Mode Personalization in Trip-Based Transit Routing

Vassilissa Lehoux
NAVER LABS Europe, Meylan, France
https://europe.naverlabs.com/people_user/vassilissa-lehoux/
firstname.lastname@naverlabs.com

Darko Drakulic
NAVER LABS Europe, Meylan, France
https://europe.naverlabs.com/people_user/darko-drakulic/
firstname.lastname@naverlabs.com

Abstract

We study the problem of finding bi-criteria Pareto optimal journeys in public transit networks. We extend the Trip-Based Public Transit Routing (TB) approach [18] to allow for users to select modes of interest at query time. As a first step, we modify the preprocessing of the TB method for it to be correct for any set of selected modes. Then, we change the bi-criteria earliest arrival time queries, and propose a similar algorithm for latest departure time queries, that can handle the definition of the allowed mode set at query time. Experiments are run on 3 networks of different sizes to evaluate the cost of allowing for mode personalization. They show that although preprocessing times are increased, query times are similar when all modes are allowed and lower when some part of the network is removed by mode selection.

2012 ACM Subject Classification Mathematics of computing → Graph theory

Keywords and phrases Public transit, Route planning, Personalization

Digital Object Identifier 10.4230/OASIcs.ATMOS.2019.13

1 Introduction

In public transit networks, part of the information is available in the form of timetables that give the schedules at given stations of different public transportation modes such as buses, trains or tramways. Transfers between those modes of transportation are possible by walking between the stations, if they are not too far away from one another.

Finding paths in such public transit networks is highly relevant in practice, as millions of users use routing applications such as Naver Map[1], Citymapper[2] or Google Maps[3] to plan their trips daily. In the recent years, research has been very active for this problem [3] and many dedicated techniques, such as Transfer Patterns [2], RAPTOR [5] or CSA [6] have been developed to make this routing efficient.

However, defining criteria and constraints to optimize those paths according to user preferences is a complicated task. Many criteria can be considered, such as earliest arrival time (given a start time or a start time range), latest departure time (given an arrival time or an arrival time range), number of transfers, travel cost, total transfer duration, total waiting time, etc. In multicriteria optimization, a solution is said to be *dominated* in the Pareto sense if there is another solution that is strictly better on one criteria and at least as good on the others. It is frequent to look for either the Pareto set, that is all the

[1] https://map.naver.com/
[2] https://citymapper.com/
[3] https://www.google.com/maps

19th Symposium on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2019).
Editors: Valentina Cacchiani and Alberto Marchetti-Spaccamela; Article No. 13; pp. 13:1–13:15

OpenAccess Series in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

non-dominated solutions in the Pareto sense, or the Pareto front, that is the image of the Pareto set in the criteria space. For minimum total cost path problems, considering two or more criteria often makes the problem of finding all the optimal solutions intractable, with possibly exponential size Pareto sets [11], although the number of optimal solutions in the Pareto sense can be manageable in practice for some problems [15]. In this work, we consider the following polynomial bi-criteria problems: minimizing the arrival time and the number of transfers and maximizing departure time and minimizing the number of transfers. We are interested in finding the Pareto front, rather than the Pareto set, and we want to be able to compute one solution corresponding to each element of the Pareto front for minimum number of transfers and either minimum arrival time or latest departure time. Note that in that case, the maximum number of solutions returned is bounded by the number of public transport trips (as we cannot make more transfers) and is hence polynomial in the size of the instance. Providing sets of solutions rather than a single solution enables users to make their own compromises between the number of transfers and the travel time of the trip, according to their preferences. We choose not to compute the complete Pareto set, but only one solution for each element in the Pareto front to ensure polynomial time construction of the set of solutions. Note that the actual Pareto set can be much larger in practice. Indeed, in multimodal networks, having several solutions with the same value in the criteria space can be frequent for the considered criteria, for instance solutions sharing the same final (resp. initial) trip when optimizing earliest arrival time (resp. latest departure time) and number of transfers. In a theoretical network, it can be of exponential size.

As a second step toward more personalized solutions, we consider an additional constraint: at query time, the user can exclude some modes of transportation from the network. For example, a user want to avoid buses, because he/she thinks that they are not reliable enough.

Indeed, the type of transit modes is an important vector of choice between itineraries. In addition to the speed to reach destination, it impacts the price, the comfort, and of course, some modes are more or less appreciated by the user depending on his/her preferences. In many cases, the mode type information is available in the transit data. For instance, the General Transit Feed Standard format [10], very often used to describe public transit information, proposes this information as mandatory. In an itinerary planning application or website, the user will often be able to choose the modes that he/she wants to enable or disable in the interface as option for the search.

Several methods have been designed for mode related personalization. In [12], the authors consider a graph based approached with a time-dependent model of the timetables [16]. They propose personalized mode sequences described by a regular language and solve the associated *regular language constrained shortest path problem* using a combination of the D_{RegLC} [1] and ALT [9] algorithms called State-Dependent ALT. Preprocessing time is light (less than a minute on Île-De-France transportation network), but the languages involved must be defined at the preprocessing step. The User-Constrained Contraction Hierarchy [7] on the other hand, doesn't have this restriction and the mode sequence can be defined at query time. It is also based on a time-depend model for the transit modes and uses Contraction Hierarchy [8] on each mode's network to speed up the search. As a consequence, preprocessing time can be important on large networks (42 min for a small Europe network with 30K stations).

