Fast and Memory Efficient Multimodal Journey Planning with Delays *

Denys Katkalo[1], Andrii Rohovyi[2], Toby Walsh[2]

[1]Igor Sikorsky Kyiv Polytechnic Institute, Kyiv, Ukraine
[2] School of Computer Science and Engineering, University of New South Wales, Sydney, Australia
katkalo.denys@lll.kpi.ua, {a.rohovyi, t.walsh}@unsw.edu.au

Abstract

State-of-the-art multimodal journey-planning algorithms, such as ULTRA, have recently been adapted to account for delays. In this work, we extend this approach to be more memory-efficient, faster, and accurate. We also adapt this framework to other state-of-the-art algorithms, like CSA and RAPTOR. We demonstrate a speedup of 1.9–$4.2\times$ over existing algorithms in the single-objective search (earliest arrival time). In the bicriteria setting, we achieve competitive speedup results but greater accuracy. We also find that our method scales much better as the delay buffer increases.

Introduction

Our work targets two tasks in multimodal public transit journey planning: the earliest arrival time at a target stop (*single-objective*) and the Pareto set over arrival time and number of trips (*bicriteria*), under real-time timetable updates with unrestricted walking transfers. Real timetables admit delays bounded above by an input parameter Δ, the *delay buffer*; the challenge is to answer such queries correctly under any delay scenario in $[0, \Delta]$.

In multimodal journey planning with unrestricted transfer distances, the state-of-the-art framework ULTRA (Potthoff and Sauer 2022; Baum et al. 2019) (*UnLimited TRAnsfers*) precomputes shortcut transfers that allow fast queries with unlimited transfer distances, but this technique involves heavy precomputation. Its delay-aware extension, Delay-ULTRA (D-ULTRA) (Bez and Sauer 2024; Sauer 2024), annotates each shortcut with a delay interval so that shortcuts can be activated or deactivated as delays are revealed. However, D-ULTRA operates exclusively at the level of individual vehicle events and targets only Trip-Based (TB) routing (Witt 2015).

In this work, we project D-ULTRA's event-level shortcuts onto the stops they connect, producing a much smaller stop-level shortcut set. This transformation makes ULTRA-CSA (Dibbelt et al. 2018) and ULTRA-RAPTOR (Delling, Pajor, and Werneck 2012) usable in the delay setting alongside D-ULTRA-TB, and yields three improvements.

Memory efficiency. The stop-level projection compresses the shortcut set by 48–$180\times$ in edge count and 141–$375\times$ in memory (from gigabytes to megabytes), because many event-level shortcuts between distinct events at the same pair of stops collapse into a single edge.

Speed. CSA with stop-level shortcuts (D-ULTRA-CSA) achieves speedups of 1.9–$4.2\times$ over D-ULTRA-TB in the single-objective setting and up to $12.2\times$ over the MR baseline (Delling et al. 2013). In the bicriteria setting, D-ULTRA-RAPTOR with stop-level shortcuts is competitive with D-ULTRA-TB on both networks tested. The advantage grows with the delay buffer: increasing Δ grows the event-level set by about $14.7\times$ and slows D-ULTRA-TB by $2.3\times$, while the stop-level set grows only $4.6\times$ and query times increase by at most 20%.

Accuracy. D-ULTRA-TB misses optimal journeys on both networks at every delay buffer tested. With stop-level shortcuts and replacement, our algorithms produce zero errors on all configurations.

Related Work

Transit routing algorithms. Public transit routing has evolved from Dijkstra-based algorithms to specialised approaches such as RAPTOR (Delling, Pajor, and Werneck 2012), CSA (Dibbelt et al. 2018), TB-routing (Witt 2015, 2016), and Transfer Patterns (Bast et al. 2010). Many of these limit transfer distance, hampering integration of secondary modes such as e-scooters, bikes, or taxis. For multimodal pathfinding with unlimited transfers algorithms fall into two categories: those without precomputation, which can work directly on delayed timetables, and those that require precomputation.

For exact delay-aware routing without shortcut precomputation, MR (Delling et al. 2013) computes Pareto-optimal journeys by running RAPTOR combining with Dijkstra (Dijkstra 1959) search on the transfer relaxation phase. It can be applied directly to a delayed timetable but is relatively slow. An alternative is the Hub Labeling (HL) version of RAPTOR (Phan and Viennot 2019; Abraham et al. 2011; Delling, Goldberg, and Werneck 2016), which is often omitted from comparison analyses (Bez and Sauer 2024) due to its marginal improvement over MR, but it still produces faster bicriteria responses than MR (Sauer 2024). There is

*Authors are listed in alphabetical order.

also a single-objective CSA adaptation that works with HL, but to our knowledge it has not previously been compared against MR on the same machine and query set. We address this gap.

A faster single-objective alternative that requires no preprocessing is classical Time-Dependent Dijkstra (TD-Dijkstra) (Rohovyi, Stuckey, and Walsh 2025; Katkalo, Rohovyi, and Walsh 2026), which requires dominated-connection filtering that can produce incorrect results on networks with buffer times. Transfer Aware Dijkstra (TAD) (Katkalo, Rohovyi, and Walsh 2026) addresses this limitation by replacing per-edge relaxation with trip-level scanning, providing exact answers on networks with or without buffer times. MR, HL-RAPTOR, HL-CSA, TD-Dijkstra and TAD serve as ground-truth baselines in our experiments, since they work directly on the delayed timetable.

The dynamic-replanning formulation of Abuaisha et al. (2025) handles delays in transit settings, but it restricts walking distance by assuming a transitively closed transfer graph. Wagner and Zündorf (2017) showed this restriction is not benign: on the German network, median nighttime travel times differ by up to two hours between restricted and unrestricted variants, and 75% of daytime queries return suboptimal paths. Lifting it while keeping query times competitive motivates the ULTRA family.

