Routing in Multimodal Transportation Networks with Non-Scheduled Lines

Darko Drakulic
NAVER LABS Europe, Meylan, France

Christelle Loiodice
NAVER LABS Europe, Meylan, France

Vassilissa Lehoux[1]
NAVER LABS Europe, Meylan, France

Abstract

Over the last decades, new mobility offers have emerged to enlarge the coverage and the accessibility of public transportation systems. In many areas, public transit now incorporates on-demand transport lines, that can be activated at user need. In this paper, we propose to integrate lines without predefined schedules but with predefined stop sequences into a state-of-the-art trip planning algorithm for public transit, the Trip-Based Public Transit Routing algorithm [30]. We extend this algorithm to non-scheduled lines and explain how to model other modes of transportation, such as bike sharing, with this approach. The resulting algorithm is exact and optimizes two criteria: the earliest arrival time and the minimal number of transfers. Experiments on two large datasets show the interest of the proposed method over a baseline modelling.

2012 ACM Subject Classification Mathematics of computing → Graph algorithms

Keywords and phrases Multimodal routing, on-demand public transportation, bicriteria shortest paths

Digital Object Identifier 10.4230/LIPIcs.SEA.2022.6

1 Introduction

Based on modern public transit routing algorithms, hundreds of trip planning applications are used by millions of users every day. They integrate public transit information with road networks and usually compute itineraries combining only public transportation services with walking transfers, which is often referred to as *public transit routing*. In order to offer a more integrated experience to their users, some applications allow for more multimodality, combining public transportation with other available transportation offers, such as taxis, bike sharing or car sharing. We then speak of *multimodal* or *intermodal routing*.

In addition to the classical scheduled public transportation, many transport authorities propose special transportation offers in sub-urban areas, or for elderly or disabled people. They are usually organized as on-demand services, where transportation authorities define lines (sequence of stops) or areas of coverage, but no fixed schedules. For this type of service, users must "activate" the desired trip by contacting the transport agency. A lot of transport authorities in France offer this type of services, for example in Montauban metropolitan area [1] (non-scheduled lines with only a subset of stops activated), in Flers metropolitan area [24] (on-demand transportation between predefined stations during given time intervals), or in Pays de Dreux [16] (on-demand transportation for elderly people from home place to

[1] Corresponding author

any destination within a zone). Note that some transport authorities also provide on-demand services with predefined schedules, but in that case, a public transit routing algorithm can integrate them as classical scheduled lines in itinerary computations.

In this paper, we propose an extension of the Trip-Based Public Transit Routing (TB) algorithm [30] which is a state-of-the-art algorithm for public transit routing. We modify it to deal with non-scheduled lines, that we define as lines that can be activated on given periods and for which a sequence of stops is defined, as well as possibly time-dependent transport duration between stops of the line, but no exact schedules. This model covers also the simpler cases where the line has exactly two stops. Bike sharing for instance could be modeled using non-scheduled lines, for example by creating one line per pair of stations that are reachable from each other. In that case, the bike section is considered as a trip from the algorithm point of view, and using bike sharing increases the number of transfers between trips of the itinerary. Similarly, taxi-like offers that cover some predefined sets of origins and destinations can be modeled as non-scheduled lines. In both cases, we then consider that using those modes is equivalent to taking one additional trip in terms of inconvenience (which is modeled by the number of transfers of an itinerary).

Many public transit routing and multimodal trip planning algorithms have been proposed recently in the literature [5], but to the best of our knowledge, general non-scheduled lines have not been considered explicitly.

In Section 2, we discuss recent algorithms for public transit and multimodal routing and models for the simpler cases that appear in the literature. In Section 3 we introduce the notation and briefly present the Trip-Based Public Transit Routing algorithm [30] that we extend in Section 4 for supporting non-scheduled lines. In Section 5, we present the results of experiments on two real world datasets (Île-De-France and Netherlands), we summarize our work and give directions for future research in Section 6.

2 Related work

In this article, we are mainly interested in two classical criteria to minimize in multimodal routing, which are the number of transfers and the arrival time. The number of transfers represents the inconvenience for the user to change vehicles and is an important criterion for evaluating itineraries. Given a start time, computing the Pareto set for those two criteria is intractable as the size of the Pareto set can be exponential [20]. However, the Pareto front is of polynomial size (bounded by the number of trips) and can be computed in polynomial time, along with one solution per value in the Pareto front (note that this is often referred to computing the Pareto set in the literature, while only a subset of the Pareto set is indeed obtained). We use here the notation of [26] and denote by *complete set* such a solution set. Most recent algorithms, considering either minimum arrival time alone or bicriteria queries, can compute earliest arrival time or Pareto front in time ranging from tens of microseconds to a few hundreds of milliseconds for large public transit networks. Transfer Patterns [4], RAPTOR [13, 10], Connection Scan (CSA) [14], Public Transit Labeling [12] or Trip-Based Public Transit Routing [30] have been specifically designed for those networks as using directly classical methods for road networks does not seem to perform well with the time-dependent schedules [3].

Although, to the best of our knowledge, combination of scheduled and non-scheduled lines in public transit networks has not been studied before, some algorithms can handle more transportation modes in combination to public transit, including bike or car sharing.

In the graph-based approaches, the different networks corresponding to each mode are combined into a single time-dependent or time-expanded graph. The timetable information and transfer constraints (for instance minimum change times at a station) are modeled into the graph structure, often increasing significantly the graph size [25]. In that type of approach, non-scheduled lines can be modeled into the graph as additional arcs and nodes available for some time intervals. In order to take into account the different modes with graph-based modeling, one possible solution is to define an automaton that restricts the possible mode sequences and solve a *label-constrained shortest path problem*. The label-constrained shortest path problem is tractable for regular languages [2] and some authors proposed algorithms allowing for mode sequences using or not public transit. Kirchler et al. [21] propose to adapt the ALT algorithm [18] to take into account predefined mode sequences, resulting in the SDALT algorithm (State-Dependent ALT). They consider a network that includes bike and car sharing. Dibbelt et al. [15] modify Contraction Hierarchy [17] to integrate user defined sequences provided at query time. The resulting algorithm is called User Constrained Contraction Hierarchy. They apply it on networks combining cars and public transportation, but the approach could be applied for bike or car sharing. One of the drawbacks of this algorithm is the preprocessing time (42 minutes on a network with 30.5K stops and 1.6M connections) that doesn't allow for real-time modification of the schedules.