If we want to select the enabled scheduled modes for a query, rather than defining specific mode sequences, dynamic programming approaches such as CSA or RAPTOR might be used with very few modifications. The Connection Scan Algorithm [6] is based on a sorted connections array that contains all the trip segments between two consecutive stops. You can pass from one connection to the next if they are one after another in the same trip or if

you can leave the first connection and reach the next on time to take it (for instance by a walking transfer). This algorithm could be modified for mode personalization by pruning at query time the search space by taking only connections corresponding to allowed scheduled modes. Similarly, the RAPTOR algorithm [5] works directly with timetable information. It uses a round-based approach where in each round, trips are taken from lines passing at stops reached at the preceding iteration. In this context, some trips of disabled scheduled modes could also be avoided at query time by saving and checking the mode of each line. This approach can be found in [17] where a wider set of mode sequences is considered. Another approach can be found in [4], where the authors use the number of buses as an additional criterion for computing a subset of the Pareto set. They can hence obtain optimal solutions with no buses (as well as solutions with several bus trips).

In this work, we are interested in extending the Trip-Based Public Transit Routing approach [18] (TB) in order to be able to personalize the set of scheduled modes used at query time. TB is a round-based approach, iterating on the maximum number of transfers allowed in a solution, that relies on a different graph model: the nodes of the graph are the trips, while the arcs represent possible transfers. A preprocessing phase computes a non-minimal arc set T such that for any value in the Pareto front, there exists a solution S with this value such that all the transfers of S belong to T. Search phase is then breadth-first search like and builds one solution for each element of the Pareto front for minimum arrival time and minimum number of transfers. Note that the author uses a slightly modified definition of Pareto dominance to call the set built *Pareto set*, but here we choose to keep the standard definition.

2 Preliminaries

As we extend the TB algorithm, we give in this section a brief description of this method and discuss some of the claims made by the author in his article.

2.1 Notations

We will use notations similar to that of [18] to describe the public transit network. A sequence of stops $\overrightarrow{p}(t) = \langle p_t^1, p_t^2, \dots \rangle$ is associated with each trip t. The schedule of t is defined by the arrival and departure times of t at the stops of its sequence. We denote by $\tau_{arr}(t, i)$ (resp. $\tau_{dep}(t, i)$) the arrival time (resp. departure time) of t at the i^{th} stop of $\overrightarrow{p}(t)$. Trips are grouped into lines, that do not exactly represent the routes of the public transport network. First, all the trips of a line L have exactly the same sequence of stops, denoted $\overrightarrow{p}(L) = \langle p_L^1, p_L^2, \dots \rangle$. Second, all the trips of a line are completely ordered following comparison relations $\preceq$ and $\prec$ defined for two trips having the same sequence:

$$\begin{cases} t \preceq u \iff \forall i \in [0, |\overrightarrow{p}(t)|), \quad \tau_{arr}(t,i) \leq \tau_{arr}(u,i) \\ t \prec u \iff t \preceq u \text{ and } \exists i \in [0, |\overrightarrow{p}(t)|), \quad \tau_{arr}(t,i) < \tau_{arr}(u,i) \end{cases}$$

L_t denotes the line of trip t. For a given stop s, we define $\mathbf{L}(s)$ as the set of all pairs (L, i) with L a line and i an index in the sequence of L such that $s = p_L^i$. A displacement between the i^{th} stop of t and the j^{th} stop of t using trip t is denoted $p_t^i \to p_t^j$ and similarly, a walking transfer between trip t at the i^{th} station and trip u at the j^{th} station is denoted $p_t^i \to p_u^j$. Walking transfer times are defined for any pair of stops (p, q), $p \neq q$ that are close enough of one another and the associated duration is $\Delta\tau_{fp}(p, q)$. When transferring between two trips at a given station ($p_t^i = p_u^j = p$), a minimum change time $\Delta\tau_{fp}(p, p)$ can be defined, to represent the time needed to move within this station.

2.2 Preprocessing

The aim of the preprocessing of the TB algorithm is to build a set T of transfers between trips such that any transfer $p_t^i \to p_u^j$ of T is *feasible*, that is $\tau_{arr}(t, i) + \Delta\tau_{fp}(p_t^i, p_u^j) \leq \tau_{dep}(u, j)$, and for any element of the Pareto front, there exists an optimal solution with this element as value, such that all its transfers $p_t^i \to p_u^j$ belong to T. Note that the algorithm does not require the set T to be minimal, that is to be a set of minimum cardinality with this property. It needs only to be *correct*, that is to contain only feasible transfers and to contain all the transfers of at least one optimal solution per element in the Pareto front.

In order to compute a correct set of transfers, the preprocessing considers the transfers from each trip separately, which allows for trivial parallelization of the algorithm. For a given trip t, starting from the last stop of $\overrightarrow{p}(t)$ and taking the sequence in reverse order, we consider for each stop p_t^i with $i > 1$ all the reachable stops q from p_t^i (i.e. such that $\Delta\tau_{fp}(p_t^i, q)$ is defined). We set the earliest arrival and change times at each of those stops at $\tau_{arr}(t, i) + \Delta\tau_{fp}(p_t^i, q)$ and we look for transfers to all the possible lines passing by q. For each of those lines, we find the earliest trip u such that the transfer $p_t^i \to p_u^j$ is feasible (with $p_u^j = q$ and $j < |\overrightarrow{p}(u)|$). We remove U-turn transfers as described in [18]. Then, in order to reduce the transfer set, we try and update or set earliest arrival and earliest change times at the stops later in the sequence of u or at the stops reachable from those stops. If any improvement occurs or a new stop is reached, the transfer is kept. If not, it will be removed from the set of transfers. Note that if two transfers are equivalent (in term of reachable stops and arrival and change times at those stops), only the first one generated will be kept. Since we are looking at the stop sequence $\overrightarrow{p}(t)$ in reverse order, the later transfers are added to the set before equivalent earlier transfers that will be checked later in the process.

2.3 Query phase

The TB algorithm deals with earliest arrival time queries, where given a start time, the objective is to find the Pareto front for earliest arrival time and number of transfers.

At initialization, origin trip segments (trips segments whose boarding stop can be reached by a walking displacement from the origin point) and destination trip segments (trips segments from the unboarding stop of which the destination point can be reached) are computed. Origin trips are added to a queue. Note that instead of considering only departure from stops, it is possible to consider a departure from any location on the transportation network by allowing to compute shortest paths in the walking network to or from the closest stops.