Transfer-graph acceleration and shortcut precomputation. We use Contraction Hierarchies (CH) (Geisberger et al. 2012), Core-CH (Bauer et al. 2010; Baum et al. 2019), Bucket-CH (Knopp et al. 2007), and Hub Labeling (HL) (Abraham et al. 2011; Delling, Goldberg, and Werneck 2016) for acceleration search on transfer graph. Full definitions appear in the Preliminaries. For shortcut precomputation in multimodal pathfinding, ULTRA (Baum et al. 2019; Potthoff and Sauer 2022) is the leading solution. Bez's bachelor thesis (2020) initiated the delay-tolerant line, introducing DB-ULTRA: *unannotated stop-level shortcuts* valid for any delay scenario within a fixed buffer Δ, evaluated with ULTRA-RAPTOR and ULTRA-CSA on Switzerland and Germany. The thesis observed rapid growth in shortcut count under the Delay-All model (over twelve million at $\Delta = 30$ min on Switzerland) and proposed two directions to address it: annotating shortcuts with delay sub-intervals, and adapting the framework to Trip-Based Routing (Witt 2015). Bez (2024) and Sauer (2024) subsequently combined both into event-level Trip-Based shortcuts with delay annotations, the algorithm we benchmark against. We propose a third direction: rather than re-deriving shortcuts natively at the stop level, we project the event-level shortcut set onto the stops, dropping annotations in the process, combining the coverage of the event-level construction with the compactness the annotations were designed to provide.

Algorithms compared in this work. MR (Delling et al. 2013), TAD (Katkalo, Rohovyi, and Walsh 2026), HL-RAPTOR, and HL-CSA produce ground-truth results, since they do not require shortcut precomputation and can be run directly on the delayed timetable. TD-Dijkstra is faster but can produce incorrect results on networks with buffer times, such as Switzerland in our experiments.

D-ULTRA-CSA, HL-CSA, TAD, and TD-Dijkstra are single-objective algorithms, returning the path with the earliest arrival time. MR, HL-RAPTOR, D-ULTRA-TB, D-ULTRA-RAPTOR, and D-ULTRA-RAPTOR (EP) return bicriteria Pareto-optimal results, optimising arrival time and number of trips.

D-ULTRA-CSA, D-ULTRA-RAPTOR, and D-ULTRA-RAPTOR (EP) use the stop-level delay shortcuts as their transfer graph. D-ULTRA-TB runs Trip-Based routing with the event-level D-ULTRA shortcuts of (Bez and Sauer 2024), using their delay annotations to decide which shortcuts to activate during the query. HL-RAPTOR and HL-CSA (Phan and Viennot 2019; Abraham et al. 2011; Delling, Goldberg, and Werneck 2016; Baum et al. 2019) replace Core-CH with hub labels computed on the transfer graph (preprocessing times in Table 1). Because hub labels encode shortest-path distances in the schedule-independent transfer graph, HL algorithms require no shortcut precomputation and are inherently delay-robust.

Preliminaries

This section defines key terminology and presents the foundational algorithms.

Terminology

We introduce the data model used throughout the paper.

Network. A public transit network consists of a set of stops $\mathcal{S}$, a set of trips $\mathcal{T}$, a set of routes, and a transfer graph $G = (V, E)$. Stops are physical locations: bus platforms, train stations, ferry terminals, where passengers board or alight vehicles.

A trip $T \in \mathcal{T}$ models a single vehicle run as a sequence of stop events $\langle \varepsilon_0, \ldots, \varepsilon_k \rangle$. Each stop event ε records the stop it serves, written $v(\varepsilon) \in \mathcal{S}$, together with an arrival time $\tau_{\text{arr}}(\varepsilon)$ and a departure time $\tau_{\text{dep}}(\varepsilon)$. A route groups trips that visit the same sequence of stops; trips within a route differ only in their scheduled times.

The transfer graph $G = (V, E)$ captures how passengers move between stops on foot or by other non-scheduled modes. Its vertex set satisfies $\mathcal{S} \subseteq V$ and may contain additional intermediate vertices representing street-network intersections or points of interest. Every edge $e = (u, w) \in E$ carries a travel time $\tau_{\text{tra}}(e)$. We impose no restrictions on G: it need not be transitively closed, it may contain cycles, and travel times may reflect walking, cycling, or any other schedule-independent mode.

Algorithms that process transfers in a single relaxation step per round, notably CSA and RAPTOR, require the transfer graph to be transitively closed among the stops it connects. A graph is transitively closed if for every pair of edges $(a, b), (b, c) \in E$ the edge (a, c) also belongs to E with $\tau_{\text{tra}}((a, c)) \leq \tau_{\text{tra}}((a, b)) + \tau_{\text{tra}}((b, c))$. Computing this closure is the primary bottleneck that limits transfer distances in practice (Wagner and Zündorf 2017), and overcoming it is the central motivation for shortcut-based approaches such as ULTRA.

Shortcuts. ULTRA precomputes shortcut edges that allow query algorithms to find optimal journeys without exploring the full transfer graph. Shortcuts can be indexed at two granularities.

An *event-level shortcut* is an edge $(\varepsilon_a, \varepsilon_b)$ connecting two stop events from different trips, together with a transfer travel time $\tau_{\text{tra}}(\varepsilon_a, \varepsilon_b)$. It encodes that a walking path of this duration from $v(\varepsilon_a)$ to $v(\varepsilon_b)$ may be part of an optimal journey. We denote the full set of event-level shortcuts by $\mathcal{E}_{\text{sc}}$. D-ULTRA (Bez and Sauer 2024) further annotates each event-level shortcut with a delay interval $[\delta_{\min}, \delta_{\max}]$ specifying under which delay realisations the shortcut remains necessary. Each event-level shortcut thus stores four attributes: destination event, travel time, minimum delay, and maximum delay. The full event-level graph is stored as a standalone adjacency array indexed by stop events, which also carries per-vertex data for all $|\mathcal{E}|$ stop-event vertices.

A *stop-level shortcut* is an edge (s, t) connecting two stops $s, t \in \mathcal{S}$, storing only the destination stop and the travel time. Since stop-level shortcuts carry no delay annotations, they are always active regardless of the delay scenario. The stop-level shortcut graph $G_{\text{sc}} = (\mathcal{S}, E_{\text{sc}})$ is obtained by projecting the event-level shortcuts onto the stops they serve; we define the projection precisely in Section .

Event-level shortcuts are a natural fit for TB-routing, which indexes transfers by stop event. Stop-level shortcuts are directly compatible with CSA and RAPTOR, which index transfers by stop and treat shortcuts as ordinary edges in the transfer graph.

Algorithms

Dijkstra and TD-Dijkstra. Dijkstra's algorithm (Dijkstra 1959) computes shortest paths from a single source by maintaining a priority queue of tentative distances and repeatedly settling the nearest unsettled vertex. In a time-dependent network, edge travel times depend on when the edge is traversed. Time-Dependent Dijkstra (TD-Dijkstra) (Stølting Brodal and Jacob 2004) adapts the classical algorithm by evaluating each edge weight at the current arrival time of its tail vertex, preserving optimality provided the network satisfies the FIFO property (earlier departure implies earlier arrival). Public transit schedules can violate FIFO when express and local services share stops; efficient implementations restore FIFO by filtering dominated connections during preprocessing.