The second main type of approaches consists in using timetable directly without modeling it into a graph. The RAPTOR algorithm [13] is one of these algorithms, using dynamic programming to perform a breadth-first search that labels the stops reached, one additional trip being taken at each iteration of the algorithm. It has been modified in [11, 28] to allow for some more complex mode sequences, such as combining public transit with bike or car sharing, by modeling it similarly as a walking transfer or with a biking part equivalent to a trip. Shortest travel times between stops using bike or car sharing are then precomputed and integrated as alternative ways to change from one line to another. A recent approach [27] combines RAPTOR with ULTRA [7] to reduce the running time and to consider several bike sharing operators simultaneously.

In this article, we are interested in the general case, where the sequence of non-scheduled lines contains more than two elements. However, as bike and car sharing are special cases of non-scheduled lines with two-stop sequences, our algorithm could be used to interleave them with public transportation, even if it is not the main objective here. The proposed method integrates non-scheduled lines in the Trip-Based Public Transit Routing algorithm that we present in Section 3.

3 Preliminaries

We introduce in this section the notation used in the paper, and explain the principle of the Trip-Based Public Transit Routing (TB) algorithm [30].

3.1 Notation

Public transit networks are defined by their stops and trip schedules. A *stop* p is a physical location where passengers can board or alight a public transportation vehicle (e.g. a bus, a tram, a metro). A *trip* t is represented by its *schedule*: a sequence of stops $\overrightarrow{p}(t) = (p_t^1, p_t^2, \dots)$ where the vehicle stops, with arrival time $\tau_{\text{arr}}(t,i)$ and departure time $\tau_{\text{dep}}(t,i)$ at its i^{th} stop p_t^i. A partial order is defined over trips with the same stop sequence $(p^1, p^2, \dots, p^n)$ by the relations $\leq$ and $<$:

$$t \leq t' \Leftrightarrow \forall i \in \{1,2,\dots,n\}, \tau_{\text{arr}}(t,i) \leq \tau_{\text{arr}}(t',i)$$

$$t < t' \Leftrightarrow (t \leq t' \text{ and } \exists i \in \{1,2,\dots,n\}, \tau_{\text{arr}}(t,i) < \tau_{\text{arr}}(t',i))$$

A scheduled line l is then a totally ordered set of trips with the same stop sequence $\overrightarrow{p}(l)$. Note that if there are two trips with the same stop sequence such that one is overtaking the other, they are associated to different lines. L is the set of all scheduled lines, p_l^i represents the i^{th} stop of line l and l_t denotes the line of trip t. For each stop p, we define the set of the lines passing by p with their corresponding stop index at p by $L(p) = \{(l, i) \mid l \in L,\, i \in \{1, 2, \dots |\overrightarrow{p}(l)|\},\, p = p_l^i\}$.

When arriving at stop p_l^i at time τ, it is possible to board a trip t of line l if $\tau \leq \tau_{\mathrm{dep}}(t, i)$. When it exists, we can hence define the earliest trip of line l departing from its i^{th} stop after time τ, that we denote by $\mathrm{earliest}(l, i, \tau)$.

The segment of the trip t between stops of index i and j is denoted by $p_t^i \to p_t^j$ and similarly, a transfer between the i^{th} stop of trip t and the j^{th} stop of trip t' is denoted by $p_t^i \to p_{t'}^j$.

Minimum walking transfer duration (footpath) between stops p and q is denoted by $\Delta\tau_{\mathrm{fp}}(p,\, q)$ and minimum changing vehicle duration at stop p by $\Delta\tau_{\mathrm{ch}}(p)$ (for example, the duration for changing platforms at the same stop). Transfer $p_t^i \to p_{t'}^j$ is *feasible* if and only if:

$$\begin{array}{rrcll} & \tau_{\mathrm{arr}}(t,i) + \Delta\tau_{\mathrm{fp}}(p_t^i,\, p_{t'}^j) & \leq & \tau_{\mathrm{dep}}(t',j), & \text{if } p_t^i \neq p_{t'}^j \\ \text{or} & \tau_{\mathrm{arr}}(t,i) + \Delta\tau_{\mathrm{ch}}(p_t^i) & \leq & \tau_{\mathrm{dep}}(t',j), & \text{if } p_t^i = p_{t'}^j \end{array}$$

Lines without a schedule. Now, we extend the above defined notation to lines without a schedule, whose set is denoted $\widehat{L}$. A non-scheduled line l has a sequence $\overrightarrow{p}(l) = (p_l^1,\, p_l^2,\, \dots)$ of stops, but no fixed timetable, as they should be activated at the user's demand. Trips for those lines can be instantiated during given time intervals when the service is available and we can define as before the set of all lines without schedule passing by p. We denote it by $\widehat{L}(p)$.

For easy computation of trip $\mathrm{earliest}(l, i, \tau)$, we defined one union of availability intervals for each stop of the non-scheduled line l, and we denote it by $I(l,\, i)$ for the i^{th} stop of l. A possible way to define those time intervals is to define them for the first stop and then translate them to the other stops of the line by adding traveling duration between stops. This could be the case for on-demand buses if the bus passes by all the stops when activated. Another possibility is to use the same time interval for all stops. It can be the case for non-scheduled lines defined for bike sharing stations or for taxi-like transportation between two points where the time-intervals represent the service availability period.

An easy way of including non-scheduled lines in existing trip planning algorithms is to discretize the intervals of $I(l,\, 1)$ and generate all possible trips (e.g. creating one trip every minute). In a context of urban mobility, the intervals can be wide (typically from 7.00 am to 6.00 pm) so this approach can significantly increase the number of trips and the number of possible transfers. For the TB algorithm, it has a significant impact on preprocessing and query times. This approach is used in our experiments as a baseline method.

In some cases, a boarding or alighting duration might be considered for the lines of $\widehat{L}$. For instance, it can model the time needed to buy a bus ticket or to get off with some luggage. For bike sharing rides, the boarding time could be the duration needed to get the bicycle from the station and the alighting time the duration to put it back in place. We denote by $\tau_{\mathrm{bo}}(l)$ the duration necessary for boarding the line and $\tau_{\mathrm{al}}(l)$ the duration necessary for alighting. To remain general, we consider boarding and alighting times for all lines, as we can just set them to 0 when they are not relevant.

3.2 Trip-Based Public Transit Routing

Trip-Based Public Transit Routing [30] is an algorithm for computing a complete solution set for minimum arrival time and number of transfers in public transit networks, considering an origin, a destination and a start time. The author claims to consider maximum departure time as a secondary criterion used to break ties, but it is proven in [22] that there is no guarantee regarding this last criterion.