The query phase is then a breadth-first search like procedure in the graph whose nodes are the trips and whose arcs are the possible transfers. At each iteration n, all the trip segments of queue Q_n are processed in order. If one is a destination trip, current earliest arrival time at destination is updated. It can be used to prune the search, by not adding transfers to the queue if they cannot be part of a solution that improves on this arrival time. For a given trip segment of Q_n, all possible transfers from it are performed, increasing the number of transfers by one in the partial solutions computed. When a transfer reaches a trip segment that is not marked, the trip segment is added to the queue for the next iteration and the trip itself and all the corresponding later trip segments of the same line are marked as processed. Iterations continue until maximum transfer number has been reached or current arrival time at destination cannot be improved (which means that the queue of trip segments is empty). The earliest arrival time itinerary to destination obtained at iteration k (when it exists) is hence an earliest arrival time itinerary with at most k transfers. The Pareto front for earliest arrival time and number of transfers is hence generated during the search phase.

2.3.1 Construction of the solution

Witt [18] claims to optimize latest departure time as a secondary criterion to break ties, but the construction proposed does not ensure this property. To build solutions, he suggests to store for each trip segment a pointer to its origin trip segment when the trip is added to the queue, in order to rebuild the sequence of trips and to compute optimal transfer between the trips as a postprocessing. Consider the very simple example shown in Figure 1.

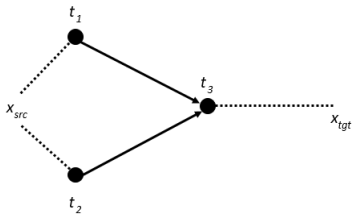

Figure 1 Small network with 3 trips and 2 transfers.

The network consists in 2 lines, one with trips t_1 and t_2 such that $t_1 \prec t_2$, one with trip t_3. Both trips t_1 and t_2 can transfer from their i^{th} stop to the same index j of trip t_3. When preprocessing those transfers, both are kept in the set as they update the arrival times of the stops of t_3. In the search phase, starting from source x_{src}, trip t_1 is the earliest trip of the line that can be reached and is hence added to the queue. Then transfer to t_3 is done to reach destination x_{tgt}. Hence, the sequence of trips is t_1 and then t_3, while with latest departure time as a secondary criterion, t_2 and then t_3 should have been returned. As there is no sorting on departure trips, a similar example can be built with trip t_2 belonging to another line if the search starts with t_1.

Note that in both cases, the transfers in the solution with latest departure time for the given earliest arrival time and number of transfers were in the computed set of transfers.

In order to actually find the maximum departure time for a given number of transfers and a given arrival time, it is possible to consider several strategies, such as using profile queries.

Profile queries are given a time range as input. In our case, for every possible start time (resp. arrival time) in that range, you want to compute optimal values of the Pareto front for earliest arrival time (resp. latest departure time) and number of transfers. In order to speed profile queries, Witt proposes for earliest arrival time profile queries to start by the end of the time range and to iterate backward on this time interval. The idea is to keep the labels of the preceding time step as journey starting later never dominates earlier ones.

Consider an earliest arrival time solution S with value (τ_{arr}, k) in the Pareto front obtained for start time τ. In order to find the latest departure time for which there exists a solution with value (τ_{arr}, k), it is possible to make a profile query with departure time range $[\tau, \tau_{arr}]$. Going backward, we compute the Pareto front for each instant of the range. Unless origin and destination are equal, for a departure time $t = \tau_{arr}$, value (τ_{arr}, k) doesn't belong to the Pareto front, but it will belong to it for a departure time of τ. Hence, decreasing the minimum departure time value from τ_{arr} to τ, (τ_{arr}, k) will belong to the Pareto set at a certain iteration. The latest departure time associated with (τ_{arr}, k) is the first instant when value (τ_{arr}, k) belongs to the Pareto set.

3 Scheduled mode selection

The preprocessing phase of the TB algorithm removes transfers from the search phase without modifying the Pareto front when all the modes are available. Consider the example of Figure 2.

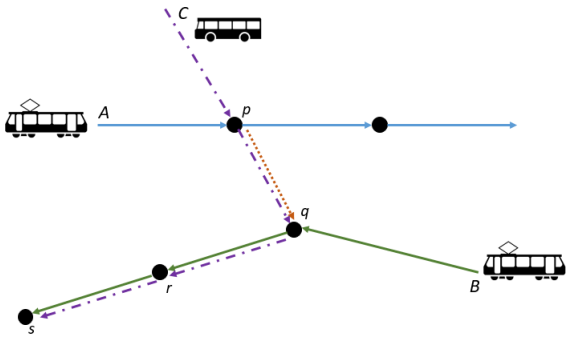

Figure 2 Example of transfer removed by the TB algorithm's preprocessing pruning phase.

Suppose that we have only those 3 lines in a public transportation network. Lines A and B are tramway lines (represented with solid lines) and the line C is a bus line (represented with a dashed line). A walking transfer (dotted line) is possible between the stop p on line A and the stop q on line B. Suppose that exploring the transfers from a trip from line A at stop p, we first look at the transfer from that trip to line C at stop p. We set the arrival and change times of the stops q, r and s and move to the next transfer. We now try and update stops' arrival times by transferring from the current trip of line A to B. The earliest trip of line B does not improve arrival or change times at stops r or s. It is hence pruned.

Now, consider that you want to allow only tramways for a given query. If buses cannot be taken, the removed transfer could have found itself in an optimal solution, for instance starting with the trip of A and arriving at s on line B.

A possible solution, to avoid loosing correctness, could be to have several instances of the routing service, each corresponding to a selection of transit modes. Each server would run on preprocessed data for the corresponding subnetwork. To consider all the possible combinations of modes, you will need $2^{|M|} - 1$ servers, if M is the set of modes of the network.

