Faster Preprocessing for the Trip-Based Public Transit Routing Algorithm

Vassilissa Lehoux[1]

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

Christelle Loiodice
NAVER LABS Europe, Meylan, France
https://europe.naverlabs.com/people_user/christelle-loiodice/
firstname.lastname@naverlabs.com

Abstract

We propose an additional preprocessing step for the Trip-Based Public Transit Routing algorithm, an exact state-of-the art algorithm for bi-criteria min cost path problems in public transit networks. This additional step reduces significantly the preprocessing time, while preserving the correctness and the computation times of the queries. We test our approach on three large scale networks and show that the improved preprocessing is compatible with frequent real-time updates, even on the larger data set. The experiments also indicate that it is possible, if preprocessing time is an issue, to use the proposed preprocessing step on its own to obtain already a significant reduction of the query times compared to the no pruning scenario.

2012 ACM Subject Classification Mathematics of computing → Graph algorithms

Keywords and phrases Public transit, Route planning, Algorithms, Preprocessing

Digital Object Identifier 10.4230/OASIcs.ATMOS.2020.3

1 Introduction

In public transit networks, itineraries can combine public transit lifts with walking between the stations. The *schedules* or *timetables* describe the arrival and departure times of the vehicles at the public transit stations, also called *stops*. Information for transfers, on the other hand, contains walking times between pairs of stops.

Given an origin, a destination and a start time, we consider the problem of finding optimal compromise paths for two criteria to minimize: arrival time and number of transfers. Those two criteria are of high practical relevance as they are important in the user's choice of an itinerary using public transportation.

In multicriteria optimization, the notion of *Pareto dominance* is often used to define the optimality of the solutions. A solution s is *dominated in the Pareto sense* by a solution s' for a set $\{c_1, c_2, \ldots, c_r\}$ of criteria to minimize if $\forall i \in \{1, 2, \ldots, r\}$, $c_i(s') \leq c_i(s)$ and $\exists i \in \{1, 2, \ldots, r\}$ such that $c_i(s') < c_i(s)$. The *optimal* solutions are then the *non-dominated* solutions. Those non-dominated solutions represent compromises between the different criteria as the value of one cannot be improved without degrading the value of another. The set of all the optimal criteria values of the non-dominated solutions is called *Pareto front*, while the maximal set of non-dominated solutions is called *Pareto set*. For two or more additive criteria to minimize, the Pareto set of the multi-objective shortest path problem can be of exponential size [14], which makes the problem of generating it intractable. As an alternative, many authors consider only *complete* optimal solution sets (we borrow the

[1] Corresponding author

term from [21]), that is solution sets that contains only one optimal solution with this value for each element of the Pareto front. Indeed, depending on the criteria, those sets can be of polynomial size, or simply much smaller than the maximal set. Typically, if some of the criteria can take only a bounded number of values, then it limits the number of elements in the Pareto front. Several public transit routing algorithms such as RAPTOR [10], CSA [12] or Public Transit Labeling [8], hence compute only a complete set for minimum number of transfers and earliest arrival time at destination, while in more multi-criteria context, authors go farther and prune the complete set to reduce its size [2, 7].

Over the years, many algorithms, dedicated to public transit networks have been designed. In fact, even if multimodal or public transit networks can be modeled directly as a graph [13, 15, 20], where classical shortest path techniques for road networks can be applied, those methods, if not adapted, are not as efficient on public transit networks, since the structure of the public transit information is different [1]. As a consequence, dealing with large scale graphs such as large metropolitan areas or small countries demands specific techniques in order to obtain low computation times. It remains the case even for polynomial problems, such as the problem of finding a complete solution set for earliest arrival time and minimum number of transfers, where the number of values in the Pareto front is bounded by the number of trips, as has been remarked in [16]. Many of those techniques rely on a *preprocessing step* to compute information that will be used in the search phase in order to reduce the query times compare to classical routing algorithms. There is often a trade-off between preprocessing time, amount of auxiliary data and query times, different for each algorithm. An overview of acceleration techniques can be found in [4].