The TB algorithm is based on the preprocessing of a set of the possible transfers between trips. The aim is to build for each trip, during a preprocessing phase, a neighborhood of reachable trips in such way that for each value in the Pareto front, there exists an optimal path with this value using only elements of the resulting neighborhoods. A bicriteria earliest arrival time query then consists in a breadth-first search like exploration in a time-independent graph where trips are vertices and transfers are arcs. Figure 1 gives an outline of the algorithm.

The method proposed in [30] also covers profile queries, where all the optimal values must be found for a given starting time range, hence effectively optimizing latest departure time as a third criterion.

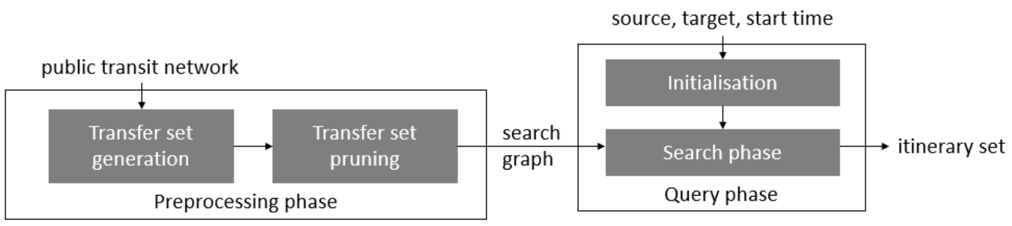

Figure 1 Phases of the TB algorithm.

Preprocessing. The transfer set size impacts the exploration time. As many transfers cannot appear in any optimal solution, it is advisable to prune the transfer set. For instance, if you consider the possible transfers between one trip and a different line, only the earliest trip that can be boarded is relevant for the above defined queries. The author hence suggests two pruning methods to reduce the set of possible transfers.

The first removes U-turn transfers for each trip, i.e. transfers that take you back to the previous stop in the trip (later than if you changed trip at the previous stop). The second aims at pruning the set of feasible transfers for each trip based on earliest arrival times at stops. Each transfer is considered, starting with the later ones. If taking later transfers (or remaining on the current trip) leads to identical or better arrival times or if all the trips reachable via the transfer can be reached via those later transfers, then the current transfer is removed from the set, as it cannot lead to other optimal values than the transfers already kept. Note that the transfer set obtained is not minimal in terms of number of transfers, and that it depends on the order of the transfers checked.

Earliest arrival time queries. In the context of the TB algorithm, earliest arrival time queries refer to bicriteria queries where a single departure time τ is provided as input, along with a source stop, denoted by p_{src}, and a target stop p_{tgt}. Minimum arrival time and the minimum number of transfers are optimized. Note that even if this case is not considered in [30], it is not necessary for the origin and destination of the queries to be stops. If they are placed anywhere on the road network, the algorithm is hardly modified but the footpaths to reach the closest stops in the network must be computed, for instance by classical shortest paths in the walking road network.

Earliest arrival time queries start with an initialization phase where the queue of the search phase is initialized and its target set is computed from the source and target stops. The target set is the set of lines $\mathcal{L}$ from which the destination can be reached:

$$\begin{aligned}\mathcal{L} \;=\; & \{(l,\, i,\, 0) \mid (l,\, i) \in L(p_{\text{tgt}})\} \;\; \cup \\ & \{(l,\, i,\, \Delta\tau_{\text{fp}}(q,\, p_{\text{tgt}})) \mid (l,\, i) \in L(q) \text{ and } q \text{ is a neighbor of } p_{\text{tgt}}\}\end{aligned}$$

In the query phase, the author labels the trips by the index $R(t)$ of the first reached stop of t, initialized to ∞ for all trips. He defines for each number of transfers one queue Q_n of trip segments reached after n transfers. Q_0 is initialized from the lines that can be taken from the source stop. For any stop q reached by walking from p_{src}, the earliest trip of $(l, i) \in L(q)$ is added to the queue, starting at index i. It is the smallest trip t of line l such that

$$\tau_{\text{dep}}(t,\, i) \geq \begin{cases} \tau & \text{if } q = p_{\text{src}} \\ \tau + \Delta\tau_{\text{fp}}(p_{\text{src}},\, q) & \text{otherwise} \end{cases}$$

After initialization, a breadth-first like search is performed. At each iteration, the algorithm scans in turn the trip segments of the queue. If the current one belongs to a target line, the arrival time at destination is compared to that of the solution set. Then, the trip segments reached by transferring from the current trip segment are added to the queue of the next iteration if they improve the trips' labels. It is the case for a trip segment (t, i, k) if t is earlier than any trip of t_l taken so far at stop i. When a trip t is marked with $R(t) = i$, all the later trips of l_t are marked with the minimum of i and their current index.

Profile queries. In *profile queries*, the user provides an earliest departure time τ_{edt} and a latest departure time τ_{ldt}, i.e. an interval in which to depart. The result of the query is a complete set of solutions for minimum arrival time, minimum number of transfers and maximum departure time starting within the interval. The computation for profile queries is as follows: perform an earliest arrival time query starting at τ_{ldt} and add the solutions to the result set of the profile search. Then restart the search starting at the preceding instant without resetting the trip labels. By iterating the process, you obtain a complete set of solutions without performing unnecessary computations as the labels only let you improve on preceding arrival times.

4 Trip-Based algorithm with non-scheduled lines

As we mentioned above, to the best of our knowledge, lines without a schedule are not covered in the literature, although the special case of bike sharing appears in several articles (e.g. [11, 21, 28]). We explain here our method for the general case where the non-scheduled lines can have more than two stops in their sequence.

4.1 Defining a trip for a non-scheduled line

We consider that non-scheduled lines have predefined stop sequences and availability intervals for each stop of the line. In order to define the earliest trip segment of a trip t of a non-scheduled line l starting after a time τ, we need to evaluate the duration of this trip segment. One possibility is to use the same principle as in the General Transit Feed Specification (GTFS) format [19] for frequency-based trips: one trip with a complete schedule is defined and the others are translations of it with different start times. A more complex solution could consider time-dependent travel times between the line's consecutive stops and time-dependent arrival and departure times of the trips at its stops. In that case, one must keep in mind that trips of a line cannot overtake one another and that those time-dependent travel times need to respect the FIFO property.

The algorithm proposed here is independent of the solution chosen as long as we can define a schedule for earliest(l, i, τ). Note that the trip segment's departure time is either time τ if $\tau \in I(l, i)$ or the earliest instant of $I(l, i)$ after τ. If such time doesn't exist in the current day, then we can consider taking the line the day after at $\min_{\theta \in I(l,i)} \theta$, if the service is available. If the intervals of $I(l, i)$ are sorted in increasing start time order, τ can be computed in logarithmic time of the number of intervals using binary search.