A second option, that we will use as a base line, it to use the complete set of transfers in the search phase, without any pruning based on arrival times. In [18], the author reports query times about 3 times as slow with this graph compared to the pruned one.

In this article, we propose a method for computing a reduced transfer set that is correct for any set of modes asked by the user at query time, which enables to answer that type of queries with a single server. This preprocessing step is explained in Section 3.1. Then, with such set as input, the search phase requires only a few modifications that will be explained in Section 4.

3.1 Preprocessing with mode selection

The aim of the preprocessing would now be to return a set of transfers T that is correct when enabling any subset μ of the set of all transit modes M.

In order to achieve this aim, we propose to change the pruning phase to be sure that for each value of the Pareto front and for any $\mu \subseteq M$ there is at least one solution with this value that uses only transfers of T. As before, we preprocess the transfers of each trip

separately. We denote $m_t \in M$ the mode associated with the line of trip t. As we are computing transfers from trip t, mode m_t needs to be allowed for the transfer to potentially belong to an optimal solution. So for a transfer to a line l of mode m to be in an optimal solution, at least mode m_t and mode m need to be in the subset μ of allowed modes.

For a given trip t, we propose to compute the contribution to the set of transfers T that will be used in the search phase in the following way. At each stop of the network, we try and update a minimum arrival time and minimum change time for any given subset μ of M such that $\mu = \{m, m_t\}$ with $m \in M$. Hence, at most $2|M|$ values are recorded for each stop. When transferring to a trip t of mode $m \in M \setminus \{m_t\}$, we can use the same procedure as before to update arrival and change times when using only modes m and m_t. When transferring to another trip of mode m_t, the arrival and change times for all the subsets of M are updated simultaneously, as mode m_t is necessarily allowed when transferring from t. We denote by $\tau_A(q, m)$ (resp. $\tau_C(q, m)$) the minimum arrival time (resp. minimum change time) found so far during the execution of the procedure when transferring from t for subset $\mu = \{m, m_t\}$ of allowed modes. The procedure is presented in Algorithm 1.

Algorithm 1 Pruning.

Input: Timetable data, footpath data, transfer set T
Output: Reduced transfer set T
for trip t **do**
  $\tau_A(., .) \leftarrow \infty$ ▷ Earliest arrival time at stops for a given mode subset
  $\tau_C(., .) \leftarrow \infty$ ▷ Earliest change time at stops for a given mode subset
  for $i \leftarrow |\overrightarrow{p}(t)| - 1, \ldots, 1$ **do**
    UPDATE$(p_t^i, m_t, m_t, \tau_{arr}(t,i), \tau_{arr}(t,i) + \tau_{fp}(p_t^i, p_t^i))$
    for each stop $q \neq p_t^i$ such that $\Delta\tau_{fp}(p_t^i, q)$ is defined **do**
      UPDATE$(q, m_t, m_t, \tau_{arr}(t,i) + \tau_{fp}(p_t^i, q), \tau_{arr}(t,i) + \tau_{fp}(p_t^i, q))$
    for each transfer $p_t^i \to p_u^j \in T$ **do**
      $keep \leftarrow$ **false**
      for each stop p_u^k on trip u with $k > j$ **do**
        $keep \leftarrow keep \vee \tau_{arr}(u,k) < \tau_A(p_u^k, m_u)$
        $keep \leftarrow keep \vee \tau_{arr}(u,k) + \tau_{fp}(p_u^k, p_u^k) < \tau_C(p_u^k, m_u)$
        UPDATE$(q, m_t, m_u, \tau_{arr}(u,k), \tau_{arr}(u,k) + \tau_{fp}(p_u^k, p_u^k))$
        for each stop $q \neq p_u^k$ such that $\Delta\tau_{fp}(p_u^k, q)$ is defined **do**
          $\rho \leftarrow \tau_{arr}(u,k) + \Delta\tau_{fp}(p_u^k, q)$
          $keep \leftarrow keep \vee (\rho < \tau_A(q, m_u)) \vee (\rho < \tau_C(q, m_u))$
          UPDATE$(q, m_t, m_u, \rho, \rho)$
      if $\neg keep$ **then**
        $T \leftarrow T \setminus \{p_t^i \to p_u^j\}$ ▷ No improvement: remove the transfer

procedure UPDATE(q, m_t, m_u, e, c)
  Input: stop q, mode m_t, mode m_u, arrival time e, change time c
  $\tau_A(q, m_u) \leftarrow \min(\tau_A(q, m_u), e)$
  $\tau_C(q, m_u) \leftarrow \min(\tau_C(q, m_u), c)$
  if $m_t = m_u$ **then** ▷ Mode m_t is allowed since we are transferring from it
    for each $m \in M \setminus \{m_t\}$ **do**
      $\tau_A(q, m) \leftarrow \min(\tau_A(q, m_t), \tau_A(q, m))$
      $\tau_C(q, m) \leftarrow \min(\tau_C(q, m_t), \tau_C(q, m))$

▶ **Proposition 1.** *Algorithm 1 computes a correct set of transfers for earliest arrival time and minimum number of transfers, for any subset μ of M.*

Proof. First, note that, when transferring from trip t, as m_t belongs to all the considered subsets of M, it is sufficient to update the value of the *keep* variable before the UPDATE procedure, even if $m_u = m_t$. Hence, for a given starting trip t, and an index i in its sequence of stops, *keep* will be set to TRUE whenever a transfer $p_t^i \to p_u^j$ improves the arrival time or the change time at a given stop for the subset $\mu = \{m, m_t\}$ of M. Hence if transfer $p_t^i \to p_u^j$ belongs to an optimal solution for $\mu = \{m, m_t\}$, at least one equivalent transfer is added to the set of transfers returned at the end of the procedure.

