REX: A Realistic Time-Dependent Model for Multimodal Public Transport

Spyros Kontogiannis
Computer Science & Engineering Department, University of Ioannina, Greece
Computer Technology Institute & Press “Diophantus”, Rion, Greece

Paraskevi-Maria-Malevi Machaira
Department of Computer Engineering & Informatics, University of Patras, Greece
Computer Technology Institute & Press “Diophantus”, Rion, Greece

Andreas Paraskevopoulos
Department of Computer Engineering & Informatics, University of Patras, Greece
Computer Technology Institute & Press “Diophantus”, Rion, Greece

Christos Zaroliagis
Department of Computer Engineering & Informatics, University of Patras, Greece
Computer Technology Institute & Press “Diophantus”, Rion, Greece

Abstract
We present the non-FIFO time-dependent graph model with REalistic vehicle eXchange times (REX) for schedule-based multimodal public transport, along with a novel query algorithm called TRIP-based LAbel-correction propagation (TRIPLA) algorithm that efficiently solves the realistic earliest-arrival routing problem. The REX model possesses all strong features of previous time-dependent graph models without suffering from their deficiencies. It handles non-negligible exchanges from one vehicle to another, as well as supports non-FIFO instances which are typical in public transport, without compromising space efficiency. We conduct a thorough experimental evaluation with real-world data which demonstrates that TRIPLA significantly outperforms all state-of-the-art query algorithms for multimodal earliest-arrival routing in schedule-based public transport.

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

Keywords and phrases multimodal journey planning, REX model, TRIPLA query algorithm, schedule-based timetables


Funding This work was supported by the Operational Program Competitiveness, Entrepreneurship and Innovation (co-financed by EU and GR national funds), under project i-Deliver (T2EDK-03472).

1 Introduction
Nowadays, a plethora of applications allows commuters to plan their journeys using public transport. The journey planning (JP) problem refers to the computation of optimal journeys as a real-time response to routing queries. The most realistic version of this problem, known as multimodal journey planning (MJP) problem, supports a combination of different transport modes (bus, metro, train, tram, walking, etc.). MJP provides optimal journeys from an origin $A$ to a destination $B$, in schedule-based multimodal public-transport systems, which meet one or more optimization criteria. The most commonly used criteria are the earliest arrival (EA) and the minimum number of vehicle exchanges, a.k.a. the minimum number of transfers (MNT). The corresponding variants of the problem are MJP$_{EA}$ (for earliest-arrivals) and MJP$_{MNT}$ (for minimum-number-of-transfers).
A schedule-based public transport system consists of a timetable that contains the departure and arrival times of the scheduled vehicles. The challenge in designing schedule-based public transport systems is the modelling of the timetable information so that optimal journey-planning queries can be efficiently answered. In the scenario under consideration, a centralised server accessible to every customer has to respond, in real-time, to a stream of optimal-journey queries. The goal is to model timetable information in order to reduce the average response time for a query. The most common approaches use a preprocessing stage that constructs the data structure used to represent the timetable information. As for representing schedule-based public-transport instances, there are two main axes which have been considered.

The first axis concerns the type of travel-time. The simplified travel-time approach assumes that the vehicle exchanges within stations take negligible time and the FIFO property holds for all the connections between stations. The realistic travel-time approach allows the vehicle exchanges within stations to require non-negligible time, and also allows the existence of non-FIFO connections between stations (e.g., due to passing-by trains of different speeds).

The second axis concerns the graph model used to represent the timetables. The time-dependent graph model \[2, 11, 13, 14, 18\] is more compact, in the sense that the stations correspond to graph nodes. The time-expanded graph model \[10, 15, 21, 22\] allows for a more detailed representation of the timetable, by allocating, not just stations, but timestamped stations to the vertices of the graph. Due to the temporal characteristics of the vertices, the resulting graph is acyclic, allowing for quite simple query algorithms. The size of the representation blows up in this case, since there are several timestamped copies of the same station, each representing a different departure/arrival event in the timetable of the station.

Our focus in this paper is the study of time-dependent graph models for schedule-based public-transport instances. Two characteristic representatives of this family are the BJ \[2\], and the PSWZ \[18\] models. We present the non-FIFO time-dependent graph model with REalistic eXchange times (REX), which aims to combine the strong features of the BJ and the PSWZ models, without suffering from their deficiencies. In particular, REX allows for non-negligible transfer times, as in the PSWZ model, but without increasing the size of the time-dependent graph: each station is represented by a single vertex, and there is an arc between two nodes when at least one elementary connection (irrespective of the vehicle types using it) exists between them, as in the BJ model. Of course, there is a price to pay for this enhancement: the Dijkstra-like label-setting query algorithm no longer works as such. To tackle this problem, we also propose a novel query algorithm, called the TRIP-based Label correcting (TRIPLA) algorithm that solves MJP$_{E,A}$ even when the FIFO property is violated by some arcs. This is a label-correcting shortest-path algorithm, which nevertheless conducts a targeted label correction, via an appropriate data structure that we maintain at the vertices and a novel label-correction propagation (LCPROP) phase that TRIPLA uses to update the vertex labels when a delay occurs.

2 Preliminaries

Schedule-based public-transport networks are described by timetable information. Timetables consist of scheduled trips described by their sequence of stops and the corresponding departure and arrival times. More formally, a timetable $T$ is a tuple $(Z, B, C)$, where $Z$ is the set of public-transport vehicles, $B$ is the set of stops (or stations), and $C$ is the set of elementary connections. Each elementary connection is a tuple $c = (Z, S_d, S_a, t_d, t_a)$. For each attribute $x$ of an elementary connection $c \in C$, its value is denoted by $x(c)$. Therefore, $c \in C$ represents the journey of a particular vehicle $Z(c) \in Z$ which departs from the origin-stop $S_d(c) \in B$ at
(departure) time \( t_d(c) \), and arrives at the destination-stop \( S_a(c) \in B \) at (arrival) time \( t_a(c) \), with no intermediate stop. The journeys are considered to be periodic, with period \( T_p \), which may vary from one day to one week. It is assumed that every connection’s travel-time and every stop’s transfer-time is less than \( T_p \), while the a time unit of 1 minute is considered.