Transfer Aware Dijkstra (TAD). Many transit networks specify a buffer time at each stop: a minimum waiting time that transferring passengers must respect before boarding a connecting service. The dominated-connection filtering used by TD-Dijkstra is unsound when buffer times are present, because it cannot distinguish seated-through passengers from transferring ones (Katkalo, Rohovyi, and Walsh 2026). TAD (Katkalo, Rohovyi, and Walsh 2026) addresses this by replacing per-edge relaxation with trip-level scanning, preserving correctness on networks with arbitrary buffer times. Two variants have been tested: with and without Bucket-CH. In our experiments TAD uses Bucket-CH as the faster option.

We additionally evaluate a classical TD-Dijkstra variant that applies dominated-connection filtering and uses Bucket-CH for acceleration. This variant is faster than TAD but is not applicable to networks with buffer times.

CSA. The Connection Scan Algorithm (Dibbelt et al. 2018) takes an array of all elementary connections sorted by departure time and performs a single linear sweep. For each connection whose departure stop has already been reached, the algorithm updates the arrival time at the connection's arrival stop.

RAPTOR. RAPTOR (Delling, Pajor, and Werneck 2012) organises the search into rounds, where round k finds the best journeys using at most k vehicle trips. Within each round the algorithm scans every route that serves a stop whose arrival time improved in the previous round, propagates arrival times along the route's stop sequence, and then relaxes all outgoing transfer edges from every improved stop. This round structure naturally produces a Pareto set over arrival time and number of trips.

Early Pruning (EP). Early Pruning (Rohovyi, Abuaisha, and Walsh 2026) is a technique that accelerates the transfer relaxation phase of RAPTOR and its variants without affecting optimality. The key observation is that if the outgoing transfer edges at each stop are pre-sorted by travel time, the algorithm can terminate the transfer loop as soon as the next transfer would produce an arrival time later than the current best arrival at the target.

Contraction Hierarchies (CH). CH (Geisberger et al. 2012) is a preprocessing technique for accelerating shortest-path queries in static graphs. Vertices are contracted in a heuristically determined order: removing a vertex while inserting shortcut edges between its neighbours to preserve shortest-path distances. The result is an augmented graph that decomposes into an upward graph (edges from lower-ranked to higher-ranked vertices) and a downward graph (the reverse). Queries are answered by bidirectional Dijkstra, with the forward search exploring the upward graph and the backward search exploring the downward graph.

Core-CH. Core-CH (Delling et al. 2013; Baum et al. 2019) adapts CH to multimodal networks by leaving a set of core vertices uncontracted, with all stops included in the core ($\mathcal{S} \subseteq V_c$). This produces a core graph over which Dijkstra searches are run during MR's transfer relaxation phase. To prevent the core graph from growing too large, contraction is stopped once the average vertex degree in the core exceeds a specified threshold.

Bucket-CH. Bucket-CH (Knopp et al. 2007; Geisberger et al. 2012) extends CH to handle one-to-many queries efficiently. After standard CH preprocessing, a backward search from each target vertex populates a bucket at every settled vertex with the distance to that target. A subsequent forward search from the source then scans these buckets to compute all source-to-target distances in a single pass. In our setting, TAD, TD-Dijkstra, and D-ULTRA all use Bucket-CH to accelerate initial and final transfers at query time. D-ULTRA's replacement search additionally uses Bucket-CH during the

update phase to find new event-level shortcuts. We do not apply Bucket-CH to MR, matching prior work (Bez and Sauer 2024), which also benchmarked MR without it.

MR. MR (Delling et al. 2013) is a modification of RAPTOR for the unlimited transfer problem. Instead of requiring a transitively closed transfer graph, MR runs Dijkstra searches accelerated by Core-CH during the transfer relaxation phase of each round.

Hub Labeling. An alternative to Core-CH for accelerating transfer relaxation is Hub Labeling (HL) (Abraham et al. 2011; Delling, Goldberg, and Werneck 2016). HL precomputes a two-hop labeling: each vertex stores a set of hubs with distances, so the walking distance between two stops is obtained by scanning their labels for shared hubs rather than searching the graph. Phan and Viennot (2019) applied HL to both RAPTOR and CSA for unrestricted walking. HL-RAPTOR was shown to be marginally faster than MR in (Baum et al. 2019), but HL-CSA has not been evaluated in the delay setting. We include both to provide delay-robust baselines that require no shortcut precomputation.

Trip-Based Routing. Trip-Based (TB) routing (Witt 2015) replaces route scanning with a direct search over trips. It maintains, for each trip, the earliest stop at which the trip can be boarded, and expands from one trip to the next via a precomputed transfer graph linking stop events. Because transfers are resolved at the event level, TB can exploit tighter pruning: a transfer from event ε_a to event ε_b is only used if the traveller arrives at $v(\varepsilon_a)$ before $\tau_{\text{dep}}(\varepsilon_b)$ minus the walking time. This event-level granularity is the reason D-ULTRA (Bez and Sauer 2024) targets TB: delay intervals can be attached to individual event-level shortcuts. However, the same granularity also makes the shortcut set vulnerable to coverage gaps when delays shift trip timings, as we demonstrate experimentally.

ULTRA. UnLimited TRAnsfers (ULTRA) (Baum et al. 2019) decouples the transfer precomputation from the query algorithm. During preprocessing, a witness–candidate Dijkstra search identifies, for each pair of consecutive vehicle trips in an optimal journey, the shortest transfer path through the full street network. These paths are stored as shortcut edges in the transfer graph, after which any query algorithm: CSA, RAPTOR, or TB can use the augmented graph as a drop-in replacement for the original transfer graph. The key insight is that the number of necessary shortcuts is far smaller than the full transitive closure, because most stop pairs are never connected by an optimal two-trip journey. ULTRA has been extended to multicriteria search (Potthoff and Sauer 2022). D-ULTRA (Bez and Sauer 2024; Sauer 2024) further extends the precomputation to account for bounded delays, producing event-level shortcuts annotated with delay intervals that indicate under which delay scenarios each shortcut remains feasible.