4.2 Transfers to and from a line without schedule

In the TB algorithm, the transfer generation phase starts by computing all the possible transfers for each trip. For each stop p_t^i of the current trip t, find all the stops q that can be reached by footpaths (i.e. $\Delta\tau_{\text{fp}}(p_t^i, q)$ is defined), and check if a transfer can take place for each element (l, j) of $L(q)$. Stop q is reached at time $\theta = \tau_{\text{arr}}(t, i) + \Delta\tau_{\text{fp}}(p_t^i, q)$ (or $\theta = \tau_{\text{arr}}(t, i) + \Delta\tau_{\text{ch}}(q)$ if $q = p_t^i$) and we can enforce a minimum boarding time to get the minimum time $\tau = \theta + \Delta\tau_{\text{bo}}(l)$ at which a trip of line l can be taken. If it is defined, only transfer to the earliest trip of each line passing after time τ is added to the neighborhood of t. We can proceed identically for admissible transfers from trips of scheduled lines to non-scheduled lines. The earliest trip passing at q after τ is defined as in Section 4.1 and we keep only the transfer to that trip.

Initially, the trips of non-scheduled lines are not instantiated: they are implicit within the non-scheduled line definition. It is however possible to precompute some trip segments and transfers to make the search faster. We extend the set of transfers to add the transfers from a trip of a scheduled line to non-scheduled lines. We then prune the resulting extended set of transfers as before. We denote with $\widehat{T}$ the set of transfers from a trip segment of a scheduled line to a trip segment of a non-scheduled line. $T \cup \widehat{T}$ is the extended set of transfers.

Note that for non-scheduled lines, we do not perform a preprocessing of the transfers to scheduled and non-scheduled lines: instead, the transfers from trip segments of non-scheduled lines are computed online during the query phase. It avoids explicitly creating all the non-scheduled trips.

4.3 Modifications in the query phase

The algorithm for the query phase and its initialization can be found in Algorithm 1. The auxiliary procedures of both are described in Algorithm 2.

In the initialization, the lines without schedule are scanned similarly to regular lines for determining the algorithm's targets. To build the initial queue, we consider the availability intervals of non-scheduled lines at the stops reached from the origin and the minimum boarding times to propose the earliest trip segment for reached lines.

A major difference with the initial version of the algorithm is the change in determining the next trip segments to add to the queue. For transfers from scheduled lines, the set T of transfers contains all the preprocessed transfers. For transfers from non-scheduled lines, the transfers are computed on the fly. For transfers to scheduled lines, the next trip to take is computed as in the initial algorithm and the trip segments added to the queue by the procedure ENQUEUE_TRIP. For transfers to non-scheduled lines, the more complicated process of ENQUEUE_LINE and UPDATE_R is required to avoid unnecessary computations.

Algorithm 1 Earliest arrival time query.

```
input Timetable data, transfer set $T \cup \widehat{T}$
input Source stop $p_{\text{src}}$, destination stop $p_{\text{tgt}}$, start time $\tau$
output Result set $J$
$J \leftarrow \emptyset$, $\mathcal{L} \leftarrow \emptyset$
$Q_n \leftarrow \emptyset$ for $n = 0, 1, \ldots$
$R(t) \leftarrow \infty$ for each trip $t$
$\widehat{R}(l, j) \leftarrow \infty$ for each line $l$ without schedule and each index $j = 0, 1, \ldots, |\overrightarrow{p}(l)|$
INITIALIZATION()
$\tau_{min} \leftarrow \infty$                                  ▷ The current minimum arrival time at target
$n \leftarrow 0$
while $Q_n \neq \emptyset$ do
    for each $p_t^b \to p_t^e \in Q_n$ do
        for each $(l_t, i, \Delta\tau) \in \mathcal{L}$ with $b < i$ and $\tau_{\text{arr}}(t, i) + \Delta\tau < \tau_{min}$ do
            $\tau_{min} \leftarrow \tau_{\text{arr}}(t, i) + \Delta\tau$        ▷ A target is reached and arrival time is improved
            $J \leftarrow J \cup \{(\tau_{min}, n)\}$, removing dominated entries

        if $\tau_{\text{arr}}(t, b+1) + \Delta\tau_{\text{al}}(l_t) < \tau_{min}$ then      ▷ Filling the queue for the next round
            if $l_t \in \widehat{L}$ then                                ▷ Transfers must be computed
                for each stop $p_t^i$ with $b < i \leq e$ do
                    for each stop $q$ such that $\Delta\tau_{\text{fp}}(p_t^i, q)$ is defined do
                        $\tau \leftarrow \Delta\tau_{\text{fp}}(p_t^i, q) + \tau_{\text{arr}}(t, i) + \Delta\tau_{\text{al}}(l_t)$
                        for each $(l, k) \in L(q)$ do
                            $t' \leftarrow \text{earliest}(l, k, \tau + \Delta\tau_{\text{bo}}(l))$
                            ENQUEUE_TRIP($t'$, $k$, $n+1$)
                        for each $(l, k) \in \widehat{L}(q)$ do
                            ENQUEUE_LINE($l$, $k$, $\tau + \Delta\tau_{\text{bo}}(l), n+1$)
            else
                for each transfer $p_t^i \to p_u^j \in T$ with $b < i \leq e$ do
                    ENQUEUE_TRIP($u$, $j$, $n+1$)
                for each $(p_t^i \to p_l^j, \tau) \in \widehat{T}$ do
                    ENQUEUE_LINE($l$, $j$, $\tau$, $n+1$)
    $n \leftarrow n + 1$

procedure INITIALIZATION
    for each stop $q$ s.t. $\Delta\tau_{\text{fp}}(q, p_{\text{tgt}})$ is defined do      ▷ Initialization of the target lines
        $\Delta\tau \leftarrow 0$ if $p_{\text{tgt}} = q$, else $\Delta\tau_{\text{fp}}(q, p_{\text{tgt}})$
        for each $(l, i) \in L(q) \cup \widehat{L}(q)$ do
            $\mathcal{L} \leftarrow \mathcal{L} \cup (l, i, \Delta\tau + \Delta\tau_{\text{al}}(l))$
    for each stop $q$ s.t. $\Delta\tau_{\text{fp}}(p_{\text{src}}, q)$ is defined do                ▷ Initialization of $Q_0$
        $\Delta\tau \leftarrow 0$ if $p_{\text{src}} = q$, else $\Delta\tau_{\text{fp}}(p_{\text{src}}, q)$
        for each $(l, i) \in L(q)$ do
            $t \leftarrow \text{earliest}(l, i, \tau + \Delta\tau_{\text{al}}(l))$
            ENQUEUE_TRIP($t$, $i$, 0)
        for each $(l, i) \in \widehat{L}(q)$ do
            ENQUEUE_LINE($l$, $i$, $\tau + \Delta\tau_{\text{al}}(l), 0$)
```