When the preprocessing is based on the schedule information, the preprocessing time is an important aspect for integrating easily real-time updates. If the chosen technique has large preprocessing time, it will not be possible to rerun the preprocessing for each network update. Several algorithms of the literature [10, 12, 23] have no or very short preprocessing times and are well adapted to frequent network updates. It is not the case of their accelerated versions [9, 22, 24] or of some faster algorithms based on computations of optimal paths, such as Transfer Patterns [3, 5] where the preprocessing time takes 16.5 hours on a Germany network and can obviously not run fast enough. To be able to redo frequently the preprocessing, its duration must be short, at most a few minutes. Note that for some algorithms with large schedule dependent preprocessing, other solutions have been proposed to deal with real-time updates. For instance, in [17], the authors make Transfer Patterns robust to a chosen set of delays (while outside of the set, the optimality cannot be granted), and in [11], the authors describe a dynamic version of the Public Transit Labeling algorithm that can consider only positive delays.

In this article, we are more particularly focusing on the *Trip-Based Public Transit Routing* algorithm [23] (TB). This algorithm is based on a graph representation of the network where the nodes correspond to trips, i.e. a vehicle following a certain schedule, while the arcs correspond to transfers between the trips, i.e to a user alighting at a stop of the origin trip of the transfer, walking to a stop of the destination trip and boarding this destination trip. In order to obtain a search graph, the information in the timetables is preprocessed and the set of all the possible transfers is pruned to make the search more efficient. This algorithm presents a good trade-off between preprocessing time, quantity of auxiliary data and query times. The computation of the search graph is rather light, but can take several minutes for large size networks. In order to make it compatible with more frequent updates of the network, we propose here to accelerate significantly the preprocessing of [23] by adding a new pruning step.

This article is organized as follows. In Section 2, we introduce the necessary notations and describe the preprocessing of the TB algorithm. Then Section 3 presents the new preprocessing step that we propose to reduce the preprocessing time. Experimental results are detailed in Section 4. Section 5 summarizes our contribution and suggests possible extensions.

2 Preprocessing of the Trip-Based Public Transit Routing algorithm

The preprocessing step of the TB algorithm consists in the generation of the search graph from the timetable and transfer time information. This section will state the necessary notations and will explain the principles of this preprocessing.

2.1 Notations and search graph structure

In order to make it easier for the reader, we use notations similar to that of [23].

A *trip* t represents a vehicle, which, following a sequence $\overrightarrow{p}(t) = \langle p_t^1, p_t^2, \dots \rangle$ of public transit stops, arrives at stop p_t^i at time $\tau_{arr}(t, i)$ and departs from it at time $\tau_{dep}(t, i)$. When several trips share the same sequence and *if they do not overtake each other*, they can be grouped into a *line*, which is a set of trips ordered according to the relations $\leq$ and $<$ defined by:

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

when the trips u and t have the same sequence.

The sequence of stops of a line L is denoted $\overrightarrow{p}(L) = \langle p_L^1, p_L^2, \dots \rangle$, similarly as that of its trips. For two stops p_t^i and p_t^j with $i < j$ of the stop sequence $\overrightarrow{p}(t) = \langle p_t^1, p_t^2, \dots \rangle$ of trip t, we denote by $p_t^i \to p_t^j$ the trip segment of t between stops p_t^i and p_t^j. This notation refers to boarding the trip t at its i^{th} stop and alighting it at its j^{th} stop. A *connection* between two stops is a trip segment $p_t^i \to p_t^j$ where $j = i + 1$.

In the search graph, each trip is represented by a node while the arcs represent transfers between trips. A transfer between the stop p_t^i of trip t and the stop p_u^j of trip u is denoted $p_t^i \to p_u^j$ and has a transfer duration $\Delta\tau_{fp}(p_t^i, p_u^j)$, where $\Delta\tau_{fp}(p, q)$ is the duration of the walking itinerary between stop p and stop q. This transfer is *feasible* if it is possible to alight trip t at stop p_t^i at arrival time $\tau_{arr}(t, i)$ and reach stop p_u^j of trip u before departure time $\tau_{dep}(u, j)$, that is if $\tau_{arr}(t, i) + \Delta\tau_{fp}(p_t^i, p_u^j) \leq \tau_{dep}(u, j)$. When transferring at a given stop, it is possible to consider a positive *change time* $\Delta\tau_{fp}(p, p)$, necessary to move within the station p.