When a delay scenario is applied at query time, D-ULTRA's update phase adjusts the shortcut set in one of two modes (Bez and Sauer 2024). The *basic* update removes shortcuts that have become infeasible under the current delays: either the transfer is no longer physically possible (the passenger arrives too late to board), or the realised delay falls outside the shortcut's annotated interval. The *advanced* update additionally performs a heuristic replacement search: for each origin event whose shortcuts were invalidated, a Bucket-CH search finds new event-level shortcuts that are feasible under the current delay scenario, partially recovering the transfer coverage lost by the removal step. The number of replacement shortcuts is small relative to the full set (typically below 0.2%), but as we show experimentally, even this modest recovery noticeably reduces D-ULTRA-TB's error rate.

Stop-Level Delay Shortcuts

D-ULTRA (Bez and Sauer 2024; Sauer 2024) computes shortcuts at the event level: each shortcut $(\varepsilon_a, \varepsilon_b) \in \mathcal{E}_{\text{sc}}$ connects two specific stop events and carries a delay interval indicating under which delay realisations the shortcut is necessary. This granularity is a natural fit for TB-routing, which indexes transfers by stop event, but it is incompatible with CSA and RAPTOR, which index transfers by stop. We propose a stop-level alternative that projects event-level shortcuts onto the stops they serve. Each stop event ε may receive an independent arrival delay $\delta_{\text{arr}}(\varepsilon)$ and departure delay $\delta_{\text{dep}}(\varepsilon)$ in $[0, \Delta]$, yielding delayed times

$$\tilde{\tau}_{\text{arr}}(\varepsilon) = \tau_{\text{arr}}(\varepsilon) + \delta_{\text{arr}}(\varepsilon), \tag{1}$$

$$\tilde{\tau}_{\text{dep}}(\varepsilon) = \tau_{\text{dep}}(\varepsilon) + \delta_{\text{dep}}(\varepsilon). \tag{2}$$

A *delay scenario* assigns such a delay pair to every event.

Projection from Event Level to Stop Level

Given the set of event-level shortcuts $\mathcal{E}_{\text{sc}}$ produced by D-ULTRA, we construct the stop-level shortcut graph $G_{\text{sc}} = (\mathcal{S}, E_{\text{sc}})$ as follows. For each pair of stops (s, t) connected by at least one event-level shortcut, we insert a single stop-level edge:

$$E_{\text{sc}} = \big\{(s,t) \in \mathcal{S} \times \mathcal{S} : \exists\, (\varepsilon_a, \varepsilon_b) \in \mathcal{E}_{\text{sc}} \text{ with } v(\varepsilon_a) = s \text{ and } v(\varepsilon_b) = t\big\}. \tag{3}$$

Each such edge carries the minimum travel time over all contributing event-level shortcuts:

$$\tau_{\text{tra}}(s,t) = \min_{\substack{(\varepsilon_a,\varepsilon_b)\in\mathcal{E}_{\text{sc}} \\ v(\varepsilon_a)=s,\ v(\varepsilon_b)=t}} \tau_{\text{tra}}(\varepsilon_a, \varepsilon_b). \tag{4}$$

The stop-level graph G_{sc} is merged into the existing transfer graph G as a set of new edges between stop vertices, requiring no additional vertex allocation. Event-level and stop-level shortcuts are stored in two separate data structures; for D-ULTRA-CSA and D-ULTRA-RAPTOR, the stop-level graph fully replaces the event-level set rather than augmenting it as a separate layer.

Since many event-level shortcuts between distinct events at the same pair of stops collapse into a single stop-level edge, $|E_{\text{sc}}| \ll |\mathcal{E}_{\text{sc}}|$ in practice (see Table 1). Figure 1 illustrates this construction.

The projection can equivalently be expressed as a map $\pi\colon \mathcal{E}_{\text{sc}} \to E_{\text{sc}}$ defined by

$$\pi(\varepsilon_a, \varepsilon_b) = \big(v(\varepsilon_a),\ v(\varepsilon_b)\big), \tag{5}$$

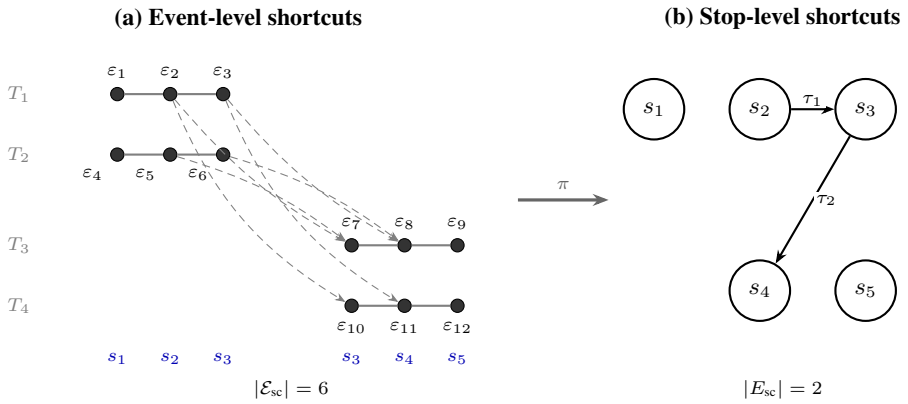

Figure 1. Projection π from event-level to stop-level shortcuts. (a) Trips T_1, T_2 serve s_1, s_2, s_3 and T_3, T_4 serve s_3, s_4, s_5. Six event-level shortcuts (dashed) link stop events across the two trip groups. (b) Shortcuts sharing a stop pair collapse into a single edge whose travel time is the minimum over its preimage (Eq. (4)), here $\tau_1 = \min\{\tau_{\text{tra}}(\varepsilon_2, \varepsilon_7),\ \tau_{\text{tra}}(\varepsilon_2, \varepsilon_{10}),\ \tau_{\text{tra}}(\varepsilon_5, \varepsilon_7)\}$ and $\tau_2 = \min\{\tau_{\text{tra}}(\varepsilon_3, \varepsilon_8),\ \tau_{\text{tra}}(\varepsilon_3, \varepsilon_{11}),\ \tau_{\text{tra}}(\varepsilon_6, \varepsilon_8)\}$.

which sends every event-level shortcut to the stop pair it connects, so $E_{\text{sc}} = \pi(\mathcal{E}_{\text{sc}})$ and each stop-level edge's travel time is the minimum over its preimage (Equation (4)). Since many trips visit the same stops, the preimage of a single stop pair (s, t) can contain hundreds of event-level shortcuts, all of which collapse into one edge. This explains the count compression ratios observed in Table 1 (48–180×); memory compression is even higher (141–375×) because the projection also discards the per-edge delay annotations.