In the time-dependent graph model, timetables are represented by a weighted graph \( G = (V, E) \), whose vertex set \( V \) represents (possibly timestamped copies of) stations and \( E \) represents (either single, or bundles of) elementary connections between stations. \( E_u \subseteq E \) is the subset of outgoing arcs from \( u \in V \). We denote by \( \pi[u](t_u) \), the label of \( u \) for a (tentative) presence-time at \( u \in V \), given that the presence time at the origin \( s \) is \( t_s \). Clearly, \( \pi[u](t_u) \) cannot be considered as a departure-time from \( u \) for all the elementary connections emanating from it, for commuters starting their journey from \( s \) at time \( t_s \) (or later). Transfer-times within \( u \) should also be taken into account, in case that commuters have to exchange vehicles to continue their trip. In addition, \( \delta[u](t_u) \) denotes the earliest presence-time at \( u \) that would eventually be returned by a time-dependent shortest-path algorithm.

Given two time values \( t \) and \( t' \), the function \( \Delta(t, t') \in [0, T_p) \) computes the duration of the interval \([t, t')\), taking into account the periodicity of reported times, as well as the fact that each reported time value concerns either the current or the next period:

\[
\Delta(t, t') = \begin{cases} 
  t' - t & \text{if } t' \geq t \\
  T_p + t' - t & \text{otherwise.} 
\end{cases}
\]

The travel-time \( \Delta(c) \) of an elementary connection \( c \) is the elapsed time between the departure from its origin and the arrival at its destination, i.e., \( \Delta(c) = \Delta(t_d(c), t_a(c)) \).

In the models described below, each elementary connection \( c \) is associated with an arc \( c = (u, v) \in E \), while \( C(c) \) denotes the set of elementary connections associated with \( c \). The duration \( D[c](t_u) \) of the elementary connection \( c \) depends on the (tentative) presence-time \( t_u \) at \( u \), and the corresponding travel-time for \( c \) : \( \forall t_u \geq 0, D[c](t_u) = \Delta(t_u, t_a(c)) + \Delta(t_d(c), t_a(c)) \).

At any station \( B \in B \), it is possible for a commuter to be transferred from one vehicle to another. The transfer-time for this exchange of vehicles is station-specific, and is given by the value \( \text{trans}(B) \). Such a transfer is only meaningful if the time-difference of the departure-time of the outgoing vehicle from \( B \) minus the arrival-time of the incoming vehicle at \( B \), along the journey of the commuter, is at least equal to \( \text{trans}(B) \). An itinerary of a timetable \( T \) is a sequence of elementary connections \( P = (c_1, c_2, ..., c_i, c_{i+1}, ..., c_k) \), where for each \( i = 2, 3, ..., k, S_a(c_{i-1}) = S_d(c_i) \) and also

\[
\Delta(t_a(c_{i-1}), t_d(c_i)) \geq \begin{cases} 
  0 & \text{if } Z(c_{i-1}) = Z(c_i) \\
  \text{trans}(S_a(c_{i-1})) & \text{otherwise.} 
\end{cases}
\]

According to the itinerary \( P \), a commuter departs from station \( S_a(c_1) \) at time \( t_d(c_1) \) and arrives at station \( S_a(c_k) \) at time \( t_a(c_k) \). If the commuter’s presence-time at \( S_a(c_1) \) is \( t_u \), then the corresponding travel-time of \( P \) is defined as follows: \( \Delta[P](t_u) = \Delta(t_u, t_d(c_1)) + \Delta(t_d(c_1), t_a(c_1)) \). A trip \( J = (c_1, c_2, ..., c_k) \) is a special case of an itinerary that is performed by only one vehicle. Thus, it must hold that \( \Delta(t_a(c_{i-1}), t_d(c_i)) \geq 0 \) and \( Z(c_{i-1}) = Z(c_i) \), for each \( 2 \leq i \leq k \). A route is a subset of trips in the timetable which follow exactly the same sequence of stops, obviously at different times.

A timetable query is defined by a tuple \( (S, T, t_s) \) where \( S \in B \) is the departure station, \( T \in B \) is the arrival station, and \( t_s \) is the presence-time at \( S \). As mentioned above, the most commonly used optimization criteria for the MJP problem are the earliest arrival (EA) and the minimum number of transfers (MNT), that define the following variants.
Earliest Arrival Multimodal Journey Planning Problem (MJP\textsubscript{EA}): The goal is to find an itinerary that may depart from $S$ no earlier than the presence-time $t_s$ at $S$, and arrives at $T$ as early as possible.

Minimum-Number-of-Transfers Multimodal Journey Planning Problem (MJP\textsubscript{MNT}): The goal is to find an itinerary that departs from $S$ no earlier than the presence-time $t_s$ at $S$, and arrives at $T$ with the minimum number of vehicle exchanges.

In this work we focus on the realistic variant of MJP\textsubscript{EA} which considers non-negligible transfer-times and allows for the existence of non-FIFO arcs.

3 Existing Models and State-Of-Art Review

We summarize the basic characteristics and a comparison of the two most prevalent schedule-based time-dependent graph models, BJ [2] and PSWZ [18], along with their query algorithm.

3.1 The BJ Model

The time-dependent graph $G = (V,E)$ consists of nodes representing stations, and arcs $e = (A,B) \in E$ from station $A$ to station $B$, if there exists in the timetable at least one elementary connection from $A$ to $B$.

The transfers between vehicles within a station are assumed to take zero time. The earliest-arrival time of an arc $e = (A,B)$ is computed “on the fly” and is given by a function $f_{(A,B)}(t_A) = t_B$, where $t_A$ is the presence-time at $A$ and $t_B \geq t_A$ is the earliest-arrival time at $B$. All the elementary connections across $e = (A,B)$ are maintained in an array whose entries consist of tuples of the form $c = (t_d, t_a, Z)$. Figure 1 illustrates an example of the time-dependent model. The BJ model makes the assumption, also known as the FIFO property, that overtaking of vehicles along an arc is not allowed. More formally:

\begin{itemize}
  \item Assumption 1 (FIFO Arcs). For any two given stations $A$ and $B$, there are no two vehicles leaving $A$ and arriving to $B$ such that the vehicle that leaves $A$ second arrives first at $B$.
\end{itemize}