In the search graph, an arc between a trip t and a trip u represents a given feasible transfer $p_t^i \to p_u^j$. If several transfers are feasible between the two trips, it is possible to have multiple arcs between the corresponding nodes, as on Figure 1. The left part of the figure represents two trips, t and t'. The dashed lines represent some possible foot paths between the stations of the two trips such that the corresponding transfers are feasible. In the resulting search graph on the right, each transfer is represented by an arc.

2.2 Preprocessing

The main idea of the preprocessing proposed in [23] is to first generate feasible transfers from the set of walking paths defined by $\Delta\tau_{fp}$. If there is a path between the stops of two different lines, then transfers will be possible between those two lines at those stops. As

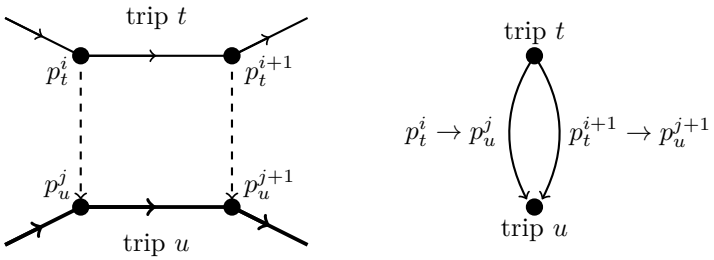

Figure 1 From the public transit data (represented on the left) to the trip-based graph (represented on the right).

arrival time is optimized, when transferring from one stop of an origin trip to a given stop of a destination line, it is possible to consider only the earliest trip such that the transfer is feasible (when it exists). Indeed, the relation $\leq$ between the trips of the destination line implies that such an earliest trip is well defined when there is at least one feasible transfer.

A large set of transfers is obtained when considering only the earliest feasible destination trip for each origin trip and each walking path. Among those transfers, many are not necessary when searching for a complete set of solutions for earliest arrival time and minimum number of transfers. The initial transfer set is hence pruned to reduce its size, but in such a way that it remains *correct* at the end of the pruning, i.e. in such a way that it is still possible to compute a complete solution set for earliest arrival time and minimum number of transfers using the search graph obtained based on the reduced transfer set.

This pruning is performed for each origin trip, removing transfers from its stations. It can hence be easily parallelized, processing different trips on different threads. The proposed pruning consists in two stages: first, removing so called *U-turn transfers*, i.e. transfers $p_t^i \to p_u^j$ such that $p_u^{j+1} = p_t^{i-1}$ if they cannot lead to improved arrival times; second, removing transfers if they cannot improve arrival times at stops compared to arrival times at the same stops considering previously checked feasible transfers. The idea is to start from the last stop of the origin trip t and to move backward along it to check the transfers from the current stop in decreasing stop sequence order. The minimum arrival time at this current stop p_t^i is updated with trip t's arrival time. Then, arrival times and change times are updated at all the stations that can be reached from p_t^i. Then for each transfer $p_t^i \to p_j^u$, the minimum arrival times and change times are updated at the stops of the destination trip that are after p_j^u in the stop sequence of trip u. Finally, foot transfers are also performed from those stops to all reachable stops in order to try and improve their arrival times and change times. If a transfer improves the arrival time or change time at any stop, it will be kept. Otherwise, it is removed. The pseudo-code of the transfer set reduction step can be found in Algorithm 3 of [23]. This pruning removes a large part of the transfers initially present in the set (9 out of 10 on a Germany network and 8 out of 10 on a London network in [23]), which speeds the search phase up by a factor 3.