Accuracy

The stop-level shortcut set E_{sc} is a conservative superset of the event-level set at the stop-pair level. To see why, consider a transfer from stop s to stop t and suppose the event-level set contains at least one shortcut $(\varepsilon_a, \varepsilon_b) \in \mathcal{E}_{\text{sc}}$ with $v(\varepsilon_a) = s$ and $v(\varepsilon_b) = t$. By construction, $\pi(\varepsilon_a, \varepsilon_b) = (s, t) \in E_{\text{sc}}$, so the stop-level graph contains this edge. Moreover, $\tau_{\text{tra}}(s, t) \leq \tau_{\text{tra}}(\varepsilon_a, \varepsilon_b)$ by the minimum in Equation (4), so the stop-level travel time is at most as large. We can therefore state the following property.

Superset Property. For every stop pair (s, t) covered by at least one event-level shortcut in $\mathcal{E}_{\text{sc}}$, the edge (s, t) is present in G_{sc} with a travel time no greater than that of any contributing event-level shortcut. Since D-ULTRA's precomputation is designed to produce event-level shortcuts for all transfers needed under any delay scenario in $[0, \Delta]$, the stop-level graph inherits this coverage at the stop-pair level.

The converse does not hold: E_{sc} may contain edges that are not needed under a particular delay scenario. An unnecessary shortcut may cause the query algorithm to attempt boarding a connection that is in fact infeasible (because the passenger arrives too late), but the algorithm will simply fail to board and fall back to alternative paths. This conservatism therefore does not harm accuracy; it can only introduce a marginal amount of extra work during the query.

In contrast, D-ULTRA (Bez and Sauer 2024) keeps shortcuts event-specific: each shortcut applies only to a particular pair of stop events. When delays shift trip timings, the event-level set may lack a shortcut for the event pair that has become necessary, even though the same stop pair is covered by a different shortcut whose endpoint events no longer line up. This coverage gap can cause D-ULTRA-TB to miss Pareto-optimal journeys, as we demonstrate experimentally.

Experiments

On both networks tested, the stop-level projection compresses D-ULTRA shortcuts in memory by 141–375×, yields D-ULTRA-CSA queries that are 1.9–4.2× faster than D-ULTRA-TB in the single-objective setting, and, with replacement shortcuts, returns zero missed Pareto-optimal journeys, whereas D-ULTRA-TB misses dozens to hundreds across configurations.

We ask two questions: (1) how fast are stop-level queries compared to existing algorithms? and (2) do they find all optimal journeys? We evaluate on two real-world transit networks, London and Switzerland, following the experimental setup of Bez and Sauer (2024).

All experiments were compiled with g++ 13.3.0 and executed on an AMD Ryzen 9 7900X (12-core, 24-thread) workstation with 48 GB of RAM running Ubuntu 24.04 under WSL2. Query times (Table 2) are wall-clock averages over 10 000 random source–target pairs with departure times sampled uniformly from the 13:00–14:00 window, matching the methodology of Bez and Sauer (2024). Accuracy (Table 3) is evaluated on 1 000 *affected* queries generated following Bez and Sauer (2024): random source–target pairs with departure times in the 12:00–13:00 delay window are sampled until 1 000 queries are found where the delay changes the Pareto-optimal journey set. Delays are generated following the synthetic model of Bez and Sauer (2024): one delay incident per trip is created by choosing a random start index; all subsequent stop events in that trip receive the same arrival and departure delay, drawn from the distribution of Bez and Sauer (2024) (based on aggregated real-world punctuality data, ranging from 0 to 60 minutes with 50% of incidents receiving zero delay). The delay sce-

nario covers the 12:00–13:00 window using seed 42. Following Bez and Sauer (2024), accuracy is evaluated in *hypothetical mode*: all shortcut updates are applied to completion before any query is issued. In a real-time system, updates take non-negligible time and queries may arrive while an update is still in progress; hypothetical mode removes this latency effect, so that any errors measured are attributable to the shortcut set itself rather than to slow update processing. We evaluate two update modes: *without replacement*, where shortcuts invalidated by the delay scenario are removed; and *with replacement*, where the update procedure of Bez and Sauer (2024) additionally computes replacement shortcuts via Bucket-CH search. The replacement shortcuts constitute a negligible fraction of the total shortcut set and have negligible impact on query times. Table 2 reports query times and error rates for the with-replacement variant; Table 3 reports both modes, as the distinction affects accuracy on the harder affected-query set.

Instances

Table 1 summarises the two networks. Both datasets are publicly available from the Karlsruhe Institute of Technology[1] and have been used in prior work on unlimited transfers (Potthoff and Sauer 2022; Baum et al. 2019; Bez and Sauer 2024; Sauer 2024), making them the natural choice for a direct comparison. London is a dense urban network with relatively few routes but many trips per route, while Switzerland is a nationwide network with an order of magnitude more routes and a sparser stop distribution spread across a larger geographic area. We evaluate London at three delay buffers ($\Delta = 0$, 120 s, and 180 s), matching the configurations of Bez and Sauer (2024), and Switzerland at two ($\Delta = 0$ and 300 s) to study how the buffer size affects both shortcut counts and query behaviour. The $\Delta = 0$ configurations serve as baselines with a zero delay buffer.

The compression ratio grows with the delay buffer because larger buffers generate many additional event-level shortcuts between the same stop pairs, all of which collapse into existing stop-level edges. Increasing Δ from 120 s to 180 s on London doubles the event-level set (86.6 M to 176.6 M) but grows the stop-level set by only 58% (622 K to 979 K), demonstrating that the projection absorbs much of the additional redundancy. This reduction is the primary reason stop-level query algorithms operate on a much smaller transfer graph, translating directly into faster query times.

In memory, the gap widens further (Table 1): the event-level graph stores $\sim$4.5 M vertices at 16 B each plus 16 B per edge (destination, travel time, $\delta_{\min}$, $\delta_{\max}$), totalling 264 MB–2.9 GB on disk, whereas stop-level shortcuts add only 8 B per edge (destination, travel time) to the existing transfer graph, occupying just 1.6–6.3 MB; the resulting 141–375$\times$ ratio exceeds the count ratio because the projection eliminates both the per-edge delay annotations (halving the edge cost) and the $\sim$72 MB stop-event vertex overhead.

The projection itself is fast: 0.6–6.4 s across configurations, negligible compared to D-ULTRA's event-level preprocessing, which on our hardware takes 24–35 min on London and 18–24 min on Switzerland. All code is implemented in C++ and is publicly available.[2]

Query Performance

Table 2 reports the average query time on both networks.

