1})$. An exception is the first bar, which also contains shortcuts with a transfer time of less than a second.

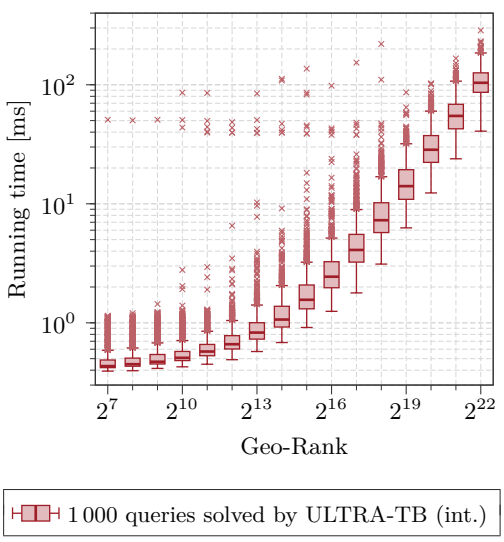

Figure 3 Query times of the integrated variant of the ULTRA-Trip-Based algorithm depending on the geo-rank. We evaluated 1 000 random vertex-to-vertex queries on the Germany network for every geo-rank. The results show that the running time greatly depends on the query distance.

the sequential approach and 2:11:16 for the integrated approach. This corresponds to a speed-up factor of 11.8 and 13.0 respectively, which matches the speed-ups observed for the ULTRA preprocessing and the Trip-Based preprocessing.

Shortcut Structure. For the original ULTRA preprocessing, it has been observed that most of the shortcuts that were computed for the Switzerland network have a transfer time of over one hour [5]. The main reason for this are candidate journeys between stops that are not connected in the transfer graph. This led to the hypothesis that most of the ULTRA shortcuts are only required by a few special journeys and that they are only relevant at a few times during a day. Given our new ULTRA-Trip-Based shortcuts (which connect stop events instead of stops) we can analyze the distribution of shortcut travel times more thoroughly. A shortcut of the original ULTRA that is used multiple times throughout a day leads to several ULTRA-Trip-Based shortcuts since they connect stop events, which occur at a fixed point in time. Thus, the number of ULTRA-Trip-Based shortcuts with a certain travel time reflects more accurately how frequently these shortcuts are required.

Figure 2 shows the number of shortcuts computed by our integrated preprocessing for the Switzerland network broken down by their travel time. We observe that most shortcuts have a travel time between 2 minutes ($\approx 2^7$ s) and 17 minutes ($\approx 2^{10}$ s). This is quite different from the original ULTRA, where most shortcuts have a travel time of more than one hour. We can therefore conclude that long shortcuts are indeed only rarely required. Furthermore, we observe that the fraction of shortcuts that are added due to candidate journeys between vertices that are not connected in the transfer graph (light blue) is much lower when using the ULTRA-Trip-Based preprocessing instead of the ULTRA preprocessing.

Table 3 Query performance for Trip-Based Routing, ULTRA-Trip-Based (ULTRA-TB, with sequential and integrated preprocessing), and ULTRA-RAPTOR. Query times are divided into phases: the Bucket-CH query (B-CH), the initial transfer evaluation (Initial), and the scanning of trips (Scan). All results are averaged over 10 000 random queries. Note that Trip-Based (marked with *) only supports stop-to-stop queries with transitive transfers. The other three algorithms support vertex-to-vertex queries on the full graph, and have been evaluated for this query type.