While this preprocessing is fast enough for not so frequent real-time updates on many networks, it can become too slow for larger graphs where the number of trips is important, for very dense networks where a lot of stops are close to one another or for more frequent updates.

3 Speeding the preprocessing up

In order to speed-up the preprocessing, we propose to add a first reduction step, based this time on the line structure of the public transit network.

For each line L, first compute all the lines L' that can be reached by transfer, that is such that

$$\exists (i,j) \in [1 \dots |\overrightarrow{p}(L)| - 1] \times [0 \dots |\overrightarrow{p}(L')| - 2], \quad \Delta\tau_{fp}(p^i_L, p^j_{L'}) \text{ is defined}$$

Note that you do not transfer from the first stop of a trip or to the last stop of a trip. When $\Delta\tau_{fp}(p^i_L, p^j_{L'})$ is defined, $(i, L', j, \Delta\tau_{fp}(p^i_L, p^j_{L'}))$ is added to the set $\mathcal{T}(L)$ of possible transfers for L.

First, u-turn transfers $(i, L', j, \Delta\tau_{fp}(p^i_L, p^j_{L'}))$ of line L where $p^{j+1}_{L'} = p^{i-1}_L$, can be removed if $\Delta\tau_{fp}(p^{i-1}_L, p^{j+1}_{L'}) \leq \Delta\tau_{fp}(p^i_L, p^j_{L'})$.

After that first step, the trips of L can be processed in such a way that we compare transfers to trips of the same line L' with later transfers (i.e. transfers leaving line L later or at the same stop). Hence, the transfers of $\mathcal{T}(L)$ are sorted first by destination line. Then, as for the arrival and change time based preprocessing, the transfers (i, L', j, Δ) of the line L are sorted first by decreasing origin index i, and then by increasing destination index j.

Instead of comparing arrival times and change times at stops for all the transfers of one trip, we consider only one destination line at a time and we make a simpler and faster comparison: we only compare the trips that can be boarded at the stops of the destination line, using relation $\leq$. For this, it is sufficient to check the earliest trip so far passing at the destination stop index and compare to the index of the current destination trip. The transfers are pruned based on the absence of update at any stop. We denote by $\mathcal{T}(t, L')$ the resulting set of transfers between trip t and destination line L'. The union of all the sets $\mathcal{T}(t, L')$ is the set $\mathcal{T}$ of transfers which is returned at the end of the line-based pruning step.

Algorithm 1 describes the complete method. We call it *line-based pruning* and we denote it LB for short. Note that this algorithm can be trivially parallelized as each origin trip is processed separately.

After this transfer set building part, the arrival and change time based pruning might be applied in order to reduced further the set. The resulting search graph keeps the optimality of the search phase.

▶ **Proposition 1.** *Algorithm 1 computes a correct set $\mathcal{T}$ of transfers for earliest arrival time and minimum number of transfers.*

Proof. Consider an optimal solution s with at least one transfer, that we define by the trip segment sequence that composes it:
$s = \left\langle p^{j_1}_{t_1} \to p^{i_1}_{t_1}, p^{j_2}_{t_2} \to p^{i_2}_{t_2} \dots, p^{j_{k+1}}_{t_{k+1}} \to p^{i_{k+1}}_{t_{k+1}} \right\rangle$
We denote by L_1, L_2, ..., L_{k+1} the lines of the trips t_1, t_2, ..., t_{k+1} respectively. We need to prove that it is possible to construct at least one solution with the same value those transfers are all in $\mathcal{T}$.

Consider the first transfer $p^{i_1}_{t_1} \to p^{j_2}_{t_2}$ of s. If t_2 is not the earliest trip of L_2 such that the transfer from t_1 at $p^{i_1}_{t_1}$ to L_2 at $p^{i_2}_{L_2}$ is feasible, we can replace it with a transfer to the earliest trip such that the transfer is feasible. Now, we suppose that it is the case. There are two possibility, either $p^{i_1}_{t_1} \to p^{j_2}_{t_2}$ is in the transfer set $\mathcal{T}(t_1, L_2)$ or it has been pruned.