This process is as follows. For a non-scheduled line l, label $\widehat{R}(l)$ contains a set of pairs with the index of a stop and the earliest departure time at that stop. This set is such that an element (i, τ) of $\widehat{R}(l)$ is not dominated by any other element of $\widehat{R}(l)$. A pair (i, τ) is dominated by a pair (j, τ') if and only if

$$i \geq j \text{ and } \tau > \tau_{\text{dep}}(\text{earliest}(l,\, j,\, \tau'), i)$$

Indeed, if $i \geq j$, the trip is boarded later at i, missing some transfer opportunities compared to boarding it at j. And if $\tau > \tau_{\text{dep}}(\text{earliest}(l,\, j,\, \tau'), i)$, then the earliest trip that can be boarded at j after τ' brings you at the i^{th} stop earlier than τ. Hence, the set $\widehat{R}(l)$ contains at most $|\overrightarrow{p}(l)|$ elements and we can check the dominance of a new pair over the elements of the set in polynomial time. To maintain the elements of $\widehat{R}(l)$, one can save for each stop i of l the earliest departure time of a trip of l at that stop during the search. In that case, $\widehat{R}(l,\, i)$ represents the earliest departure time of l at its i^{th} stop in the current search. Another possibility is to sort the pairs of the set $\widehat{R}(l)$ by increasing stop index and to use the fact that the times are sorted in decreasing order to accelerate the dominance checks while needing less memory. Procedure UPDATE_R describes the update process of $\widehat{R}$ and computes the maximum index k for which $\widehat{R}(l, k)$ is modified, so as to determine the last element of the trip segment to add to the queue in the procedure ENQUEUE_LINE if $\widehat{R}$ is modified.

Note that since profile queries are an adaptation of earliest arrival time queries, it is possible to take them into account as proposed in [30] even after the modifications.

Algorithm 2 Earliest arrival query auxiliary procedures.

```
procedure ENQUEUE_TRIP(trip $t$, index $i$, number of transfers $n$)
    if $i < R(t)$ then                              ▷ Adding the given trip segment to the queue
        $Q_n \leftarrow Q_n \cup \{p_t^i \rightarrow p_t^{R(t)}\}$
        for each trip $u$ with $t \leq u$ and $l_t = l_u$ do
            $R(u) \leftarrow \min(R(u),\, i)$

procedure ENQUEUE_LINE(line $l \in \widehat{L}$, index $i$, time $\tau$, number of transfers $n$)
    $ind,\, t \leftarrow$ UPDATE_R($l,\, i,\, \tau$)          ▷ Updating non-scheduled line labels
    if $ind \geq i$ then                        ▷ Adding the earliest trip segment to the queue
        $Q_n \leftarrow Q_n \cup \{p_t^i \rightarrow p_t^{ind}\}$

procedure UPDATE_R(line $l \in \widehat{L}$, index $i$, time $\tau$)
    output Maximum index $j$ s.t. $\widehat{R}(l, j)$ is modified, $i - 1$ if no modification
    $updated \leftarrow i$
    $t \leftarrow \text{earliest}(l,\, i,\, \tau)$
    while $updated \leq |\overrightarrow{p}(l)|$ and $\tau_{\text{dep}}(t,\, updated) < \widehat{R}(l, updated)$ do
        $\widehat{R}(l, updated) \leftarrow \tau_{\text{dep}}(t,\, updated)$
        $updated \leftarrow updated + 1$
                                    ▷ Trip is scanned to its end or stop is reached by an earlier trip
    return $updated - 1,\, t$
```

4.4 Complexity and correctness

Complexity. In [30], the complexity of the algorithm is not indicated. However, it can be shown that the algorithm performs a number of operations polynomial in the input size. We discuss here the worst case complexity for the non-scheduled line extension that we propose. An important difference is that only part of the instance is represented in the search graph.

The set of vertices V of the search graph contains the trips of the scheduled lines and the destination trips of the transfers of $\widehat{T}$. The number of elements in V is hence bounded by the number of trips of scheduled lines, denoted N_s, plus the size of $\widehat{T}$. Given an origin trip t, it would be possible to transfer from each stop of t (except the first one) to each stop (except the last one) of each non-scheduled line and to keep those transfers in $\widehat{T}$. We hence have $|\widehat{T}| = \mathcal{O}(N_s\,|\widehat{L}|\,|S|^2)$, if S is the set of stops, and $|V|$ is polynomial. Similarly, we can bound the number of elements of T by $|T| = \mathcal{O}(N_s^2\,|S|^2)$. The arcs A of the search graph represent the transfers of $T \cup \widehat{T}$. So $|A| = \mathcal{O}(N_s(N_s + |\widehat{L}|)|S|^2)$. Arcs from trips of non-scheduled lines are implicit. Computing the earliest trip of a line leaving a given stop after a time τ can be done efficiently using binary search in $\mathcal{O}(\log n)$ for scheduled lines, if n is the number of trips of the line and $\mathcal{O}(\log|S|)$ for non-scheduled lines. The set of earliest feasible transfers can hence be obtained in $\mathcal{O}(N_s(\log(N_s)N_s + \log(|S|)|\widehat{L}|)|S|^2)$. The pruning phase is also polynomial, as for each transfer, at most all the stops' labels must be updated.

For the initialization of the query phase, at most all the stops can be reached from p_{src} and all the lines taken, which takes $O(|S|\,|L \cup \widehat{L}|)$ 'ENQUEUE' operations. Then, at each iteration, we loop over the queue's content. For each trip segment in the queue, we first iterate over the targets (those number is bounded by $|S|\,|L \cup \widehat{L}|$). Then if it is from a scheduled line, we scan its outgoing arcs. At most, $|T \cup \widehat{T}|$ arcs are processed and elements are added to the queue. Otherwise, the worst case corresponds to the existence of a transfer from each stop of the non-scheduled line to each stop of any other line. It is hence bounded by $\mathcal{O}(|L \cup \widehat{L}|\,|S|^2)$. It results in a polynomial number of 'ENQUEUE' operations.