Network	Algorithm	Full graph	Scans [k]		Time [ms]			
			Trips	Shortcuts	B-CH	Initial	Scan	Total
Stuttgart	Trip-Based*	○	11.49	257.09	0.01	0.03	2.04	2.09
	ULTRA-TB (seq.)	●	25.05	1 528.56	1.41	0.92	5.99	8.33
	ULTRA-TB (int.)	●	17.02	218.41	1.35	0.81	2.38	4.55
	ULTRA-RAPTOR	●	–	–	1.38	–	–	10.50
London	Trip-Based*	○	22.75	1 376.26	0.01	0.05	6.10	6.16
	ULTRA-TB (seq.)	●	34.09	1 545.15	0.91	0.80	7.47	9.19
	ULTRA-TB (int.)	●	24.69	450.50	0.90	0.70	4.05	5.66
	ULTRA-RAPTOR	●	–	–	0.93	–	–	7.55
Switzerland	Trip-Based*	○	23.80	757.47	0.01	0.04	5.64	5.70
	ULTRA-TB (seq.)	●	36.46	1 551.14	1.09	1.15	7.18	9.44
	ULTRA-TB (int.)	●	23.48	238.12	1.07	1.03	3.19	5.32
	ULTRA-RAPTOR	●	–	–	1.25	–	–	14.45
Germany	Trip-Based*	○	337.49	16 116.64	0.01	0.05	116.14	116.21
	ULTRA-TB (seq.)	●	439.35	38 092.34	25.34	18.96	151.35	195.67
	ULTRA-TB (int.)	●	204.23	3 149.87	26.12	19.13	46.38	91.65
	ULTRA-RAPTOR	●	–	–	25.68	–	–	415.17

4.2 Queries

We continue by evaluating the query performance of our algorithm. To this end, we analyze how query times depend on the query distance. Furthermore, we compare our approach to the fastest multi-modal query algorithm that currently exists, namely ULTRA-RAPTOR.

Impact of the Query Distance. In order to assess the impact that the distance of a query has on the running time of our algorithm, we use geo-rank queries, which are commonly used for this purpose [26, 24]. For a geo-rank query, the source vertex is picked uniformly at random among all vertices in the network. Afterwards, all vertices are sorted by their beeline distance from the source vertex. The vertex with index i in this order is then the target of the geo-rank query for rank i. The query times of 1 000 geo-rank queries performed on the Germany network are aggregated in Figure 3. We observe that the query time of our algorithm strongly correlates with the geo-rank of the query, with local queries being more than two orders of magnitude faster than long-range queries. Furthermore, we see that some queries require a running time that is significantly longer than the median running time (independently of the geo-rank). However, in comparison to running times of the original Trip-Based query as reported in [26], we observe that our algorithm has much fewer outliers. The extreme outliers can be attributed to queries where the source vertex is

located in particularly sparse parts of the network. The reason for this is a poor correlation between geo-rank and actual distance in sparse parts of the network. Thus, a query can be a long-range query despite having a low geo-rank. An example for this are the queries with geo-rank 2^7, which corresponds to a distance of less than 1 km for most source-target pairs. However, the source of the query that took about 500 ms is located in Prague, while its target is located in Germany, which is more than 80 km away.

Overall Query Performance. Table 3 presents average query performance (based on 10 000 random queries) for all four networks. For comparison, we also include the original Trip-Based algorithm, which cannot solve multi-modal queries and was therefore evaluated using a different set of random queries. Overall, we see that our improved Trip-Based query in combination with the integrated preprocessing yields the lowest query times, independent of the network. For the Germany network, our new algorithm is more than 4 times faster than ULTRA-RAPTOR, which previously was the fastest algorithm for multi-modal journey planning. For most networks, ULTRA-Trip-Based is even faster than the original Trip-Based algorithm, despite the fact that ULTRA-Trip-Based handles a large, realistic transfer graph while Trip-Based can only consider transitively closed transfer graphs. The reason for this is the reduced size of the search space due to better pruning of the shortcuts and the existence of faster journeys in a network with unlimited transfers. The only exception to this is the Stuttgart network, which has the fewest trips, but the second-largest transfer graph out of our four networks. Thus, the comparison with an algorithm that cannot handle unlimited transfer graphs, such as Trip-Based, is particularly unfair for the Stuttgart network.