3.2 The PSWZ model

The PSWZ model is an extension of the BJ model and is also based on a time-dependent digraph $G = (V,E)$, which is called the vehicle-route graph. The vehicle exchanges at stations are now allowed to take (either constant, or varying) non-negligible times. For simplicity we consider the case of constant transfer-time per station. In the PSWZ model the set of
vehicles is divided in vehicle routes, where two vehicles belong to the same route if they pass through exactly the same sequence of stations, probably at different times within a day. For each station that a route stops by, the vehicle-route graph contains a node to indicate that event, called a route node. Moreover, the vehicle-route graph contains station nodes corresponding to stations. The arcs are distinguished in three different types: route arcs between route nodes of the same route, transfer arcs from a route node to a station node, and boarding arcs from a station node to a route node. The cost of route arcs is assigned “on the fly”, while the cost of the transfer and boarding arcs are predetermined. In particular, the arrival-time of a route arc \( e = (p^A_i, p^B_i) \) is given by \( f(p^A_i, p^B_i)(t_A) = t_B \) where \( t_B \) is the time that \( p^B_i \) will be reached, given that \( p^A_i \) was reached at time \( t_A \). Its cost is then \( \Delta(t_A, t_B) \). A transfer-arc \((A, p^A_i)\) from a station-node \(A\) to a route node \(p^A_i\) has cost equal to \( \text{trans}(A) \). A boarding arc \((p^A_i, A)\) from a route node \(p^A_i\) to a station node \(A\) has zero cost. The elementary connections from a station to another are maintained in a separate array per route arc, ordered in increasing departure times.

The PSWZ model is based on the following FIFO assumption.

**Assumption 2 (FIFO Vehicle Routes).** There exist no two vehicles \( Z_1, Z_2 \in Z \) belonging to the same vehicle-route such that the (slow) vehicle \( Z_1 \) departs earlier than the (fast) vehicle \( Z_2 \) from a station \( A \) but it arrives later than \( Z_2 \) at the next station \( B \) along the route.

If this assumption is violated, the PSWZ model can enforce it by introducing new vehicle routes, one for each different speed class, where all vehicles follow the same schedule as before. Figure 2 illustrates an example of this model.

### 3.3 Query algorithm for BJ and PSWZ models

The query algorithm used by both models is a variant of Dijkstra’s algorithm [6] which solves the simplified version (i.e., when Assumption 1 holds) of MJP\(_{EA}\) in the BJ model, and the realistic version (i.e., when Assumption 2 holds) of MJP\(_{EA}\) in the PSWZ model.
In particular, given a query \((S, T, t_s)\), the query algorithm is a time-dependent variant of Dijkstra’s algorithm (we call it TDD): Initially the presence-time label \(\pi[S](t_s)\) of the origin-station \(S\) is initialized to \(t_s\), and all other labels are set to infinity. The costs of the transfer and boarding arcs in the PSWZ model are all predetermined. The costs of each time-dependent arc \(e = (A, B)\) (i.e., every arc in the BJ model, only route arcs in the PSWZ model) is computed “on the fly”, when its tail \(A\) is selected by TDD for settling its label. The label \(\pi[A](t_s)\) of \(A\) is optimal when it is chosen to be settled, due to the correctness of TDD in time-dependent graphs whose arc costs obey the FIFO property: \(\delta[A](t_s) = \pi[A](t_s)\). The minimum travel-time \(D[e](\pi[A](t_s)) = \min_{i \in C(e)} \{ D[e](\pi[A](t_s)) \} \) of arc \(e\) is then easily computed. TDD considers updating the label of \(B\), due to the relaxation of \(e\): \(\pi[B](t_s) = \min \{ \pi[B](t_s), \pi[A](t_s) + D[e](\pi[A](t_s)) \} \).

It is also known, due to Assumptions 1 and 2, which is the next elementary connection to be used to reach node \(B\) via \(A\) along the particular arc \(e = (A, B)\), as early as possible: the first one that departs no earlier than the presence-time \(\pi[A](t_s)\) at \(A\). This connection can be easily found by conducting a binary search on the array \(C(e)\), whose elementary connections are ordered in non-decreasing departure-times.

The time complexity of the above algorithm is \(O(m \log(W) + n \log(n))\) [18], where \(n\) and \(m\) are the number of nodes and the number of arcs of the time-dependent graph, respectively, and \(W\) is the maximum number of elementary connections of an arc.

### 3.4 Comparison of BJ vs PSWZ and Other Approaches

Both models have their strengths and weaknesses. The BJ model is based on a digraph where each node represents a station and each arc between two nodes represents the existence of at least one elementary connection between them. The PSWZ model, on the contrary, considers a digraph which contains, in addition to station nodes, route nodes and route arcs that correspond to elementary connections for a given route, and transfer and boarding arcs connecting station nodes with route nodes. In particular, assume that we have a timetable involving a set \(B\) of stations and a set \(C\) of elementary connections is given. The BJ model considers a time-dependent digraph \((V_{BJ}, E_{BJ})\) with \(|V_{BJ}| = |B|\) vertices and \(|E_{BJ}| \leq |C|\) arcs. The PSWZ model, on the other hand, considers a digraph \((V_{PSWZ}, E_{PSWZ})\) with \(|V_{PSWZ}| \in |B| + O(|C|)\) vertices and \(|E_{PSWZ}| \in O(|B| + |C|)\) arcs: each station \(S \in B\) corresponds to a station-node \(v^S\) and to a constant number of route-nodes \(p^S\), for all the passing-by routes from \(S\); \(S\) also induces a constant number of transfer and boarding arcs between \(v^S\) and each of \(p^S\). Finally, the number of route-arcs is, again, at most \(|C|\).

The space and query-time requirements in the PSWZ model are still linear in the size of the timetable, but clearly larger than those in the BJ model. On the other hand, the extra nodes and arcs in the PSWZ model make the model more realistic, since it can handle both non-negligible transfer times and violations of the FIFO property when moving from one station to another (possibly via vehicles of different types). The simplicity of the BJ model (and thus smaller space and query-time requirements) is due to the fact that it neglects transfer times and assumes universal enforcement of the FIFO property, for all pairs of stations connected by at least one elementary connection. These assumptions make the BJ model not applicable in real-world instances. In conclusion, the BJ model is simpler, lighter and faster than the PSWZ model, but the PSWZ model overcomes the BJ model in applicability, because it handles more realistic scenarios.