In the case where the transfer has been pruned, there exists a transfer $p^i_{t_1} \to p^j_t$ of $\mathcal{T}(t_1, L_2)$ such that $i \geq i_1$, $j \leq j_2$ and $t \leq t_2$.

Algorithm 1 Transfer set building with line-based pruning.

Input: Timetable data, transfer duration data
Output: Reduced transfer set $\mathcal{T}$
$\mathcal{T} \leftarrow \emptyset$
for each line L **do**
  $\mathcal{T}(L) \leftarrow LINE\_TRANSFERS(L, \text{transfer duration data})$
  for each trip t of L **do**
    $T \leftarrow \emptyset$ ▷ Transfer set for each target line
    $L_{prev} \leftarrow$ **null**
    for each transfer (i, L', j, Δ) of $\mathcal{T}(L)$ **do**
      if $L_{prev} \neq L'$ **then**
        $\mathcal{T} \leftarrow \mathcal{T} \cup T$
        $T \leftarrow \emptyset$, $L_{prev} = L'$
        $R(.) \leftarrow \infty$ ▷ Earliest destination trip at index j
      end if
      $t' \leftarrow$ earliest trip of L' at j such that $\tau_{dep}(t', j) \geq \tau_{arr}(t, i) + \Delta$
      if $T = \emptyset$ **then**
        $T \leftarrow \{p_t^i \rightarrow p_{t'}^j\}$
        for each index $j' \in [j \dots |\overrightarrow{p}(L')| - 1]$ **do**
          $R(j') \leftarrow t'$
        end for
      else
        if $t' < R(j)$ **then**
          $T \leftarrow T \cup \{p_t^i \rightarrow p_{t'}^j\}$
          for each index $j' \in [j \dots |\overrightarrow{p}(L')| - 1]$ **do**
            $R(j') \leftarrow \min\{t', R(j')\}$
          end for
        end if
      end if
    end for
    $\mathcal{T} \leftarrow \mathcal{T} \cup T$
  end for
end for
return $\mathcal{T}$
procedure LINE_TRANSFERS(line L, transfer duration data)
  $T \leftarrow \emptyset$ ▷ Builds the line neighborhood
  for $i \leftarrow |\overrightarrow{p}(L)| - 1, \dots, 1$ **do**
    for each stop q such that $\Delta\tau_{fp}(p_L^i, q)$ is defined **do**
      for each (L', j) such that $q = p_{L'}^j$ **do**
        $T \leftarrow T \cup \left\{(i, L', j, \Delta\tau_{fp}(p_L^i, p_{L'}^j))\right\}$
      end for
    end for
  end for
  Sort T first by target line, then by decreasing origin line index, then by increasing target line index, then by chosen sorting
  return T
end procedure

Table 1 Data sets used for the experiments.

	stops	trips	lines	foot paths	connections
NL	48 694	332 164	2 773	439 129	6 144 380
IDF	42 325	319 151	1 869	846 246	7 031 782
Korea	180 948	446 741	31 708	4 195 659	22 346 975

Table 2 Preprocessing for the NL network.

Preprocessing	Nb kept transfers (M)	Nb removed transfers (M)	Graph size (MB)	Mean proc. time (s)	Mean save to file time (s)
No pruning	246.17	0	4 620	32.9	51.5
LB	96.30	149.87	1 782	15.0	18.0
LB & Witt [23]	35.44	210.73	650	18.8	7.1
Witt [23]	35.20	210.97	645	25.8	6.8

In solution s, we replace $p^{j_1}_{t_1} \to p^{i_1}_{t_1}$ by $p^{j_1}_{t_1} \to p^{i}_{t_1}$, and $p^{j_2}_{t_2} \to p^{i_2}_{t_2}$ by $p^{j}_{t} \to p^{i_2}_{t}$ to obtain a new solution s'. Note that transfer $p^{i_2}_{t} \to p^{j_3}_{t_3}$ is feasible, since transfer $p^{i_2}_{t_2} \to p^{j_3}_{t_3}$ was feasible and $t \leq t_2$.