The ENQUEUE_TRIP procedure updates in the worst case the labels of all the trips of the line of its input trip t. It hence has complexity $\mathcal{O}(N_s)$. ENQUEUE_LINE updates at most $|\overrightarrow{p}(l)|$ labels of the set of labels of its input line l. It is hence bounded by $|S|$.

Overall, each step of the search phase is hence polynomial in the instance size.

To bound the number of iterations, first note that it is not possible to take twice the same trip in an optimal solution. A solution that alights a trip to board it again has at least one more transfer than the solution remaining on the current trip. It hence cannot be built by the algorithm and, for the original algorithm, the number of iterations is bounded by N_s. Taking twice the same line would be possible, for instance if the line is passing twice by the same stop, but not taking an earlier trip at a stop already passed by a preceding trip of the same line. The number of non-scheduled line trip segments in a solution built by the algorithm is hence bounded by $\mathcal{O}(|S|)$. Hence, the number of iterations is in $\mathcal{O}(N_s + |S|\,|\widehat{L}|)$.

Correctness. To prove that the algorithm is correct, we need to show that for any optimal solution in the Pareto set, there exists an optimal solution with the same value whose transfers are either in the pruned transfer set $T \cup \widehat{T}$ or are transfers from non-scheduled lines. Let s be an optimal solution with at least one transfer described by the trip segments that compose it: $s = \left\langle p_{t_1}^{j_1} \to p_{t_1}^{i_1}, p_{t_2}^{j_2} \to p_{t_2}^{i_2}, \ldots, p_{t_{K+1}}^{j_{K+1}} \to p_{t_{K+1}}^{i_{K+1}} \right\rangle$. If all its transfers are either in the transfer set $T \cup \widehat{T}$ or are transfers from non-scheduled lines, we are done. If not, from this solution, we build another optimal solution s' whose transfers are either in $T \cup \widehat{T}$ or are transfers from non-scheduled lines. First, iterating from $p_{t_2}^{j_2} \to p_{t_2}^{i_2}$, we replace the trip segments $p_{t_k}^{j_k} \to p_{t_k}^{i_k}$ that are not the earliest for which the transfer to l_{t_k} is possible from $p_{t_{k-1}}^{i_{k-1}}$. Since s is optimal, it is not possible to arrive sooner at stop $p_{t_{K+1}}^{i_{K+1}}$ and trip segment $p_{t_{K+1}}^{j_{K+1}} \to p_{t_{K+1}}^{i_{K+1}}$ is not modified. To simplify, we keep the same notation for the modified trip segments of s if any.

Table 1 Netherlands and IDF datasets.

Netherlands

# stops	# lines	# foot paths
47313	2773	429.4K

	# trips	# connections
Baseline	364.2K	6.527M
Non-sch.	317.7K	5.938M

IDF

# stops	# lines	# foot paths
42302	1869	752K

	# trips	# connections
Baseline	373.3K	7.867M
Non-sch.	318.0K	7.014M

Consider the last transfer $p_{t_K}^{i_K} \rightarrow p_{t_{K+1}}^{j_{K+1}}$ of s. If trip t_K is from a non-scheduled line, we keep it in s'. Otherwise, suppose that this transfer is not in $T \cup \widehat{T}$. In that case, there exists a transfer $p_{t_K}^{i'_K} \rightarrow p_{t'_{K+1}}^{j'_{K+1}}$ in $T \cup \widehat{T}$ from t_K such that $i'_K \geq i_k$ and t'_{K+1} arrives at $p_{t_{K+1}}^{i_{K+1}}$ at time $\tau_{\text{arr}}(t_{K+1}, i_{K+1})$ or before by definition of the pruning phase. Note that as the solution is optimal, it is exactly at time $\tau_{\text{arr}}(t_{K+1}, i_{K+1})$. If we denote with i'_{K+1} the smallest index in the stop sequence of t'_{K+1} such that $i'_{K+1} \geq j'_{K+1}$ and $p_{t_{K+1}}^{i_{K+1}} = p_{t'_{K+1}}^{i'_{K+1}}$, we can hence replace the two last trip segments by $p_{t_K}^{j_K} \rightarrow p_{t_K}^{i'_K}, p_{t'_{K+1}}^{j'_{K+1}} \rightarrow p_{t'_{K+1}}^{i'_{K+1}}$ in s'.

Proceeding likewise for the other transfers going backward in the solution, we can obtain a solution s' with the same value as s those transfers are all in $T \cup \widehat{T}$ or are transfers from a non-scheduled line.

5 Experiments

To the best of our knowledge, there is no open transit dataset with non-scheduled lines. One of the reason is that the most widespread data format, the GTFS format [19], does not provide specifications for defining non-scheduled lines. A recent proposal [23] extends it to some on-demand transports [29], but it doesn't cover the general case of non-scheduled lines, where the stop sequences of the lines are defined. Due to lack of standards, service providers usually develop their own methods for specification and integration of non-scheduled lines in their trip planners if they wish to propose them.

For our experiments, we modified public datasets for Netherlands [8] and Île-De-France [9] (IDF). The Netherlands dataset contains on-demand lines, but with predefined schedules, which require phone activation. From the perspective of the TB algorithm, this type of lines are handled as standard lines as they have predefined schedules and are not appropriate for our need. We hence slightly change the original dataset by converting 253 on-demand lines with predefined schedules to lines without schedule. For the IDF dataset, we obtain 201 non-scheduled lines. For each line, we set the availability period to 7.30 am to 7 pm for the first stop and translate the interval for each later stop of the line according to a fixed travel time between the origin stop and that stop. We denote by *Non-sch.* those datasets and we use the proposed algorithm to compute itineraries in those networks.

We also generate another variation of those datasets: instead of non-scheduled lines, we instantiated all the possible trips for the non-scheduled lines by generating one trip every two minutes in the interval. Those datasets are our baseline, as they allow to take into account non-scheduled lines without modification of the base algorithm.

Table 1 summarizes the datasets. Non sch. and baseline have the same number of stops, lines and foot paths, but the number of trips and connections (transfer between two consecutive stops taking a trip) differ. The experiments are run on a 2.7 GHz CPU Intel(R) Xeon(R) CPU E5-4650 server with 64 cores, 20M of L3 cache and 504 GB of RAM by using a solver developed in the Rust programming language.

Table 2 Preprocessing phase.

	Netherlands			IDF		
	Time (s)	# Transfers to scheduled lines	# Transfers to non-sch. lines	Time (s)	# Transfers to scheduled lines	# Transfers to non-sch. lines
Baseline	49	62.087M	0	75	90.183M	0
Non-sch.	46	60.306M	0.121M	73	84.212M	1.191M

Table 3 Query times for Baseline and Non-sch.