Now, consider an arbitrary subset μ of M and an optimal value (τ_{arr}, k) of the Pareto front with $1 \leq k$ and a solution s from the Pareto set of value (τ_{arr}, k). We represent this solution by the trip segment sequence that composes it:
$s = \left\langle p_{t_1}^{j_1} \to p_{t_1}^{i_1}, p_{t_2}^{j_2} \to p_{t_2}^{i_2} \ldots, p_{t_{k+1}}^{j_{k+1}} \to p_{t_{k+1}}^{i_{k+1}} \right\rangle$
Consider the last transfer $p_{t_k}^{i_k} \to p_{t_{k+1}}^{j_{k+1}}$ of s. Since s is an optimal solution for μ, it is not possible to arrive sooner at stop $p_{t_{k+1}}^{i_{k+1}}$ from trip segment $p_{t_k}^{j_k} \to p_{t_k}^{i_k}$. Hence, either $p_{t_k}^{i_k} \to p_{t_{k+1}}^{j_{k+1}}$ is in T or there is a transfer from t_k at $p_{t_k}^{i'_k}$, $i_k \leq i'_k$ in T leading to a trip with the same arrival time at $p_{t_{k+1}}^{i_{k+1}}$ for the subset $\left\{m_{t_k}, m_{t_{k+1}}\right\}$ of μ. Let $p_{t_k}^{i'_k} \to p_{t'_{k+1}}^{j'_{k+1}}$ be the transfer that is actually in T. Trip t'_{k+1} is either of mode m_{t_k} or of mode $m_{t_{k+1}}$ which both belong to μ.

Now consider the previous transfer in s, $p_{t_{k-1}}^{i_{k-1}} \to p_{t_k}^{j_k}$. Either this transfer is in T or, as $j_k \leq j'_k$, there exist another transfer from t_{k-1} in T that has a change time at least as early at stop $p_{t_k}^{j'_k}$. We denote $p_{t_{k-1}}^{j'_{k-1}} \to p_{t'_k}^{i'_k}$, with $j_{k-1} \leq j'_{k-1}$ the transfer actually in T. The transfer from t'_k at stop $p_{t_k}^{j'_k}$ to trip segment $p_{t'_{k+1}}^{j'_{k+1}} \to p_{t'_{k+1}}^{i'_{k+1}}$ with $p_{t'_{k+1}}^{i'_{k+1}} = p_{t_{k+1}}^{i_{k+1}}$ is feasible as trip t'_k has a change time at least as early as t_k at that stop. Hence, either it is in T or there is a transfer with at least as good an arrival time at $p_{t_{k+1}}^{i_{k+1}}$ for the subset of modes $\left\{m_{t'_k}, m_{t'_{k+1}}\right\} \subseteq \mu$ that is in T. Hence, going backward in the transfer of s, we can build a solution using a subset of the modes of s, with the same number of transfers, all its transfers being in T and the arrival time at $p_{t_{k+1}}^{i_{k+1}}$ being identical. This solution has therefore the same value as s. ◀

▶ **Lemma 2.** *Algorithm 1 computes a correct set of transfers for latest departure time and minimum number of transfers for any subset μ of M.*

Proof. As before, we need to prove that for any element (τ_{dep}, k) of the Pareto front, there exists a solution those transfers are in T such that the value of s is (τ_{dep}, k). Consider an instance $I_{dep} = (\tau, \mu, x_{org}, x_{tgt})$ of the latest departure time problem with τ the latest departure time, $\mu \subseteq M$ the allowed modes, x_{org} the origin and x_{dest} the destination. Let (τ_{dep}, k) be an optimal value of the Pareto front and τ_{arr} the earliest arrival time when starting at τ_{dep} and using at most k transfers. Let s be an optimal solution of the latest departure time problem for I_{dep} with value (τ_{dep}, k) and arrival time τ_{arr}.

Consider the following earliest arrival time problem and an instance $I_{arr} = (\tau_{dep}, \mu, x_{org}, x_{tgt})$ with the same origin, the same destination and a minimum departure time equals to τ_{dep}.

Suppose that a solution s' with k transfers arrives at τ_{arr} and leaves at $\tau > \tau_{dep}$. Then, it dominates s for the latest departure time problem and instance I_{dep} which is not possible as s is optimal. So no solution with k transfers can improve other s.

It remains the possibility of an optimal solution s'' with value (τ_{arr}, k') with $k' < k$ for the earliest arrival time problem with instance I_{arr} and departure time $\tau \geq \tau_{dep}$. Suppose

first that $\tau > \tau_{dep}$. In that case, solution s' dominates s for the latest departure time problem and instance I_{dep}, which is not possible as s is optimal. Hence $\tau = \tau_{dep}$. In that case, we have a contradiction as (τ_{dep}, k) is optimal for I_{dep} and would be dominated by (τ_{dep}, k').

So all the optimal solutions with k transfers of the earliest arrival time problem for I_{arr} start at τ_{dep} and arrive at τ_{arr}. From Proposition 1, T contains all the transfers of at least one of those solutions, that we denote s_{arr}. s_{arr} is also optimal for the latest departure time problem and instance I_{dep}, which completes the proof. ◀

4 Earliest arrival time and latest departure time queries

Algorithm 2 Latest departure time query.