The new solution has the same arrival time, and the same number of transfers as the solution s, they are hence both optimal with the same value.

We can proceed iteratively with the next transfers to replace all the transfers that do not belong to $\mathcal{T}$ by transfers belonging to $\mathcal{T}$. We hence obtain an optimal solution equivalent to s such that its transfers are all in $\mathcal{T}$. ◀

4 Experiments

We perform our experiments on a 64 threads (4 sockets, 8 cores, 2 threads per core) 2.7 GHz Intel(R) Xeon(R) CPU E5-4650 server with 20 MB of L3 cache and 504 GB of RAM. We use 3 data sets of large size, two of which are public.

The first data set is open and provided by Ovapi [19]. It contains public transit information for Netherlands and we denote it *NL*. The *IDF* data set is provided by Île-de-France Mobilités [6] with permissive license, and covers the Île-de-France area in France. This data set has been used in several previous publications, but note that the version used here might be different from the one cited due to regular updates. The third data set is a proprietary data set used in Naver Map [18] that covers the whole Korea. Table 1 summarizes the main figures relative to the size of those data sets. Note that the footpaths are a mixture of the provided transfer information (if any) and generated footpaths. The TB algorithm does not require closure of the footpaths (as opposed to the initial version of RAPTOR [10] or to CSA [12]) but we choose to generate additional footpaths as users are often willing to walk between stations if the distance is limited. We set this bound to 600 m and set the walking speed to 3.6 kph. Footpaths between any two stops such that their distance via the road network is lower than 600 m are added to the public transit network. Note that the resulting data sets are hence significantly larger than the ones in [23] in terms of number of footpaths (the Germany network has only 100K footpaths), which impacts the computation times of the preprocessing and search phases.

Tables 2, 3 and 4 give the preprocessing times for 4 versions of the search graph generation step. We indicate for each version the number of transfers kept at the end of the preprocessing, the number of transfers removed, the size in memory of the search graph obtained, the mean

Table 3 Preprocessing for the IDF network.

Preprocessing	Nb kept transfers (M)	Nb removed transfers (M)	Graph size (MB)	Mean proc. time (s)	Mean save to file time (s)
No pruning	876.27	0	16 782	115.1	197.3
LB	201.13	675.15	3 833	36.3	44.1
LB and Witt [23]	81.27	795.00	1 542	40.2	14.6
Witt [23]	86.78	789.49	1 650	99.6	18.7

Table 4 Preprocessing for the Korean network.

Preprocessing	Nb kept transfers (M)	Nb removed transfers (M)	Graph size (MB)	Mean proc. time (s)	Mean save to file time (s)
No pruning	2 795.59	0	53 339	307.6	686.1
LB	631.60	2 163.99	11 943	91.4	148.0
LB and Witt [23]	228.07	2 567.52	4 290	183.0	41.9
Witt [23]	234.76	2 560.83	4 410	448.1	47.7

processing time, and the time necessary to save the computed graph to file. Of course, the saving step is not mandatory, as the search graph can be used directly once computed in an end-to-end application. However, in a real-time update context, the preprocessing could be performed by a preprocessing service while the query service is running, possibly on a different server. We hence choose to indicate our mean save to file time as well, as it might be relevant for practical applications.

The first preprocessing version that we test performs no pruning. It corresponds to the transfer generation step of [23] and only computes the earliest possible trip for a transfer from a trip at an origin index to a destination line at a destination index to be possible and save the obtained graph structure for it to be loaded and used by the query server. As so many transfers are generated and put into RAM, the computation time and the time necessary to save them to file are significant. Indeed, the size of the graph is 4.6 GB for the NL network, 16.8 GB for the IDF network and 53.3 GB for the larger Korean network. It is hence time consuming to generate and save.

In the second version of the preprocessing, only the line-based pruning (LB) is applied. We can see that the number of arcs in the search graph is already significantly reduced, as it is divided by 2.55 for the NL data set, by 4.36 for the IDF data set and by 4.43 for the Korean network. As a result, the total preprocessing time is also reduced significantly compared to the no pruning version.