		Netherlands			IDF		
	Query	Mean time (ms)	Min time (ms)	Max time (ms)	Mean time (ms)	Min time (ms)	Max time (ms)
Baseline	Earliest arrival	60	23	173	56	21	153
	Profile 8.30	147	47	319	118	42	199
	Profile 14.30	134	44	244	131	41	238
Non-sch	Earliest arrival	45	18	147	43	8	106
	Profile 8.30	122	33	289	103	15	170
	Profile 14.30	99	25	268	102	13	183

Preprocessing times of the two settings are similar, although the setting using non-scheduled lines is slightly faster, as it has less trips to process (see Table 2). Remark that the preprocessing time is low enough to allow for real-time updates of the network every couple of minutes. It implies that in the case when a non-scheduled trip is booked, it is possible to update the network to include it as a scheduled trip. The availability intervals of some stops-line pairs can also be modified to take into account the fact that this booked vehicle is no longer available.

To compare query times, we selected randomly 300 origin and destination pairs over each network. We generated 3 queries per origin-destination pair: an earliest arrival time query and two profile queries. For each query, a complete set of solutions is computed. We fixed the departure times of the earliest arrival time queries at 8.30 (am), a time at which trips are usually more frequent (which makes the exploration more time consuming) and for profile queries, we considered time intervals of length one hour, starting at 8.30 and 14.30 respectively. Profile queries starting at 14.30, a time where trip frequencies are less high, result in fewer solutions and are hence expected to run faster than the ones starting at 8.30. Results of the experiments can be found in Table 3.

For our experiments, we turned about 10% of the lines into non-scheduled lines, and hence cannot expect a huge difference in query times. However, the difference is significant enough for the method to be interesting from a performance point of view, as mean query times are between 13% and 27% faster than that of the baseline version (see Table 4).

Table 4 Non-sch. query times divided by baseline query times.

	Netherlands			IDF		
Query	Mean time	Min time	Max time	Mean time	Min time	Max time
Earliest arrival	0.75	0.78	0.85	0.77	0.38	0.74
Profile 8.30	0.83	0.70	0.91	0.87	0.36	0.85
Profile 14.30	0.73	0.57	1.1	0.78	0.32	0.77

6 Conclusion and future work

In this article, we proposed a method for computing itineraries in public transit or multimodal networks with scheduled and non-scheduled lines. It extends the Trip-Based Public Transit Routing algorithm to on-demand lines with a predefined stop sequence and availability intervals but no associated schedules. Experimental results over two large datasets show that the proposed approach performs better than the baseline consisting in discretizing the availability interval to generate all the possible trips for the non-scheduled lines.

This model has car and bike sharing as a special case. A perspective of our work could hence be to test our method with multimodal networks including those modes, against classical modeling as a transfer, and not as a trip. Another line of work could be concerned with applying classical acceleration techniques, such as Transfer Patterns [4, 6], to the proposed algorithm. Transfer patterns have been adapted to Trip-Based Public Transit Routing in [31] and could be extended to take into account non-scheduled lines.

References

1 Transports montalbanais. Access date: 2021/03/29. URL: `https://www.montm.com/transport-a-la-demande-et-pmr/`.

2 Chris Barrett, Riko Jacob, and Madhav Marathe. Formal-language-constrained path problems. *SIAM J. Comput.*, 30(3):809–837, May 2000. `doi:10.1137/S0097539798337716`.

3 Hannah Bast. Car or Public Transport – Two Worlds. In Susanne Albers, Helmut Alt, and Stefan Näher, editors, *Efficient Algorithms: Essays Dedicated to Kurt Mehlhorn on the Occasion of His 60th Birthday*, pages 355–367. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009. `doi:10.1007/978-3-642-03456-5_24`.

4 Hannah Bast, Erik Carlsson, Arno Eigenwillig, Robert Geisberger, Chris Harrelson, Veselin Raychev, and Fabien Viger. Fast routing in very large public transportation networks using transfer patterns. In *Proceedings of the 18th Annual European Conference on Algorithms: Part I*, ESA'10, pages 290–301, Berlin, Heidelberg, 2010. Springer-Verlag.

5 Hannah Bast, Daniel Delling, Andrew Goldberg, Matthias Müller-Hannemann, Thomas Pajor, Peter Sanders, Dorothea Wagner, and Renato F. Werneck. Route Planning in Transportation Networks. In *Kliemann L., Sanders P. (eds) Algorithm Engineering - Selected Results and Surveys*, volume 9220 of *Lecture Notes in Computer Science*, pages 19–80. Springer, Cham, 2016. `doi:10.1007/978-3-319-49487-6_2`.

6 Hannah Bast, Matthias Hertel, and Sabine Storandt. Scalable Transfer Patterns. In *2016 Proceedings of the Eighteenth Workshop on Algorithm Engineering and Experiments (ALENEX)*, pages 15–29, January 2016. `doi:10.1137/1.9781611974317.2`.

7 Moritz Baum, Valentin Buchhold, Jonas Sauer, Dorothea Wagner, and Tobias Zündorf. UnLimited TRAnsfers for Multi-Modal Route Planning: An Efficient Solution. In Michael A. Bender, Ola Svensson, and Grzegorz Herman, editors, *27th Annual European Symposium on Algorithms (ESA 2019)*, volume 144 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 14:1–14:16, Dagstuhl, Germany, 2019. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. `doi:10.4230/LIPIcs.ESA.2019.14`.

8 datahub. Timetables for transit in netherlands. Access date: 2019/07/29. URL: `https://old.datahub.io/dataset/gtfs-nl`.

9 Île de France Mobilités. Open data portal. Access date: 2020/05/04. URL: `https://data.iledefrance-mobilites.fr/pages/home/`.

10 Daniel Delling, Julian Dibbelt, and Thomas Pajor. Fast and Exact Public Transit Routing with Restricted Pareto Sets. In *Proceedings of the Twenty-First Workshop on Algorithm Engineering and Experiments (ALENEX)*, pages 54–65, San Diego, California, USA, 2019. Editor(s): Stephen Kobourov and Henning Meyerhenke. `doi:10.1137/1.9781611975499.5`.

11 Daniel Delling, Julian Dibbelt, Thomas Pajor, Dorothea Wagner, and Renato F. Werneck. Computing Multimodal Journeys in Practice. In *Experimental Algorithms - Proceedings of the 12th International Symposium, SEA 2013*, volume 7933 of *Lecture Notes in Computer Science*, pages 260–271. Springer Berlin Heidelberg, 2013. doi:10.1007/978-3-642-38527-8_24.