Input: Transfer set T, origin x_{src}, destination x_{tgt}, latest arrival time τ, mode selection μ
Output: Pareto front J
$J \leftarrow \emptyset$ ▷ Pareto front
$\mathcal{L} \leftarrow \emptyset$ ▷ Target lines
$Q_n \leftarrow \emptyset$ for all $n = 1, 2, \ldots$ ▷ Queue of trips for each iteration
$R(t) \leftarrow 0$ for all trips t ▷ Maximum index at which a trip is unboarded during the search
for each stop q such that $\Delta\tau_{fp}(x_{src}, q)$ is defined **do**
  $\Delta\tau \leftarrow 0$ if $x_{src} = q$, else $\Delta\tau_{fp}(x_{src}, q)$
  for each $(L, i) \in \mathbf{L}(q)$ such that $m_L \in \mu$ **do**
    $\mathcal{L} \leftarrow \mathcal{L} \cup \{(L, i, \Delta\tau_{fp}(x_{src}, q))\}$
for each stop q such that $\Delta\tau_{fp}(q, x_{tgt})$ is defined **do**
  $\Delta\tau \leftarrow 0$ if $x_{tgt} = q$, else $\Delta\tau_{fp}(q, x_{tgt})$
  for each $(L, i) \in \mathbf{L}(q)$ such that $m_L \in \mu$ **do**
    $t \leftarrow$ latest trip of L such that $\tau_{arr}(t, i) + \Delta\tau \leq \tau$
    BW_ENQUEUE$(t, i, 0)$
$n \leftarrow 0$
$\tau_{\max} \leftarrow 0$
while $Q_n \neq \emptyset$ **do**
  for each $p_t^b \rightarrow p_t^e \in Q_n$ **do**
    for each $(L, i, \Delta\tau) \in \mathcal{L}$ with $b \leq i < e$ and $\tau_{dep}(t, i) - \Delta\tau > \tau_{\max}$ **do**
      $\tau_{\max} \leftarrow \tau_{dep}(t, i) - \Delta\tau$
      $J \leftarrow J \cup \{(\tau_{\max}, n)\}$ and remove dominated entries
    for each $p_u^i \rightarrow p_t^j \in T$ with $b \leq j < e$ and $m_t \in \mu$ **do**
      if $\tau_{dep}(u, i-1) < \tau_{\max}$ **then**
        BW_ENQUEUE$(u, i, n+1)$
  $n = n + 1$
return J
procedure BW_ENQUEUE(trip t, index i, nb transfers n)
  if $R_n(t) < i$ **then**
    $Q_n \leftarrow Q_n \cup \left\{p_t^{R_n(t)} \rightarrow p_t^i\right\}$
    for each trip u such that $L_t = L_u$ and $u \preceq t$ **do**
      $R(u) \leftarrow \max(R(u), i)$

In order to adapt earliest arrival time queries from [18] to transit mode selection, only a few modifications are necessary. First, only add to the queue by the ENQUEUE procedure trips that belong to the selected set of modes μ. Then, when considering transfers from a given mode, only scan transfers to modes that belong to μ. The set of transfers being correct for any value of μ with preprocessing of Section 3.1, the search will compute the Pareto front.

Table 1 Data sets used for the experiments.

	stops	trips	lines	foot paths	connections	Modes
TCL	4583	70614	578	87834	1425044	Bus, subway, tram, funicular
IDFM	42404	351908	1869	1061959	7803633	Bus, subway, rail, tram, funicular
Korea	180948	446741	31708	4195659	22346975	Bus, subway, rail, tram

In order to deal with latest departure time queries, we propose the following modifications of the base algorithm. Basically, the search is a backward search in the graph of trips and transfers. When we add a trip t to the queue, we hence mark the maximum index $R(t)$ at which t is unboarded rather than the minimum index at which it is taken (see procedure BW_ENQUEUE of Algorithm 2). When a trip segment of the queue is processed, we first check if we can improve on the latest departure time found so far and update the Pareto front accordingly. Then transfers are scanned, and the origin trip segment of the transfer is potentially added to the queue. To take into account mode selection, the same modifications as before are necessary, as we check the modes of the trip segments before adding them to the queue. The algorithm can be found in Algorithm 2.

5 Experiments

The experiments are run on a 64 2.7 GHz CPU Intel(R) Xeon(R) CPU E5-4650 server with 20 M of L3 cache and 504 GB of RAM. We perform our tests on three data sets. The first, the TCL [13] (Transports en Commun de Lyon) data set, is made available by the Grand Lyon metropolitan area for research purpose. The second covers the Ile-De-France area and is provided by Île de France Mobilités [14] with permissive license. We denote it IDFM. Note that although it has been used in previous publications, it might be different to the one cited due to regular updates. In [12], for instance, the size of the IDF network is closer to that of the TCL data set. The last is a proprietary data set for whole Korea. For those data sets, we use a mixture of the provided footpaths (if any) and generated footpaths. Closure of the footpaths is not required by the TB algorithm or our adaptation (as opposed to RAPTOR [5] or CSA [6]) but users will often accept to walk, for limited distances, between stations. Hence, for each stop, we include footpaths to all the stops reachable within a distance of 600 m, using a walking speed of 3.6 kph. Data set information is summarized in Table 1.

Table 2 Comparison of preprocessing steps between MS, STD and NP versions.

Data Set	TCL	IDFM	Korea
Preproc. MS (s)	18	521	1326
Preproc. STD (s)	6	141	381
Preproc. NP (s)	4	56	69
Preproc. MS/STD	3.0	3.7	3.5
# transfers MS (in million)	11.7	110.7	259
# transfers STD (in million)	10.9	103.6	245
# transfers NS (in million)	136	1984	3479
# transfers per trip MS	166	315	580
# transfers per trip STD	155	295	571
# transfers per trip NS	1952	5651	7800

Table 2 compares the results of the preprocessing obtained with the standard version (STD) and with the modified version that allows correct results for a selection of modes (MS). As an additional base line, we also run our experiment with a version that performs no pruning, but implements the mode selection in the search phase (NP). As it uses the complete set of transfers, this baseline will give correct solution sets.

As expected, the number of transfers in T is only slightly increased by the modifications of the preprocessing. On the other hand, the duration of the preprocessing is significantly increased. This is probably explained by the interconnection of the different networks: a large part of the lines will be able to connect to most of the different modes and hence, earliest arrival times are updated for nearly all trips and modes.

In order to test the effect of our modifications on query times, we generate 500 random origin-destination pairs for each data set. We compare in Table 3 our 3 implementations (with and without mode selection, and mode selection without pruning). Note that our execution times include the computation of one solution for each element in the Pareto front. The average and maximum number of solutions for the different test sets can be found in Appendix A in Table 6. As expected from Witt [18], the version without pruning is much slower (more than 3 times slower on Korea and IDFM) due to the larger number of transfers in the search phase.