In the third version, the initial arrival and change time-based pruning is applied to the reduced transfer set obtained after line-based pruning. This additional step increases the preprocessing time for the larger Korean data set (it is nearly twice as long), but reduces significantly the time necessary to save the search graph to file at the end of the process.

The last version is the initial preprocessing proposed by Witt [23], those figures are provided in order to compare with the proposed method. Note that to make the comparison more meaningful, we have slightly modified it in order to make it more efficient: instead of generating all the transfers and then applying the reduction steps, we generate transfers for each trip separately and prune them on the flight. This avoids saving all transfers into memory at once, as in the no-pruning version, since the non-useful transfers will be removed online. In the original article, it is proposed to first compute all the transfers and then

Table 5 Bi-criteria queries for different levels of pruning for the NL network.

Preprocessing	Graph size (MB)	Mean nb of solutions	Mean duration (ms)	Mean queue size (k)
No pruning	4 620	1.60	150	96.53
LB	1 782	1.60	103	49.97
LB and Witt [23]	650	1.60	63	35.55
Witt [23]	645	1.60	66	35.16

Table 6 Bi-criteria queries for different levels of pruning for the IDF network.

Preprocessing	Graph size (MB)	Mean nb of solutions	Mean duration (ms)	Mean queue size (k)
No pruning	16 782	1.48	330	68.16
LB	3 833	1.48	113	37.77
LB and Witt [23]	1 542	1.48	79	29.65
Witt [23]	1 650	1.48	86	30.46

prune them, but as the no pruning version of the preprocessing indicates, this would make the preprocessing step significantly and unnecessarily longer. It might be what the author suggested in [23] by proposing to "merge the two steps" at Section 3.1.

The third and forth versions give similar results in terms of number of transfers in the search graph. It is expected as the order of the transfers to check for each trip is similar in both cases in our implementation. However, the complexity of the pruning step proposed in [23] is linearly dependent of the initial number of transfers to which it is applied. Hence, applying it on a reduced set leads to improved computation times. For the NL network, the processing time part of the preprocessing is divided by 1.4, that of the IDF network by 2.48 and that of the Korean network by 2.45. With the proposed improvement, the total preprocessing is below 4 minutes for the largest data set, which makes it suitable for frequent real-time updates.

We then tested the different search graphs with an implementation of the standard TB algorithm in order to observe the impact on query times. We performed bi-criteria earliest arrival time queries between randomly chosen stops of the public transit networks. The corresponding solutions are computed along with the Pareto front values. For each network, we generated 100 such queries.

As is shown in Table 5, Table 6 and Table 7, the query times are very similar for the last two versions of the preprocessing. It is of course linked to the fact that the obtained search graphs are very similar.

Table 7 Bi-criteria queries for different levels of pruning for the Korean network.

Preprocessing	Graph size (MB)	Mean nb of solutions	Mean duration (ms)	Mean queue size (k)
No pruning	53 339	1.73	2 211	173.87
LB	11 943	1.73	642	90.41
LB and Witt [23]	4 290	1.73	399	75.68
Witt [23]	4 410	1.73	332	76.93

However, it is interesting to see that the line-based pruning on its own already provides a significant reduction of the query times. Indeed, compared to the no pruning case, computation times are divided by 1.5 for Netherlands, by 2.9 for Île-De-France and by 3.4 for Korea. It could hence be an alternative in the case where the search graph doesn't need to be saved to file at the end of the preprocessing. Indeed, in some cases, trading slower queries for lesser preprocessing time might be useful in practice, for instance to allow more frequent network updates.

5 Conclusion and perspective

In this article, we propose an additional preprocessing step for the Trip-Based Public Transit Routing algorithm. This pruning step reduces significantly the total preprocessing time, while keeping the optimality of the search phase. It has been tested on 3 data sets of different sizes and reduces their preprocessing time by a factor 2.5 on the two largest and 1.4 on the smallest one compared to an improved version of the initial pruning.