Other approaches of public transport networks representation concern vector-based models. Characteristic representatives are RAPTOR [4] that works in rounds where in the \(i\)-th round it discovers the earliest arrival time to every stop by using at most \(i - 1\) transfers, CSA [5]
that assumes that the time table is not cyclic – there are no connections after midnight – and scans a single array of connections containing traveling events sorted in ascending order of departure time, and Trip-Based Routing [23] which requires the precomputation of transfers between traveling events and also works in rounds, where each round scans segments of trips that are reached in the previous round. The basic advantage of vector-based models relies on the vector-cache-friendly processing of the traveling events. In this work we focus only on graph-based models.

# 4 A novel time-dependent graph model

In this section we present the non-FIFO time dependent graph model with REalistic vehicle eXchange times (REX), which aims to keep the simplicity of the digraph in the BJ model, but also to support the existence of non-negligible transfer times between stations and non-FIFO abiding arcs. The ambition of REX is to guarantee the strong points of both BJ and PSWZ models, without suffering from their weaknesses.

Since the FIFO property is not a precondition for our time-dependent digraph, we can no longer use TDD as our query algorithm. We therefore present a novel label-correcting query algorithm, called TRIPLA, that computes optimal earliest-arrival routes within our model. Clearly, since REX insists on the simplicity of the graph, more work has to be done by TRIPLA. Nevertheless, rather than conducting blind label-corrections until optimality is reached, TRIPLA uses a novel label-correction propagation process, which conducts targeted label corrections only across affected routes in the digraph, upon improving the label of a particular node. In the rest of this section we first present the REX model, then we continue with the description of the TRIPLA query algorithm.

## 4.1 The REX model

REX is based on the time-dependent graph $G = (V, E)$ of the BJ model [2]. We add three additional attributes to each elementary connection of BJ: $c = (S_d, S_a, t_d, t_a, Z, t_r, p_{next}, p_{prev})$. The attribute $p_{next}(c)$ (resp. $p_{prev}(c)$) is a pointer indicating the next (resp. previous) elementary connection $c'$ (resp. $c''$) along the same trip with $c$ using the same vehicle $(Z(c') = Z(c) = Z(c''))$, or is set to null when no such connection exists. As for $t_r(c)$, it indicates a tentative estimation of the earliest-arrival time at $S_d(c)$ using the particular vehicle $Z(c)$ considered by $c$. The initial value of $t_r(c)$ is set to $\infty$ and changes “on the fly” when the previous elementary connection $c' = p_{prev}(c)$ is relaxed.

REX maintains the set $C(c)$ of all elementary connections for $e = (A, B)$ in an array $\textit{ConnectionArray}_c$, which is ordered in non-decreasing arrival-times at $B$, rather than departure-times from $A$ (as in the BJ and the PSWZ models). For any given elementary connection $c \in C(e)$ along an arc $e = (A, B)$, the boarding-time $t_z(A)$ on $Z(c)$ after a vehicle exchange at $A$ is station-dependent and is computed as $t_z(A) = \pi[A](t_s) + \textit{trans}(A)$. On the other hand, all commuters arriving at $A$ with the vehicle $Z(c)$ do not need to make a vehicle exchange in order to get on-board and continue their itinerary with $c$. Therefore, their on-board-time in that case is exactly the value $t_r(c)$, which may only have a finite value if at least one route has been discovered that departs from the origin and has already used some previous elementary connection of the vehicle $Z(c)$. Otherwise, $t_r(c) = \infty$.

The arrival-time at $B$ via the elementary connection $c \in C(e)$ is computed by the function $g_c(t_z(A))$ as $g_c(t_z(A)) = \min \{ t_z(A) + D[c](t_z(A)) , t_r(c) + D[c](t_r(c)) \}$. This function considers two different scenarios for commuters willing to use $c$ as the next leg of their itineraries. They have either hopped on the vehicle $Z(c)$ earlier than station $S_d(c)$, or
they will do it exactly at this station. Since we refer to the same elementary connection \( c \)

involving the same vehicle but possibly for different periods of the timetable, the function \( f_c(t) = t + D[c](t) = t_d(c) + k \cdot T_p \)

\( k \cdot T_p + t_d(c) \geq t > (k-1) \cdot T_p + t_d(c) \) is a non-decreasing step function. Therefore, \( g_c(t_A(A)) \) is also non-decreasing. The arrival-time at \( B \) via any connection of \( e = (A, B) \), as a function of the boarding time \( t_x(A) \) within \( A \) (after a vehicle exchange) is then \( f_c(t_x(A)) = \min_{e \in \mathcal{C}(c)} \{ g_e(t_x(A)) \} \).

Besides the connection arrays (for arcs), we also introduce a new data structure per node, the Index Array (cf. Figure 3). For each \( A \in V \), we consider the sets \( \mathcal{E}_A = \{ (A, X_i) \in E : 1 \leq i \leq k \} \) of outgoing arcs, \( \mathcal{N}_A = \bigcup_{e \in \mathcal{E}_A} \mathcal{C}(e) \) of elementary connections departing from \( A \), and \( \mathcal{I}_A = \bigcup_{e \in \mathcal{N}_A} \{ t_d(e) \} \) of departure-events at \( A \). Then, \( \text{IndexArray}_{A} \) contains pairs \((t_x, P_x)\) where \( t_x \in \mathcal{I}_A \) is a departure-event and \( P_x \) is an array of (pointers to) elementary connections: \( \forall e \in \mathcal{E}_A \), \( P_x[e] \) indicates the first elementary connection in \( \text{ConnectionArray}_{e} \)

having \( t_d(c) \geq t_x \). The number of the pairs in \( \text{IndexArray}_{A} \) is equal to \( |\mathcal{I}_A| \). The pairs of \( \text{IndexArray}_{A} \) are ordered in increasing departure-events \( t_x \). The number of (pointers to) elementary connections in each array \( P_x \) is \( |\mathcal{E}_A| \).