In addition to the total query time, we also report time measurements for the three phases of the Trip-Based query algorithm in Table 3. Analyzing these measurements, we see that the Bucket-CH query and the initial transfers evaluation take a non-negligible fraction of the total query running time. Furthermore, we observe that using the integrated preprocessing mainly affects the trip scanning phase of the algorithm. This was expected, as the preprocessing does not affect initial transfers, but only intermediate transfers, which are handled in the trip scanning phase. Moreover, we observe that the integrated preprocessing not only reduces the number of shortcuts that are scanned during the query, but also the number of trips.

5 Conclusion

In this work, we proposed a multi-modal variant of Trip-Based Routing, one of the fastest known journey planning algorithms for public transit networks. We achieved this by combining it with ULTRA, a preprocessing technique that replaces the transfer graph with a small number of transfer shortcuts. A naive combination of the two, which uses the output of ULTRA as input for the preprocessing phase of Trip-Based Routing, leads to many unnecessary shortcuts. Therefore, we proposed a new, integrated preprocessing phase which produces up to 9 times fewer shortcuts than the naive sequential preprocessing, at only a slight increase in preprocessing time. By analyzing the produced shortcuts, we were able to confirm a hypothesis from the original ULTRA publication that long intermediate transfers are only rarely required, even though they are responsible for a large share of the time-independent shortcuts. The resulting query algorithm, ULTRA-Trip-Based, is up to 4 times faster than the fastest previously known multi-modal algorithm for bi-criteria optimization, ULTRA-RAPTOR.

References

1 Ittai Abraham, Daniel Delling, Andrew V Goldberg, and Renato F Werneck. A Hub-Based Labeling Algorithm for Shortest Paths in Road Networks. In *International Symposium on Experimental Algorithms*, pages 230–241. Springer, 2011.

2 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 *European Symposium on Algorithms*, pages 290–301. Springer, 2010.

3 Hannah Bast, Daniel Delling, Andrew Goldberg, Matthias Müller-Hannemann, Thomas Pajor, Peter Sanders, Dorothea Wagner, and Renato F Werneck. Route Planning in Transportation Networks. In *Algorithm engineering*, pages 19–80. Springer, 2016.

4 Hannah Bast, Jonas Sternisko, and Sabine Storandt. Delay-robustness of Transfer Patterns in Public Transportation Route Planning. In *ATMOS-13th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems-2013*, volume 33, pages 42–54. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2013.

5 Moritz Baum, Valentin Buchhold, Jonas Sauer, Dorothea Wagner, and Tobias Zündorf. UnLimited TRAnsfers for Multi-Modal Route Planning: An Efficient Solution. In *27th Annual European Symposium on Algorithms (ESA 2019)*, volume 144 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 14:1–14:16, Dagstuhl, Germany, 2019. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

6 Lars Briem, Sebastian Buck, Holger Ebhart, Nicolai Mallig, Ben Strasser, Peter Vortisch, Dorothea Wagner, and Tobias Zündorf. Efficient Traffic Assignment for Public Transit Networks. In *LIPIcs-Leibniz International Proceedings in Informatics*, volume 75. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2017.

7 Daniel Delling, Julian Dibbelt, Thomas Pajor, Dorothea Wagner, and Renato F. Werneck. Computing Multimodal Journeys in Practice. In *Proceedings of the 12th International Symposium on Experimental Algorithms (SEA'13)*, volume 7933 of *Lecture Notes in Computer Science*, pages 260–271. Springer, 2013.

8 Daniel Delling, Julian Dibbelt, Thomas Pajor, and Tobias Zündorf. Faster Transit Routing by Hyper Partitioning. In *17th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2017)*, volume 59 of *OpenAccess Series in Informatics (OASIcs)*, pages 8:1–8:14, Dagstuhl, Germany, 2017.

9 Daniel Delling, Thomas Pajor, and Renato F. Werneck. Round-Based Public Transit Routing. In *Proceedings of the 14th Workshop on Algorithm Engineering and Experiments (ALENEX'12)*, pages 130–140. SIAM, 2012.