Table 3 Comparison of query times between MS, STD and NP versions.

Data set	Algorithm	EAT (ms)	profile 1H (ms)	profile day (ms)
TCL	MS	12	56	366
TCL	STD	20	55	304
TCL	NP	34	123	702
IDFM	MS	57	157	848
IDFM	STD	57	173	857
IDFM	NP	382	1007	6432
Korea	MS	46	236	1922
Korea	STD	51	239	1940
Korea	NP	148	940	8042

For queries with sets of allowed modes, we try and remove different scheduled modes and look at the influence on query times. Note that for the 3 data sets, the network contains a majority of bus trips. Table 4 compares our results with the no pruning base line. We also provide the results of [12] in Table 5 as an example of integration in the time-expended model: although the data set used is not exactly identical to ours, the algorithm also builds solutions (only the earliest arrival time one) and results are provided for several mode selections. Note that for [12], the query times are similar for the different mode selections, but with the TB modifications that we propose, we see that removing parts of the public transit network improves the query time. It is expected as the transfers to disabled modes will not be performed during the search, effectively reducing the number of trip segments processed (see Table 7 in Appendix A). This property also holds for the version without pruning, but although forbidding some modes reduces the execution times, the improvement brought by the pruning is still clear, especially on more time consuming profile instances.

Table 4 Comparisons of query times for several selections of modes.

Data set	Forbidden modes	MS EAT (ms)	NP EAT (ms)	MS profile 1H (ms)	NP profile 1H (ms)	MS profile day (ms)	NP profile day (ms)
TCL	None	12	34	56	123	366	702
TCL	Bus	5	10	16	26	96	128
TCL	Bus, train	5	10	15	29	96	126
TCL	Subway	10	25	41	195	277	528
TCL	Subway, tram, train	10	21	37	88	223	445
IDFM	None	57	382	173	1007	857	6432
IDFM	Bus	17	72	34	126	114	647
IDFM	Bus, tram, train	8	25	18	56	56	257
IDFM	Subway	50	318	135	1621	698	5165
IDFM	Subway, tram, train	51	319	144	816	680	4392
Korea	None	46	148	236	940	1922	8042
Korea	Bus	28	56	47	82	185	340
Korea	Bus, tram, train	25	48	44	76	172	323
Korea	Subway	39	112	227	892	1746	7313
Korea	Subway, tram, train	37	116	216	923	1759	7698

Table 5 Query times of SDALT from [12] for several selections of modes.

Data set	Forbidden modes	EAT (ms)
IDFM - SDALT	None	186
IDFM - SDALT	Bus, train	175
IDFM - SDALT	Subway, tram, train	216

6 Conclusion

In this article, we present an extension of the Trip-Based Public Transit Routing algorithm [18]. It enables the user to select any subset of the possible scheduled modes at query time as the enabled modes for the query. The preprocessing time is increased by the modification, but we show that it guaranties that the Pareto front is returned by the algorithm, and that, similarly to the standard version, it significantly improves the query times. We also prove that the computed transfer set is still correct for latest departure time queries, that we propose as an extension. Query times are not much impacted when all the modes are allowed, and removing any scheduled mode from the list of the enabled modes reduces the computation time significantly, making those personalized queries faster than the regular ones.

A perspective of this work could concerns the adaptation of the Trip-Based Public Transit Routing using condensed search trees [19] to mode selection. In his article, Witt propose to a speed-up technique based on the idea of Transfer Patterns [2]. A specific search graph is precomputed for each origin from one-to-all all day profile queries. The result of those queries is of course dependent of the allowed lines and hence the obtained search graphs cannot be used directly to compute optimal queries for all possible mode selections. As the preprocessing time is important even for the standard version (231 hours for Germany on 64 threads), trade-off between correctness and preprocessing execution times might be needed for enabling mode selection at query time.

References

1 Christoper L. Barrett, Riko Jacob, and Madhav Marathe. Formal-language-constrained path problems. *SIAM Journal on Computing*, 30(3):809–837, 2000. doi:10.1137/S0097539798337716.

2 Hannah Bast, Erik Carlsson, Arno Eigenwillig, Robert Geisberger, Chris Harrelson, Veselin Raychev, and Fabien Viger. Fast routing in very large public transportation networks using transfer patterns. In *Proceedings of the 18th Annual European Conference on Algorithms: Part I*, ESA'10, pages 290–301, Berlin, Heidelberg, 2010. Springer-Verlag. URL: http://dl.acm.org/citation.cfm?id=1888935.1888969.

3 Hannah Bast, Daniel Delling, Andrew Goldberg, Matthias Müller-Hannemann, Thomas Pajor, Peter Sanders, Dorothea Wagner, and Renato F. Werneck. *Algorithm Engineering: Selected Results and Surveys*, chapter Route Planning in Transportation Networks, pages 19–80. Springer International Publishing, Cham, 2016. doi:10.1007/978-3-319-49487-6_2.

4 Daniel Delling, Julian Dibbelt, and Thomas Pajor. Fast and exact public transit routing with restricted pareto sets. In Stephen Kobourov and Henning Meyerhenke, editors, *Proceedings of the Twenty-First Workshop on Algorithm Engineering and Experiments (ALENEX)*, pages 54–65, 2019. doi:10.1137/1.9781611975499.5.

5 Daniel Delling, Thomas Pajor, and Renato F. Werneck. Round-based public transit routing. In *Proceedings of the Fourteenth Workshop on Algorithm Engineering and Experiments (ALENEX)*, pages 130–140, 2012. doi:10.1137/1.9781611972924.13.

6 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. SEA 2013*, volume 7933 of *Lecture Notes in Computer Science*, pages 43–54, Berlin, Heidelberg, 2013. Springer. doi:10.1007/978-3-642-38527-8_6.