Given the presence time \( t_x(A) \) at station \( A \) (after having already gone off-board from the previous vehicle), \( \text{IndexArray}_{A} \) allows the computation of the first elementary connection \( c_i \in \text{ConnectionArray}_{A, X_i} \), for each outgoing arc from \( A \), having \( t_d(c_i) \geq \pi[A](t_x) \). This computation requires only one binary search in \( \text{IndexArray}_{A} \), so as to find the earliest departure event \( t_x \geq \pi[A](t_x) \). Then, the elementary connection \( c_i = P_x[A, X_i] \) has the earliest-arrival time at \( X_i \) according to the schedule, among all connections of \( (A, X_i) \) with departure time at least \( \pi[A](t_x) \). This is because \( \text{ConnectionArray}_{A, X_i} \) is ordered by non-decreasing arrival times at \( X_i \). All the index arrays of the stations are precomputed during a preprocessing phase. Their preprocessing-space requirement is linear in the size of the time-dependent graph, assuming that each node \( A \) has a constant number \( |\mathcal{I}_A| \in O(1) \) of departure events during the entire period \([0, T_p]\) of the timetable: \( \sum_{A \in V} |\mathcal{I}_A| (|\mathcal{E}_A| + 1) \in O(1) \cdot \sum_{A \in V} (|\mathcal{E}_A| + 1) = O(|E| + |V|) \). In the worst case, there exist \( T_p \) departure events at each node, \(|\mathcal{I}_A| \leq T_p\), implying worst-case space \( O(T_p \cdot (|E| + |V|)) \).

Finally, we construct the CheckArray data structure for the arcs, whose role is to jointly describe chains of elementary connections (comprising trips) along which our query algorithm will have to perform (targeted) label-correction propagations for the attributes \( t_x(c) \) of these connections. In particular, for \( e = (A, B) \in E \), the array \( \text{CheckArray}_{e} \) is initially empty, and is augmented with elementary connections during the query algorithm’s execution. For two incident arcs \( e = (X, Y) \) and \( e' = (Y, Z) \), the elementary connection \( c \in \mathcal{C}(e) \) is appended to \( \text{CheckArray}_{e} \) when, for the next elementary connection \( c' = p_{next}(c) \in \mathcal{C}(e') \) along the trip of \( Z(c) \), the query algorithm is in position to conclude that the boarding time \( t_x(Z) \) at \( Z \) is suboptimal, that is, \( Z \) could have been reached earlier if \( c \) had been relaxed before \( c' \).

Consider the following example (Figure 4): Assume that a 1-minute time unit is used, and \( T_p = 1440 \). \( B \) is settled before \( A \) and \( C \), since the \( \pi[B](t_x) = 4 < \pi[C](t_x) = 15 < \pi[A](t_x) = \infty \), and has boarding time (after vehicle exchange) \( t_x(B) = \pi[B](t_x) + \text{trans}(B) = 12 \). Assume that \( B \) realizes that \( t_x(c_{21}) = \infty \) and \( t_x(c_{23}) = \infty \), i.e., the vehicles \( M \) and \( K \) have not been considered yet by some of its predecessor stations for the routes of \( M \) and \( K \). Unavoidably, these two vehicles may be used only after an exchange of vehicles at station \( B \): \( g_{c_{21}}(12) = \min \{12 + D[c_{21}](12), \infty\} = 1450 \), and \( g_{c_{23}}(12) = \min \{12 + D[c_{23}](12), \infty\} = 1454 \). If, on the other hand, \( c_{11} \) and \( c_{14} \) become relaxed at some time after the settlement of \( B \), then it would hold that \( t_x(c_{11}) = 8 \) and \( t_x(c_{23}) = 11 \). As a result, the valuations of \( g_e \) for the two connections has to be updated accordingly: \( g_{c_{21}}(12) = \min \{12 + D[c_{21}](12), 8 + D[c_{21}](8)\} = \)}
The index array of a node $A$. For $t_δ = 7$, the (pointer to the) elementary connection $c_i = P_7[(A,X_i)]$ indicates the first (in order of $ConnectionArray(A,X_i)$) connection having an eligible departure time $t_d(c_i) \geq 7$, therefore also providing the earliest arrival time at $X_i$ among eligible connections.

$10 < 1450$ and $g_{c_{23}}(12) = \min\{12 + D[c_{23}](12), 11 + D[c_{23}](11)\} = 14 < 1454$. Therefore, upon settlement of $B$, $CheckArray_{c_1}$ should be augmented with $c_{11}$ and $c_{14}$ so that, if these two connections are ever relaxed in the future, the changes in the values $t_r(c_{11})$ and $t_r(c_{14})$ are updated accordingly. As for $c_{15}$, since $t_r(c_{15}) \geq 13 > t_x(B) = 12$, there is no need to be added in $CheckArray_{c_1}$.

4.2 TRIPLA: Query algorithm for REX

We present now the TRIP-based Label-correction propagation (TRIPLA) query algorithm for MJP$_{EAP}$, in the REX model. Due to possible violation of the FIFO property, TRIPLA cannot be a label-setting algorithm. Nevertheless, it is built as a time-dependent variant of Dijkstra’s algorithm, with a priority queue storing each node $A$ with its label (presence-time) $\pi[A](t_s)$, and relaxes (possibly more than once) elementary connections of arcs using the auxiliary data structures described in Section 4.1, according to an appropriate label-correction propagation (LCPROP) phase.

Given an EA query $(S,T,t_s)$, the algorithm, after an initialization phase, executes a number of iterations until the destination station is extracted from the priority queue. Here is the high-level description of TRIPLA.

**Initialization.** $S$ is inserted into the priority queue with label $\pi[S](t_s) = t_s$, and the transfer-time of $S$ is set to $\text{trans}(S) = 0$.

**Iteration.** A new node $A$ is extracted from the priority queue. The earliest arrival time $\delta[A](t_s)$ at station $A$, as we prove later, is equal to $\pi[A](t_s)$ and therefore $A$ is settled. Consequently, all the outgoing arcs from $A$ are scanned. For each $e := (A,B) \in E$ s.t. $B$ has not been settled yet, a relaxation phase starts. Otherwise, the label-correction propagation (LCPROP) phase starts. TRIPLA returns the earliest arrival time to $T$ when $T$ is settled.
We shall now describe the relation and label-correction propagation phases.

**Relaxation phase.** Let $e = (A, B) \in E$. Since the FIFO property does not apply, an elementary connection that departs first from $A$ is not necessarily the one that arrives first at $B$. Therefore, multiple elementary connections of an arc must be now relaxed. If $c_i = ConnectionArray_{c}[i]$ $(1 \leq i \leq k)$ is the first elementary connection of $ConnectionArray_c$, having departure time $t_d(c_i) \geq \pi[A](t_s)$, then the elementary connections towards $B$ that have to be relaxed are in the ordering $\langle c_1, c_{i+1}, ..., c_k, c_1, c_2, ..., c_{i-1} \rangle$, up to an elementary connection $c_p$, where $1 \leq p \leq k$, such that $c_p$ is the first elementary connection within the ordering for which $\pi[A](t_s) + D[c_p](\pi[A](t_s)) \geq t_x(B) = \pi[B](t_s) + \text{trans}(B)$.