10 Daniel Delling, Thomas Pajor, and Renato F Werneck. Round-based Public Transit Routing. *Transportation Science*, 49(3):591–604, 2014.

11 Julian Dibbelt, Thomas Pajor, Ben Strasser, and Dorothea Wagner. Intriguingly Simple and Fast Transit Routing. In *International Symposium on Experimental Algorithms*, pages 43–54. Springer, 2013.

12 Julian Dibbelt, Thomas Pajor, and Dorothea Wagner. User-Constrained Multimodal Route Planning. *ACM Journal of Experimental Algorithmics*, 19:3.2:1–3.2:19, April 2015.

13 Edsger W Dijkstra. A Note on Two Problems in Connexion with Graphs. *Numerische mathematik*, 1(1):269–271, 1959.

14 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.

15 Robert Geisberger, Peter Sanders, Dominik Schultes, and Daniel Delling. Contraction Hierarchies: Faster and Simpler Hierarchical Routing in Road Networks. In *Proceedings of the 7th Workshop on Experimental Algorithms (WEA'08)*, volume 5038 of *Lecture Notes in Computer Science*, pages 319–333. Springer, June 2008.

16 Robert Geisberger, Peter Sanders, Dominik Schultes, and Christian Vetter. Exact Routing in Large Road Networks Using Contraction Hierarchies. *Transportation Science*, 46(3):388–404, August 2012.

17 Kalliopi Giannakopoulou, Andreas Paraskevopoulos, and Christos Zaroliagis. Multimodal Dynamic Journey-Planning. *Algorithms*, 12(10):213, 2019.

18 Jan Hrnčíř and Michal Jakob. Generalised Time-Dependent Graphs for Fully Multimodal Journey Planning. In *16th International IEEE Conference on Intelligent Transportation Systems (ITSC 2013)*, pages 2138–2145. IEEE, 2013.

19 Dominik Kirchler. *Efficient Routing on Multi-Modal Transportation Networks*. PhD thesis, Ecole Polytechnique X, 2013.

20 Sebastian Knopp, Peter Sanders, Dominik Schultes, Frank Schulz, and Dorothea Wagner. Computing Many-to-Many Shortest Paths Using Highway Hierarchies. In *Proceedings of the 9th Workshop on Algorithm Engineering and Experiments (ALENEX'07)*, pages 36–45. SIAM, 2007.

21 Duc-Minh Phan and Laurent Viennot. Fast Public Transit Routing with Unrestricted Walking through Hub Labeling. In *Proceedings of the Special Event on Analysis of Experimental Algorithms (SEA^2)*, Lecture Notes in Computer Science. Springer, 2019.

22 Evangelia Pyrga, Frank Schulz, Dorothea Wagner, and Christos Zaroliagis. Efficient Models for Timetable Information in Public Transportation Systems. *ACM Journal of Experimental Algorithmics*, 12(2.4):1–39, 2008.

23 Jonas Sauer. Faster Public Transit Routing with Unrestricted Walking. Master's thesis, Karlsruhe Institute of Technology, April 2018.

24 Ben Strasser and Dorothea Wagner. Connection Scan Accelerated. In *2014 Proceedings of the Sixteenth Workshop on Algorithm Engineering and Experiments (ALENEX)*, pages 125–137. SIAM, 2014.

25 Dorothea Wagner and Tobias Zündorf. Public Transit Routing with Unrestricted Walking. In *17th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2017)*. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2017.

26 Sascha Witt. Trip-Based Public Transit Routing. In *23th Annual European Symposium on Algorithms (ESA 2015)*, pages 1025–1036. Springer, 2015.

27 Sascha Witt. Trip-Based Public Transit Routing Using Condensed Search Trees. In *16th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2016)*, volume 54 of *OpenAccess Series in Informatics (OASIcs)*, pages 10:1–10:12, Dagstuhl, Germany, 2016. Schloss Dagstuhl–Leibniz-Zentrum für Informatik.