D-ULTRA-CSA is the fastest algorithm on all configurations by a wide margin: 7.5–9.3$\times$ faster than MR on London and 10.5–12.2$\times$ on Switzerland. Compared to D-ULTRA-TB, the query algorithm evaluated in Bez and Sauer (2024), D-ULTRA-CSA achieves a 1.9–4.2$\times$ speedup across all configurations. The gap widens with increasing delay buffer: at $\Delta = 0$ D-ULTRA-CSA is 1.9–2.2$\times$ faster than D-ULTRA-TB, but at $\Delta = 180$ s on London it reaches 4.2$\times$, because D-ULTRA-TB must process the full event-level shortcut set, which grows steeply with the buffer, while D-ULTRA-CSA operates on the much smaller stop-level set.

Among the bicriteria algorithms, D-ULTRA-TB is fastest at $\Delta = 0$ on both networks and at $\Delta = 300$ s on Switzerland, where its trip-based indexing benefits from the large number of routes (13 786). However, on London, D-ULTRA-RAPTOR (EP) overtakes D-ULTRA-TB as soon as the delay buffer exceeds zero: at $\Delta = 120$ s D-ULTRA-RAPTOR (EP) is 10% faster (5.77 ms vs. 6.38 ms), and at $\Delta = 180$ s the gap widens to 21% (6.51 ms vs. 8.26 ms), because the event-level shortcut set that D-ULTRA-TB must process doubles while the stop-level set used by D-ULTRA-RAPTOR grows by only 58%. On Switzerland, D-ULTRA-TB retains a 42–55% advantage over D-ULTRA-RAPTOR (EP) thanks to the route-rich topology.

D-ULTRA-TB's speed advantage at $\Delta = 0$ comes at the cost of accuracy: it misses Pareto-optimal journeys on both networks at every delay buffer, including $\Delta = 0$ (Table 3).

Among the single-objective baselines with no preprocessing costs, TAD provides a 2.3–3.1$\times$ speedup over MR and is fully correct on both networks. TD-Dijkstra is 19–25% faster than TAD on London but is limited to buffer-free networks as noted above.

HL-RAPTOR, a delay-robust baseline that bypasses shortcuts entirely, matches or slightly underperforms MR (0.9–1.2$\times$) on both networks (Table 2). Despite its one-time preprocessing cost of 3 min (London) or 15 min (Switzerland), HL-RAPTOR yields no meaningful query-time benefit over MR. HL-CSA is slower (0.7$\times$ of MR). To our knowledge, this is the first comparison of HL-CSA against MR on the same machine and query set. Both HL algorithms are fully correct by construction, since, like MR and TAD, they operate on the delayed timetable directly.

Accuracy

With replacement shortcuts, D-ULTRA-CSA, D-ULTRA-RAPTOR, and D-ULTRA-RAPTOR (EP) all produce no errors on any configuration in our experiments (Table 3). Without replacement, stop-level algorithms remain error-free on London at every delay buffer; on Switzerland at $\Delta = 0$, D-ULTRA-RAPTOR misses 2 out of 1 000 affected queries and D-ULTRA-CSA returns one suboptimal arrival, caused

[1] https://i11www.iti.kit.edu/PublicTransitData/ULTRA/

[2] https://github.com/andrii-rohovyi/PublicTransitRoutingWithUnlimitedTransfer

	London			Switzerland	
	Δ=0	Δ=120 s	Δ=180 s	Δ=0	Δ=300 s
Stops		19 682		25 125	
Routes		1 955		13 786	
Trips		114 508		350 006	
Stop events		4 508 644		4 686 865	
Connections		4 394 136		4 336 859	
Transfer graph vertices		181 642		603 691	
Transfer graph edges (stops level)		334 112		465 067	
Transfer graph edges (event level)		575 364		1 853 260	
Event-level shortcuts	11 988 145	86 573 818	176 630 535	9 414 856	119 664 033
Stop-level shortcuts	213 705	621 597	979 238	194 948	668 627
Count ratio	56×	139×	180×	48×	179×
Event-level disk (MB)	264	1 457	2 898	226	1 990
Stop-level disk (MB)	1.7	5.0	7.8	1.6	5.3
Memory ratio	155×	291×	372×	141×	375×
D-ULTRA preprocessing time (min)	24	31	35	18	24
Projection computation time (s)	1.2	3.9	6.4	0.6	3.2
Bucket-CH preprocessing time		7.25 s		33.37 s	
Core-CH preprocessing time		1.81 s		3.66 s	
HL preprocessing time		3 min		15 min	

Table 1. Network characteristics. D-ULTRA preprocessing builds event-level shortcuts; projection converts them to stop-level. Preprocessing times for transfer-graph acceleration structures (Bucket-CH, Core-CH, HL) are also reported.

	London						Switzerland			
	Δ=0		Δ=120 s		Δ=180 s		Δ=0		Δ=300 s	
Algorithm	ms/q	×	ms/q	×	ms/q	×	ms/q	×	ms/q	×
MR	15.34	1.0	14.36	1.0	14.81	1.0	28.69	1.0	28.43	1.0
HL-RAPTOR	12.59	1.2	13.62	1.1	14.20	1.0	31.60	0.9	32.83	0.9
HL-CSA	20.99	0.7	22.00	0.7	21.32	0.7	43.90	0.7	40.06	0.7
TAD	5.78	2.7	6.19	2.3	6.07	2.4	9.32	3.1	9.06	3.1
TD-Dijkstra	4.67	3.3	4.61	3.1	4.77	3.1	—		—	
D-ULTRA-TB	3.60	4.3	6.38	2.3	8.26	1.8	4.38	6.5	6.41	4.4
D-ULTRA-RAPTOR	5.68	2.7	5.96	2.4	7.16	2.1	10.19	2.8	11.40	2.5
D-ULTRA-RAPTOR (EP)	4.72	3.2	5.77	2.5	6.51	2.3	9.85	2.9	11.01	2.6
D-ULTRA-CSA	1.64	9.3	1.82	7.9	1.97	7.5	2.36	12.2	2.71	10.5
Error rate [%] on the same queries (with replacement)										
	F.Q	F.J	F.Q	F.J	F.Q	F.J	F.Q	F.J	F.Q	F.J
D-ULTRA-TB	1.07	0.29	0.76	0.21	0.60	0.17	0.28	0.08	0.07	0.02
D-ULTRA-RAPTOR	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
D-ULTRA-RAPTOR (EP)	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
D-ULTRA-CSA	0.00		0.00		0.00		0.00		0.00	

Table 2. Average query time (ms), speedup relative to MR, and error rate on 10 000 random queries with replacement shortcuts. The bottom section reports F.Q (% of queries with at least one missed Pareto-optimal journey) and F.J (% of missed journeys).

by the removal of shortcuts at $\Delta = 0$, where no delay buffer cushions the loss. These isolated errors disappear with replacement shortcuts and at all other configurations. TD-Dijkstra matches MR on all London configurations, confirming that dominated-connection filtering is safe on buffer-free networks; it is omitted from Switzerland where buffer times make the filtering unsound.