With a binary search in $IndexArray_{A,c}$, TRIPLA can locate $c_1$. It then sequentially scans the connections of $ConnectionArray_{c}$, until $c_p$ is discovered. Consequently, the arrival-time of $c_p$ is computed and the connection to follow (along the same trip) $c'_p = p_{\text{next}}(c_p)$, if it exists, updates its attribute $t_r(c'_p)$ accordingly.

Let the arcs $e_1 = (A, B)$, $e_2 = (B, C)$, $e_3 = (C, D)$ and the elementary connections $c_1 \in C(e_1), c_2 \in C(e_2), c_3 \in C(e_3)$ where $p_{\text{next}}(c_1) = c_2$ and $p_{\text{next}}(c_2) = c_3$. The arrival-time of $c_2$ at $C$ is updated as $w_{c_2} = g_{c_2}(t_x(B)) = \min\{ D[c_2](t_x(B)) + t_x(B), D[c_2](t_r(c_2)) + t_r(c_2) \}$ where $t_r(c_2)$ is the on-board arrival-time of $c_2$ to $B$ using vehicle $Z(c_2) = Z(c_1)$ (i.e., without vehicle exchange), and $t_x(B) = \pi[B](t_s)$ is the boarding time (after vehicle exchange) at $B$. So long as $\pi[A](t_s) > \pi[B](t_s)$, $B$ may be extracted from the priority queue only before $A$, and thus the relaxation of $c_2$ would take place before $t_r(c_2)$ is computed, i.e., $t_r(c_2) = \infty$ at that time. Let $t'_r(c_2)$ be the optimal value of the on-board arrival-time of $Z(c_2)$ at $B$, and $w'_{c_2}$ be the corresponding arrival-time of $c_2$ at $C$. If it holds $t_x(B) = \pi[B](t_s) + \text{trans}(B) > t'_r(c_2)$, then $w_{c_2} \geq w'_{c_2}$, due to the monotonicity of $g_c$.

![Figure 4](image-url) Example of CheckArrays.

**ConnectionArray**

<table>
<thead>
<tr>
<th>$c_1$</th>
<th>$c_2$</th>
<th>$c_3$</th>
<th>$c_4$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t_d$</td>
<td>$t_r$</td>
<td>$t_x$</td>
<td>$t_r$</td>
</tr>
<tr>
<td>5</td>
<td>6</td>
<td>M</td>
<td>6</td>
</tr>
<tr>
<td>9</td>
<td>9</td>
<td>N</td>
<td>9</td>
</tr>
<tr>
<td>9</td>
<td>6</td>
<td>L</td>
<td>7</td>
</tr>
<tr>
<td>11</td>
<td>7</td>
<td>K</td>
<td>7</td>
</tr>
<tr>
<td>13</td>
<td>10</td>
<td>Y</td>
<td>10</td>
</tr>
</tbody>
</table>

**CheckArray**

- $\pi[A](t_s) = \infty$
- $\pi[B](t_s) = 4$
- $\pi[C](t_s) = 15$

- $\text{trans}(A) = 5$
- $\text{trans}(B) = 8$
- $\text{trans}(C) = 3$

- $e_1$
- $e_2$
- $e_3$
relaxation conditions yet). Before ending LCPROP, extracted from the priority queue, a connection emanating from it are not affected (their arrival-times have not been computed if the arrival-node (i.e., its arrival-station has not been extracted from the priority queue yet, thus the Elementary
is the case, it inserts its preceding connection along the trip of
In the LCPROP phase for
The optimal arrival-time via
We illustrate the LCPROP phase regarding the re-relaxation of certain arcs through the
\( g_e(t_x(B)) > g_e(t_x(S_d(c))) \), and updates accordingly \( t_r(c') \), where \( c' = p_{next}(c) \). It then recomputes the arrival-time of \( c'' \) and updates accordingly \( t_r(p_{next}(c'')) \), and so on. The LCPROP phase stops when an elementary correction \( c' \) is considered which provides clearly suboptimal arrival-time at its arrival-node (i.e., \( t_x(S_o(\tilde{c})) \) \( \leq t_x(\tilde{c}) \), or \( S_o(\tilde{c}) \) has not been settled yet. In the latter case, the arrival-station has not been extracted from the priority queue yet, thus the elementary connections emanating from it are not affected (their arrival-times have not been computed yet). Before ending LCPROP, \( t_r(\tilde{c}) \) is also computed, and \( \pi[S_o(\tilde{c})](t_x) \) is updated if necessary.

We illustrate the LCPROP phase regarding the re-relaxation of certain arcs through the example of Figure 5, where a 1-minute time unit is considered, and \( T_p = 1440 \). Consider the arcs \( e_1 = (A, B) \), \( e_2 = (B, C) \), \( e_3 = (C, D) \), \( e_4 = (D, X) \), \( e_5 = (X, Q) \), the elementary connections \( c_1, c_2, c_3, c_4, c_5 \), where \( c_i \in C(e_i) \), \( 1 \leq i \leq 5 \), and the trip \( P = (c_1, c_2, c_3, c_4, c_5) \) from \( A \) to \( Q \). The transfer times are shown below the station-nodes. The departure-times \( t_d \), the arrival-times \( t_a \) and the vehicles \( Z \) of the connections are shown on the right in Figure 5. The presence time at the origin-node \( S \) is \( t_a = 2 \). At a certain point during the execution of TRIPLA, the labels of the nodes \( A, B, C, D, X, Q \) are also shown in Figure 5. When \( B \) is extracted from the priority queue, \( t_r(c_2) \) is still \( \infty \) because \( A \) is not settled yet. Therefore, a transfer must occur at \( B \) and the boarding time at \( B \) is \( t_x(B) = 25 > t_d(c_2) = 21 \). TRIPLA will then relax \( c_2 \) with arrival-time \( w_{c_2} = g_{c_2}(25) = \min\{D[c_2](25) + 25, \infty\} = 1464 \), and will
The source of the pedestrian network is Berlin and Switzerland. The source of the timetable data for London is format, for the rest is format. The integrated networks concern the metropolitan areas of Athens, Rome, London, and (b) road and pedestrian network data sets in the Open Street Map (OSM) format. The input data used to implement the multimodal transport networks are (a) timetable data sets in the General Transit Feed Specification (GTFS) format, containing various means of public transport, and (b) road and pedestrian network data sets in the Open Street Map (OSM) format. The integrated networks concern the metropolitan areas of Athens, Rome, London, Berlin and Switzerland. The source of the timetable data for London is [17], for the rest is [16]. The source of the pedestrian network is [12].