12 Daniel Delling, Julian Dibbelt, Thomas Pajor, and Renato F. Werneck. Public Transit Labeling. In Evripidis Bampis, editor, *Experimental Algorithms - Proceedings of the 14th International Symposium (SEA 2015)*, volume 9125 of *Lecture Notes in Computer Science*, pages 273–285. Springer International Publishing, 2015. doi:10.1007/978-3-319-20086-6_21.

13 Daniel Delling, Thomas Pajor, and Renato F. Werneck. Round-Based Public Transit Routing. In Society for Industrial and Applied Mathematics, editors, *Proceedings of the 14th Meeting on Algorithm Engineering and Experiments (ALENEX'12)*, pages 130–140, 2012. doi:10.1287/trsc.2014.0534.

14 Julian Dibbelt, Thomas Pajor, Ben Strasser, and Dorothea Wagner. Intriguingly Simple and Fast Transit Routing. In Vincenzo Bonifaci, Camil Demetrescu, and Alberto Marchetti-Spaccamela, editors, *Experimental Algorithms. International Symposium on Experimental Algorithms, SEA 2013*, volume 7933 of *Lecture Notes in Computer Science*, pages 43–54, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg. doi:10.1007/978-3-642-38527-8_6.

15 Julian Dibbelt, Thomas Pajor, and Dorothea Wagner. User-Constrained Multi-Modal Route Planning. In *Proceedings of the 14th Workshop on Algorithm Engineering and Experiments (ALENEX'12)*, pages 118–129. SIAM, 2012. Editors David A. Bader and Petra Mutzel. doi:10.1137/1.9781611972924.12.

16 Agglo du Pays de Dreux. Linéad. Access date: 2021/03/29. URL: https://www.linead.fr/8-Transport-a-la-demande.html.

17 Robert Geisberger, Peter Sanders, Dominik Schultes, and Daniel Delling. Contraction Hierarchies: Faster and Simpler Hierarchical Routing in Road Networks. In C.C. McGeoch, editor, *Experimental Algorithms. 7th Workshop on Experimental Algorithms (WEA 2008)*, volume 5038 of *Lecture Notes in Computer Science*, pages 319–333. Springer, Berlin, Heidelberg, 2008. doi:10.1007/978-3-540-68552-4_24.

18 Andrew Goldberg and Chris Harrelson. Computing the shortest path: A* search meets graph theory. In *Proceedings of the 16th Annual ACM–SIAM Symposium on Discrete Algorithms(SODA'05)*, pages 156–165. SIAM, 2005.

19 General transit feed specification. Access date: 2021/03/29. URL: https://gtfs.org/.

20 Pierre Hansen. Bicriterion Path Problems. In Günter Fandel and Tomas Gal, editors, *Multiple Criteria Decision Making Theory and Application*, volume 177 of *Lecture Notes in Economics and Mathematical Systems*, pages 109–127. Springer Berlin Heidelberg, 1980. doi:10.1007/978-3-642-48782-8_9.

21 Dominik Kirchler, Leo Liberti, and Roberto Wolfler Calvo. Efficient Computation of Shortest Paths in Time-Dependent Multi-Modal Networks. *ACM Journal of Experimental Algorithmics (JEA)*, 19, January 2015. doi:10.1145/2670126.

22 Vassilissa Lehoux and Darko Drakulic. Mode Personalization in Trip-Based Transit Routing. In Valentina Cacchiani and Alberto Marchetti-Spaccamela, editors, *19th Symposium on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2019)*, volume 75 of *OpenAccess Series in Informatics (OASIcs)*, pages 13:1–13:15, Dagstuhl, Germany, 2019. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/OASIcs.ATMOS.2019.13.

23 Ross MacDonald. Mobility on demand (mod) sandbox: Vermont agency of transportation (vtrans) flexible trip planner. Technical Report 0150, Federal Transit Administration (FTA) Research, 2020.

24 Transports publics de Flers Agglo. Némus. Access date: 2021/03/29. URL: https://nemus.flers-agglo.fr/se-deplacer/transport-a-la-demande.

25 Evangelia Pyrga, Frank Schulz, Dorothea Wagner, and Christos Zaroliagis. Efficient Models for Timetable Information in Public Transportation Systems. *ACM Journal of Experimental Algorithmics (JEA)*, 12(2.4):1–39, 2008. `doi:10.1145/1227161.1227166`.

26 Andrea Raith, Marie Schmidt, Anita Schöbel, and Lisa Thom. Extensions of labeling algorithms for multi-objective uncertain shortest path problems. *Networks*, 72(1):84–127, 2018. _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/net.21815. `doi:10.1002/net.21815`.

27 Jonas Sauer, Dorothea Wagner, and Tobias Zündorf. Faster Multi-Modal Route Planning With Bike Sharing Using ULTRA. In Simone Faro and Domenico Cantone, editors, *18th International Symposium on Experimental Algorithms (SEA 2020)*, volume 160 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 16:1–16:14, Dagstuhl, Germany, 2020. Schloss Dagstuhl–Leibniz-Zentrum für Informatik. `doi:10.4230/LIPIcs.SEA.2020.16`.

28 Luis Ulloa, Vassilissa Lehoux, and Frédéric Roulland. Trip Planning Within a Multimodal Urban Mobility. *IET Intelligent Transport Systems*, 12(2):87–92, 2018. `doi:10.1049/iet-its.2016.0265`.

29 GTFS-Flex v2. Flexible public transit services in gtfs. URL: `https://github.com/MobilityData/gtfs-flex/blob/master/spec/reference.md`.

30 Sascha Witt. Trip-Based Public Transit Routing. In Nikhil Bansal and Irene Finocchi, editors, *Algorithms - Proceedings of the 23rd Annual European Symposium on Algorithms (ESA'15)*, volume 9294 of *Lecture Notes in Computer Science*, pages 1025–1036, Berlin, Heidelberg, 2015. Springer Berlin Heidelberg. Editors: Nikhil Bansal and Irene Finocchi. `doi:10.1007/978-3-662-48350-3_85`.

31 Sascha Witt. Trip-Based Public Transit Routing Using Condensed Search Trees. In Marc Goerigk and Renato Werneck, editors, *Proceedings of the 16th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2016)*, volume 54 of *OpenAccess Series in Informatics (OASIcs)*, pages 10:1–12, Dagstuhl, Germany, 2016. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. Editors: Marc Goerigk and Renato Werneck. `doi:10.4230/OASIcs.ATMOS.2016.10`.