In contrast, D-ULTRA-TB misses Pareto-optimal journeys on both networks at every delay buffer and under both update modes (Table 3). These errors arise from *coverage gaps* in the event-level shortcut set: the precomputed shortcut for a specific event pair may not exist under the delayed timetable, even though the same stop pair is covered by a different event-level shortcut that projects to the same stop-level edge. This mechanism is present at all values of Δ, including $\Delta = 0$, where the delay buffer is zero and the origin delay interval of every shortcut is $[0, 0]$.

D-ULTRA-TB error rates *decrease* with larger Δ: without replacement, from 411 to 168 failed queries on London and from 152 to 28 on Switzerland; with replacement, from 249 to 143 on London and from 51 to 2 on Switzerland. Larger delay buffers generate more event-level shortcuts, improv-

	London						Switzerland			
	Δ=0		Δ=120 s		Δ=180 s		Δ=0		Δ=300 s	
Algorithm	F.Q	F.J	F.Q	F.J	F.Q	F.J	F.Q	F.J	F.Q	F.J
Without replacement shortcuts										
D-ULTRA-TB	411	577/4 598	222	301/4 598	168	217/4 598	152	178/4 771	28	34/4 771
% of affected	41.1	12.5	22.2	6.5	16.8	4.7	15.2	3.7	2.8	0.7
% of all	7.38	2.26	3.98	1.18	3.02	0.85	0.27	0.07	0.05	0.01
D-ULTRA-RAPTOR / (EP)	0	0/4 598	0	0/4 598	0	0/4 598	2	3/4 771	0	0/4 771
% of affected	0.0	0.0	0.0	0.0	0.0	0.0	0.2	0.1	0.0	0.0
% of all	0.00	0.00	0.00	0.00	0.00	0.00	<0.01	<0.01	0.00	0.00
D-ULTRA-CSA	0		0		0		1		0	
% of affected	0.0		0.0		0.0		0.1		0.0	
% of all	0.00		0.00		0.00		<0.01		0.00	
With replacement shortcuts										
D-ULTRA-TB	249	327/4 598	171	222/4 598	143	182/4 598	51	56/4 771	2	2/4 771
% of affected	24.9	7.1	17.1	4.8	14.3	4.0	5.1	1.2	0.2	<0.1
% of all	4.47	1.28	3.07	0.87	2.57	0.71	0.09	0.02	<0.01	<0.01
D-ULTRA-RAPTOR / (EP)	0	0/4 598	0	0/4 598	0	0/4 598	0	0/4 771	0	0/4 771
% of affected	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
% of all	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
D-ULTRA-CSA	0		0		0		0		0	
% of affected	0.0		0.0		0.0		0.0		0.0	
% of all	0.00		0.00		0.00		0.00		0.00	

Table 3. Accuracy on 1 000 affected queries (queries where the delay changes the Pareto-optimal journey set), evaluated in hypothetical mode (see text). "F.Q" counts queries with at least one missed Pareto-optimal journey; "F.J" counts individual missed journeys. Error rates are shown as percentages under two normalizations: "% of affected" divides by the 1 000 affected queries (and their 4 598 / 4 771 journeys for London / Switzerland); "% of all" divides by all randomly generated queries.

ing transfer coverage and reducing coverage-gap errors. Replacement shortcuts further reduce D-ULTRA-TB's errors by recovering some removed event-level shortcuts (e.g., 411 vs. 249 at $\Delta = 0$ on London), but substantial errors persist because replacement cannot address the fundamental coverage-gap mechanism. With replacement shortcuts, stop-level algorithms avoid this failure mode: the projection provides broad stop-pair coverage regardless of which individual event-level shortcuts exist. Without replacement, the 3 failures on Switzerland at $\Delta = 0$ (2 queries for D-ULTRA-RAPTOR / (EP), 1 for D-ULTRA-CSA) confirm that immunity depends on the replacement step recovering any shortcuts lost during the update.

Table 3 reports errors under two normalizations: "% of affected" divides by the 1 000 affected queries (the challenging cases that test delay handling); "% of all" divides by all randomly generated queries (5 572 for London, 56 754 for Switzerland), matching the implicit denominator in Bez and Sauer (2024), as confirmed with the authors.

Conclusion and Future Steps

Three main conclusions follow. (i) D-ULTRA-CSA yields the fastest single-objective query times (1.9–4.2× faster than D-ULTRA-TB and up to 12.2× faster than MR) and reduces the shortcut memory footprint by 141–375× (from gigabytes to megabytes); the advantage grows with the delay buffer because the event-level set D-ULTRA-TB must process scales steeply while the stop-level set stays compact. (ii) With replacement shortcuts, the stop-level approach eliminates the journey errors D-ULTRA-TB exhibits at every delay buffer tested: these errors are coverage gaps in the event-level shortcut set, which the projection plugs by merging shortcuts across events at the same stop pair. Without replacement, stop-level algorithms exhibit only a handful of failures across all configurations (Switzerland $\Delta = 0$), compared with hundreds for D-ULTRA-TB. (iii) For bicriteria queries, D-ULTRA-RAPTOR with stop-level shortcuts, enhanced by Early Pruning (Rohovyi, Abuaisha, and Walsh 2026), outperforms D-ULTRA-TB on London at non-zero delay buffers by up to 21%.

A natural extension is to adapt the framework to additional criteria (transfer time, costs, etc), already worked out for classical ULTRA (Potthoff and Sauer 2022) but not for the delay-tolerant setting.

More broadly, the projection idea generalises whenever a precomputed structure stores finer detail than the query algorithm uses: applying projection yields a conservative superset that is smaller, faster to traverse, and free of annotations that may not match the realised scenario. Whether this pays off in other temporal domains is an open question.