![Figure 5](image)

Figure 5 A trip \( P = (c_1, c_2, c_3, c_4, c_5) \) from A to Q, whose elementary connections are shown in the table. The transfer-time per station is indicated by \( \text{trans} \). At a certain point during the algorithm’s execution, the labels of the nodes get the values shown in the figure.

Also update \( c_3 \) with \( t_r(c_3) = 1464 \). TRIPLA will also insert \( c_1 \) to \( \text{CheckArray} \), since \( c_2 \) fulfills all the Re-relaxation Conditions. Node \( C \) will be extracted next from the priority queue. Its boarding time (after vehicle exchange) is \( t_x(C) = 26 \). TRIPLA will relax \( c_3 \) with the arrival-time \( w_{c_3} = g_{c_3}(26) = \min \{ D[c_3](26) + 26, D[c_3](1464) + 1464 \} = 1467 \) and thus, \( c_4 \) will be updated with \( t_r(c_4) = 1467 \). Consequently, TRIPLA will relax \( c_4 \) with the arrival-time \( w_{c_4} = g_{c_4}(1467) = \min \{ D[c_4](1467) + 1467, \infty \} = 1469 \) and will also update \( c_5 \) with \( t_r(c_5) = 1469 \). Observe that, after the relaxation of \( c_4 \), \( X \) will still have label \( \pi[X](t_x) = 50 \) instead of 29, because \( t_r(c_4) = 1467 \) and not 27. Node \( D \) will be extracted next from the priority queue. Its boarding time (after vehicle exchange) is \( t_x(D) = 19 + 9 = 28 \). TRIPLA will relax \( c_5 \) with the arrival-time \( w_{c_5} = g_{c_5}(28) = \min \{ 28 + D[c_5](28), 1467 + D[c_5](1467) \} = 1469 \) and thus, \( c_5 \) will be updated with \( t_r(c_5) = 1469 \). Consequently, since \( t_x(X) = 50 + 5 = 55 \), TRIPLA will relax \( c_5 \) with the arrival-time \( w_{c_5} = g_{c_5}(55) = \min \{ 55 + D[c_5](55), 1469 + D[c_5](1469) \} = 1470 \).

Now is the time for \( A \) to be extracted from the priority queue. TRIPLA computes the arrival-time of \( c_1 \) and updates accordingly \( t_r(c_2) = 23 \). All the subsequent elementary connections of \( c_1 \) along the same trip must be re-relaxed with as follows: \( w_{c_2} = 24 \Rightarrow t_r(c_2) = 24, w_{c_3} = 27 \Rightarrow t_r(c_3) = 37, w_{c_4} = 29 \Rightarrow t_r(c_5) = 29 \). Since \( X \) has not been settled yet, the LCPROP phase updates also its label: \( \pi[X](t_x) \) to 29. As a consequence, before being extracted, station \( X \) has its own label corrected to the optimal value.

For more details on the pseudocode of TRIPLA, its proof of correctness, the \( O(1) \) time-complexity of LCPROP and the \( O(m + n \log(n)) \) time-complexity of TRIPLA for real-world instances, the reader is deferred to the full version of this paper.

5 Experimental Evaluation

In this section, we present the experimental evaluation of the TRIPLA algorithm. All the experiments have been performed on a workstation equipped with an Intel Xeon CPU E5-2643 v3 3.40 GHz and 256 GB RAM. All algorithms were implemented in C++ and compiled with gcc (v7.5.0, optimization level O3) and Ubuntu Linux (18.04 LTS). The input data used to implement the multimodal transport networks are (a) timetable data sets in the General Transit Feed Specification (GTFS) format, containing various means of public transport, and (b) road and pedestrian network data sets in the Open Street Map (OSM) format. The integrated networks concern the metropolitan areas of Athens, Rome, London, Berlin and Switzerland. The source of the timetable data for London is [17], for the rest is [16]. The source of the pedestrian network is [12].
Table 1 contains detailed information concerning the input timetables. It contains the number of stations |B| and the number of elementary connections |C| between stops (a proxy size), along with the number of nodes |V| and arcs |E| of the graph-based models REX and MDTM [7], which is the most efficient time-expanded model. The departure time for a query is within 1, 2 or 7 days. The timetable time period of the connection sets is the valid departure period plus two days, starting from Monday. It is evident from Table 1 that the graph of REX is much smaller than that of MDTM. The corresponding Table for maximum walking time 600 secs, is included in the full version of this paper.

Table 1 Benchmark instances and sizes of corresponding graphs; restricted walking: 300 sec.

| Period   | Map   | |B| | |C| | Transfers | MDTM | REX |
|----------|-------|---|---|---|---|---|---|---|---|
| One day  | Athens| 6771 | 2178677 | 27734 | 2185448 | 6506253 | 6771 | 31980 |
|          | Rome  | 6883 | 2551316 | 27972 | 2558199 | 7592671 | 6883 | 33606 |
|          | London| 19706| 13391869 | 81798 | 13411575 | 39901166 | 19706 | 96436 |
|          | Berlin| 27917| 42229299 | 73445 | 4250846 | 12530654 | 27917 | 110339 |
|          | Switz.| 26757| 6639655 | 36112 | 6666412 | 19412126 | 26757 | 84847 |
| Two days | Athens| 6771 | 2904772 | 27734 | 2911543 | 8653372 | 6771 | 31980 |
|          | Rome  | 6893 | 3402181 | 28424 | 3409074 | 10115850 | 6893 | 34071 |
|          | London| 19706| 17839626 | 81836 | 17859332 | 53125144 | 19706 | 96468 |
|          | Berlin| 27920| 5630882 | 73461 | 5658802 | 16683977 | 27920 | 110372 |
|          | Switz.| 26805| 8855105 | 36268 | 8881910 | 25877261 | 26805 | 85225 |
| Seven days| Athens| 7041 | 4603557 | 28524 | 4610598 | 13717448 | 7041 | 32971 |
|          | Rome  | 6917 | 5502358 | 28576 | 5509275 | 16343179 | 6917 | 34405 |
|          | London| 19706| 29979408 | 82216 | 29999114 | 89224052 | 19706 | 96854 |
|          | Berlin| 28096| 8794883 | 74933 | 8822979 | 26021451 | 28096 | 112345 |
|          | Switz.| 27468| 14252368 | 38008 | 14279836 | 41586992 | 27468 | 90422 |