7 Julian Dibbelt, Thomas Pajor, and Dorothea Wagner. User-constrained multi-modal route planning. In SIAM, editor, *Proceedings of the 14th Workshop on Algorithm Engineering and Experiments (ALENEX'12)*, pages 118–129, 2012. doi:10.1137/1.9781611972924.12.

8 R. Geisberger, P. Sanders, D. Schultes, and D. Delling. Contraction hierarchies: Faster and simpler hierarchical routing in road networks. In Catherine C. McGeoch, editor, *Proceedings of the 7th Workshop on Experimental Algorithms (WEA'08)*, volume 5038 of *Lecture Notes in Computer Science*, pages 319–333, Berlin, Heidelberg, 2008. Springer. doi:10.1007/978-3-540-68552-4_24.

9 Andrew V. Goldberg and Chris Harrelson. Computing the shortest path: A search meets graph theory. In *Proceedings of the Sixteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '05, pages 156–165, Philadelphia, PA, USA, 2005. Society for Industrial and Applied Mathematics. URL: http://dl.acm.org/citation.cfm?id=1070432.1070455.

10 General Transit Feed Standard (GTFS). https://developers.google.com/transit/gtfs/reference/.

11 Pierre Hansen. Bicriterion path problems. In Günter Fandel and Tomas Gal, editors, *Multiple Criteria Decision Making Theory and Application. Proceedings of the Third Conference Hagen/Königswinter, West Germany, August 20-24, 1979*, volume 177 of *Lecture Notes in Economics and Mathematical Systems*, pages 109–127, Berlin, Heidelberg, 1980. Springer. doi:10.1007/978-3-642-48782-8_9.

12 Dominik Kirchler, Leo Liberti, and Roberto Wolfler Calvo. Efficient computation of shortest paths in time-dependent multi-modal networks. *Journal of Experimental Algorithmics*, 19:1–29, January 2014. doi:10.1145/2670126.

13 Data Grand Lyon. https://data.grandlyon.com/.

14 Île De France Mobilités. Open data. URL: https://opendata.stif.info.

15 Matthias Müller-Hannemann and Karsten Weihe. Pareto shortest paths is often feasible in practice. In Gerth Stølting Brodal, Daniele Frigioni, and Alberto Marchetti-Spaccamela, editors, *Algorithm Engineering. WAE 2001*, pages 185–197, Berlin, Heidelberg, 2001. Springer. doi:10.1007/3-540-44688-5_15.

16 Evangelia Pyrga, Frank Schulz, Dorothea Wagner, and Christos Zaroliagis. Efficient models for timetable information in public transportation systems. *ACM Journal of Experimental Algorithmics*, 12(2.4):1–39, 2008. doi:10.1145/1227161.1227166.

17 Luis Ulloa, Vassilissa Lehoux, and Frédéric Rouland. Trip planning within a multimodal urban mobility. *IET Intelligent Transport Systems*, 12(2):87–92, 2018. doi:10.1049/iet-its.2016.0265.

18 Sascha Witt. Trip-based public transit routing. In Nikhil Bansal and Irene Finocchi, editors, *Algorithms - ESA 2015*, volume 9294 of *Lecture Notes in Computer Science*, pages 1025–1036, Berlin, Heidelberg, 2015. Springer. doi:10.1007/978-3-662-48350-3_85.

19 Sascha Witt. Trip-based public transit routing using condensed search trees. In Marc Goerigk and Renato Werneck, editors, *Proccedings of the 16th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS'16)*, number Article No. 10, pages 10:1–10:12, 2016. doi:10.4230/OASIcs.ATMOS.2016.10.

A Additional numerical results

In this section, we add figures about the earliest arrival time and profile queries. In Table 6, the mean and maximum number of solutions for earliest arrival time and profile queries on the different networks are compared. In Table 7, the mean total number of elements in the queues are displayed for the Korean network for each set of forbidden modes used during the experiments.

Table 6 Comparisons of mean and max number of solutions for several selections of modes.

Data set	Forbidden modes	mean - EAT	max - EAT	mean - profile 1H	max - profile 1H	mean - profile day	max - profile day
TCL	None	2.82	7	28.36	117	268.75	1152
TCL	Bus	1.17	5	8.27	89	87.95	975
TCL	Bus, tram, train	1.17	5	8.25	82	87.91	953
TCL	Subway	2.48	6	21.82	87	205.12	778
TCL	Subway, tram, train	2.42	6	19.84	63	176.03	598
IDFM	None	2.15	6	5.78	57	25.2	419
IDFM	Bus	1.12	4	2.1	60	8.8	403
IDFM	Bus, tram, train	1.05	4	2.31	60	13.28	403
IDFM	Subway	2.12	6	9.41	32	14.56	269
IDFM	Subway, tram, train	4.06	10	16.09	41	98.2	337
Korea	None	2.94	10	32.78	80	317.84	791
Korea	Bus	1.32	5	11.72	44	105.11	320
Korea	Bus, tram, train	1.32	5	11.71	44	105.28	324
Korea	Subway	2.59	7	26.44	62	224.5	695
Korea	Subway, tram, train	2.58	7	28.17	63	273.1	695

Table 7 Comparisons of mean queue sizes for several selections of modes on the Korean network.

Data set	Forbidden modes	MS EAT (k)	NP EAT (k)	MS profile 1H (k)	NP profile 1H (k)	MS profile day (k)	NP profile day (k)
Korea	None	35.6	87.5	183.2	373.8	1881.9	3536.2
Korea	Bus	9.7	11.3	13.8	17.0	71.6	76.2
Korea	Bus, tram, train	8.5	9.4	13.5	14.5	65.4	68.3
Korea	Subway	30.7	78.2	169.6	358.0	1668.1	3213.3
Korea	Subway, tram, train	28.6	72.8	166.0	350.6	1650.5	3193.8