Acknowledgments

We thank Jonas Sauer for detailed feedback on an earlier draft, including the pointer to Dominik Bez's BA thesis and discussion that refined our experimental analysis. Also, we thank Abdallah Abuaisha and Peter J. Stuckey for discussions during the early stages of this work.

Andrii Rohovyi and Toby Walsh were supported by an ARC Laureate award on Trustworthy AI, FL200100204.

References

Abraham, I.; Delling, D.; Goldberg, A. V.; and Werneck, R. F. F. 2011. A Hub-Based Labeling Algorithm for Shortest Paths in Road Networks. In *Experimental Algorithms, 10th International Symposium (SEA)*, volume 6630 of *Lecture Notes in Computer Science*, 230–241.

Abuaisha, A.; Shen, B.; Harabor, D.; Stuckey, P. J.; and Wallace, M. 2025. Dynamic Replanning for Improved Public Transport Routing. In *Proceedings of the 34th International Joint Conference on Artificial Intelligence (IJCAI)*.

Bast, H.; Carlsson, E.; Eigenwillig, A.; Geisberger, R.; Harrelson, C.; Raychev, V.; and Viger, F. 2010. Fast Routing in Very Large Public Transportation Networks Using Transfer Patterns. In *ESA 2010*, volume 6346, 290–301.

Bauer, R.; Delling, D.; Sanders, P.; Schieferdecker, D.; Schultes, D.; and Wagner, D. 2010. Combining Hierarchical and Goal-Directed Speed-up Techniques for Dijkstra's Algorithm. *Journal of Experimental Algorithmics (JEA)*, 15: 2.3:1–2.3:31.

Baum, M.; Buchhold, V.; Sauer, J.; Wagner, D.; and Zündorf, T. 2019. UnLimited TRAnsfers for Multi-Modal Route Planning: An Efficient Solution. In *Proceedings of the 27th Annual European Symposium on Algorithms (ESA'19)*, volume 144 of *Leibniz International Proceedings in Informatics (LIPIcs)*, 14:1–14:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.

Bez, D. 2020. *UnLimited TRAnsfer Shortcuts with Delay Tolerance for Multi-Modal Journey Planning*. Bachelor's thesis, Karlsruhe Institute of Technology.

Bez, D.; and Sauer, J. 2024. Fast and Delay-Robust Multimodal Journey Planning. In *Proceedings of the 26th Workshop on Algorithm Engineering and Experiments (ALENEX'24)*, 105–117. Society for Industrial and Applied Mathematics (SIAM).

Delling, D.; Dibbelt, J.; Pajor, T.; Wagner, D.; and Werneck, R. F. 2013. 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 (LNCS)*, 260–271. Springer.

Delling, D.; Goldberg, A. V.; and Werneck, R. F. 2016. Hub Labeling (2-Hop Labeling). In *Encyclopedia of Algorithms*, 932–938. Springer.

Delling, D.; Pajor, T.; and Werneck, R. F. 2012. Round-Based Public Transit Routing. In *Proceedings of the 14th Meeting on Algorithm Engineering and Experiments (ALENEX)*, 130–140. SIAM.

Dibbelt, J.; Pajor, T.; Strasser, B.; and Wagner, D. 2018. Connection Scan Algorithm. *ACM Journal of Experimental Algorithmics*, 23: 1.7:1–1.7:56.

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

Geisberger, R.; Sanders, P.; Schultes, D.; and Vetter, C. 2012. Exact Routing in Large Road Networks Using Contraction Hierarchies. *Transportation Science*, 46(3): 388–404.

Katkalo, D.; Rohovyi, A.; and Walsh, T. 2026. Adapting Dijkstra for Buffers and Unlimited Transfers. *arXiv preprint arXiv:2603.11729*.

Knopp, S.; Sanders, P.; Schultes, D.; Schulz, F.; and Wagner, D. 2007. Computing Many-to-Many Shortest Paths Using Highway Hierarchies. In *Proceedings of the 9th Workshop on Algorithm Engineering and Experiments (ALENEX'07)*, 36–45. Society for Industrial and Applied Mathematics (SIAM).

Phan, D.-M.; and Viennot, L. 2019. Fast Public Transit Routing with Unrestricted Walking Through Hub Labeling. In *Analysis of Experimental Algorithms – Special Event, SEA^2 2019*, volume 11544 of *Lecture Notes in Computer Science*, 237–247. Springer.

Potthoff, M.; and Sauer, J. 2022. Fast Multimodal Journey Planning for Three Criteria. In *Proceedings of the 24th Workshop on Algorithm Engineering and Experiments (ALENEX'22)*, 145–157. Society for Industrial and Applied Mathematics (SIAM).

Rohovyi, A.; Abuaisha, A.; and Walsh, T. 2026. Early Pruning for Public Transport Routing. *arXiv preprint arXiv:2603.12592*. Accepted at WCTR 2026. To appear in Transportation Research Procedia.

Rohovyi, A.; Stuckey, P. J.; and Walsh, T. 2025. Multimodal Pathfinding with Personalized Travel Speed and Transfers of Unlimited Distance. In *Proceedings of the 37th IEEE International Conference on Tools with Artificial Intelligence (ICTAI'25)*, 925–931. IEEE.

Sauer, J. 2024. *Closing the Performance Gap Between Multimodal and Public Transit Journey Planning*. Ph.D. thesis, Karlsruhe Institute of Technology (KIT), Karlsruhe, Germany.

Stølting Brodal, G.; and Jacob, R. 2004. Time-dependent Networks as Models to Achieve Fast Exact Time-table Queries. *Electronic Notes in Theoretical Computer Science*, 92: 3–15. Proceedings of ATMOS Workshop 2003.

Wagner, D.; and Zündorf, T. 2017. Public Transit Routing with Unrestricted Walking. In D'Angelo, G.; and Dollevoet, T., eds., *17th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2017)*, volume 59 of *Open Access Series in Informatics (OASIcs)*, 7:1–7:14. Dagstuhl, Germany: Schloss Dagstuhl – Leibniz-Zentrum für Informatik.

Witt, S. 2015. Trip-Based Public Transit Routing. In Bansal, N.; and Finocchi, I., eds., *Algorithms — ESA 2015*, volume 9294 of *Lecture Notes in Computer Science*, 1025–1036. Springer. ISBN 978-3-662-48350-3.

Witt, S. 2016. 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 *Open Access Series in Informatics (OASIcs)*, 10:1–10:12. Dagstuhl, Germany: Schloss Dagstuhl – Leibniz-Zentrum für Informatik.