Our implementation is engineered by applying a series of algorithmic optimizations, the most important of which which we present next. Further optimizations and extensions of REX and TRIPLA, such as heuristic methods aiming to boost the performance of TRIPLA (one trying to avoid unnecessary binary searches in our data structures, and one that tries to accelerate TRIPLA in the rationale of ALT [8]) including the integration of walking, are described in the full version of the paper.

- **Graph representation**: A static forward-star array-based and cache-friendly variant of the PGL library [9] was used for the graph representation.
- **Priority queue**: For Dijkstra-based algorithms, we used as priority queue Sanders’ cache-friendly implementation\(^1\) of the sequence heap [19].
- **Vertex reordering**: Similar to well-known observations concerning performance enhancements on Dijkstra-based core processing steps [3, 20], we reorder the vertices of the graph so that neighboring vertices are located in adjacent memory blocks. This way, the cache misses and run time are decreased. That re-ordering is formed with respect to a combination of DFS and BFS traversal of the graph.
- **Data allocation**: We order the required data (e.g., distances, predecessors, and time event containers) of all the algorithms for each vertex and arc, to enforce a contiguous memory allocation and thus decrease the cache misses on memory access operations.

\(^1\) [http://algo2.iti.kit.edu/sanders/programs/spq](http://algo2.iti.kit.edu/sanders/programs/spq)
The experimental evaluation compares TRIPLA with some of the fastest state-of-art routing algorithms (CSA [5], ULTRA+CSA [1] and MDTM-QH-ALT [7]). The input for ULTRA preprocessing are the limited walking graphs. For each input we generated 10,000 random queries, and reported average execution times (in ms). Table 2 shows the performance of the algorithms when the departure time of a query is within one day, two days or seven days, to demonstrate how the increment of the timetable period affects query times. We observe that TRIPLA has faster average query times in all cases. Especially in Switzerland, TRIPLA is at least 2.5 times faster than all other algorithms.

<table>
<thead>
<tr>
<th>Map</th>
<th>Algorithm</th>
<th>QT [ms] - 1d (300)</th>
<th>QT [ms] - 1d (600)</th>
<th>QT [ms] - 2d (300)</th>
<th>QT [ms] - 2d (600)</th>
<th>QT [ms] - 7d (300)</th>
<th>QT [ms] - 7d (600)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Athens</td>
<td>CSA</td>
<td>1.52 6.88</td>
<td>1.52 6.69</td>
<td>2.07 7.54</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>ULTRA + CSA</td>
<td>0.43 0.64</td>
<td>0.47 0.67</td>
<td>1.12 0.89</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>MDTM-QH-ALT</td>
<td>0.72 0.93</td>
<td>0.81 1.00</td>
<td>1.14 1.36</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>TRIPLA</td>
<td>0.31 0.49</td>
<td>0.32 0.49</td>
<td>0.52 0.69</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Rome</td>
<td>CSA</td>
<td>1.59 5.54</td>
<td>1.63 5.76</td>
<td>1.76 6.06</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>ULTRA + CSA</td>
<td>0.59 0.72</td>
<td>0.67 0.80</td>
<td>0.85 0.85</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>MDTM-QH-ALT</td>
<td>0.97 1.17</td>
<td>1.04 1.24</td>
<td>1.29 1.50</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>TRIPLA</td>
<td>0.44 0.62</td>
<td>0.44 0.61</td>
<td>0.62 0.74</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>London</td>
<td>CSA</td>
<td>10.32 78.39</td>
<td>10.55 80.34</td>
<td>11.52 86.13</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>ULTRA + CSA</td>
<td>3.61 4.17</td>
<td>3.81 4.31</td>
<td>4.45 4.80</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>MDTM-QH-ALT</td>
<td>2.17 2.96</td>
<td>2.48 3.02</td>
<td>3.11 3.18</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>TRIPLA</td>
<td>0.98 1.60</td>
<td>1.04 1.43</td>
<td>2.34 1.89</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Berlin</td>
<td>CSA</td>
<td>2.80 15.01</td>
<td>3.02 15.36</td>
<td>4.41 18.14</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>ULTRA + CSA</td>
<td>3.34 3.47</td>
<td>3.56 3.70</td>
<td>5.10 5.28</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>MDTM-QH-ALT</td>
<td>7.19 8.11</td>
<td>7.90 8.84</td>
<td>8.70 9.57</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>TRIPLA</td>
<td>2.71 3.46</td>
<td>2.78 3.55</td>
<td>3.74 4.57</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Switz.</td>
<td>CSA</td>
<td>5.08 6.66</td>
<td>5.38 6.85</td>
<td>8.00 9.14</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>ULTRA + CSA</td>
<td>5.52 5.34</td>
<td>5.85 5.56</td>
<td>8.60 7.87</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>MDTM-QH-ALT</td>
<td>4.48 4.83</td>
<td>5.07 5.48</td>
<td>6.50 6.49</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>TRIPLA</td>
<td>1.75 1.89</td>
<td>1.62 1.71</td>
<td>2.58 2.52</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

6 Conclusions and Future Work

In this work, the REX model for multimodal route planning in schedule-based public transport systems is presented, along with a novel query algorithm, TRIPLA, that efficiently solves the realistic earliest-arrival routing problem. An extensive experimental study on real-world benchmark instances demonstrates that TRIPLA outperforms the state-of-the-art multimodal route planners.

We are currently working on another novel query algorithm, that exploits the REX model to solve the multicriteria variant of the routing problem in schedule-based public-transport systems with walking transfers, where apart from the earliest-arrival objective, the commuters also care for minimizing the number of vehicle exchanges.

References

1 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


OpenStreetMap datasets. https://download.geofabrik.de/europe.html.