Reducing the preprocessing time of routing algorithms is particularly relevant in real-time network update contexts, but also when adding additional features to the algorithm, such as the mode customization described in [16], that increases the preprocessing time. This reduction step could hence allow for integrating new constraints or customization of the search phase in the algorithm while keeping the preprocessing times compatible with real-time updates on some large networks.

As a perspective, in a context where constraints or search customization would make the preprocessing time too long for real-time updates, adapting the preprocessing to make it compatible with dynamic changes of the networks could be considered.

References

1 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, 2009. doi:10.1007/978-3-642-03456-5_24.

2 Hannah Bast, Mirko Brodesser, and Sabine Storandt. Result Diversity for Multi-Modal Route Planning. In Daniele Frigioni and Sebastian Stiller, editors, *ATMOS - 13th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems - 2013*, volume 33 of *OpenAccess Series in Informatics (OASIcs)*, pages 123–136, Sophia Antipolis, France, September 2013. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/OASIcs.ATMOS.2013.123.

3 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.

4 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*, volume 9220 of *Lecture Notes in Computer Science*, pages 19–80. Springer, Cham, 2016. doi:10.1007/978-3-319-49487-6_2.

5 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.

6 Île de France Mobilités. Open data. https://opendata.stif.info.

7 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.

8 Daniel Delling, Julian Dibbelt, Thomas Pajor, and Renato F. Werneck. Public Transit Labeling. In Evripidis Bampis, editor, *Experimental Algorithms. SEA 2015.*, Lecture Notes in Computer Science, pages 273–285. Springer, Cham, 2015. doi:10.1007/978-3-319-20086-6_21.

9 Daniel Delling, Julian Dibbelt, Thomas Pajor, and Tobias Zündorf. Faster Transit Routing by Hyper Partitioning. In Gianlorenzo D'Angelo and Twan Dollevoet, editors, *17th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2017)*, volume 59 of *OpenAccess Series in Informatics (OASIcs)*, pages 8:1–8:14, Dagstuhl, Germany, 2017. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/OASIcs.ATMOS.2017.8.

10 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'12)*, pages 130–140, 2012. doi:10.1137/1.9781611972924.13.

11 Mattia D'Emidio and Imran Khan. Dynamic Public Transit Labeling. In Sanjay Misra, Osvaldo Gervasi, Beniamino Murgante, Elena N. Stankova, Vladimir Korkhov, Carmelo Maria Torre, Ana Maria A. C. Rocha, David Taniar, Bernady O. Apduhan, and Eufemia Tarantino, editors, *Computational Science and Its Applications - ICCSA 2019 - 19th International Conference, Saint Petersburg, Russia, July 1-4, 2019, Proceedings, Part I*, volume 11619 of *Lecture Notes in Computer Science*, pages 103–117. Springer, 2019. doi:10.1007/978-3-030-24289-3_9.

12 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.

13 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.

14 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.

15 Dominik Kirchler, Leo Liberti, and Roberto Wolfler Calvo. Efficient Computation of Shortest Paths in Time-Dependent Multi-Modal Networks. *Journal of Experimental Algorithmics*, 19:2.5:1–2.5:29, 2014. doi:10.1145/2670126.

16 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.

17 Thomas Liebig, Sebastian Peter, Maciej Grzenda, and Konstanty Junosza-Szaniawski. Dynamic Transfer Patterns for Fast Multi-modal Route Planning. In Arnold Bregt, Tapani Sarjakoski, Ron van Lammeren, and Frans Rip, editors, *Societal Geo-innovation*, pages 223–236, Cham, 2017. Springer International Publishing. doi:10.1007/978-3-319-56759-4_13.

18 Nave Map. https://map.naver.com/v5/.

19 OVapi. http://gtfs.ovapi.nl/.

20 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), 2008. doi:10.1145/1227161.1227166.

21 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. doi:10.1002/net.21815.

22 Ben Strasser and Dorothea Wagner. Connection Scan Accelerated. *2014 Proceedings of the Meeting on Algorithm Engineering and Experiments (ALENEX)*, pages 125–137, 2014. doi:10.1137/1.9781611973198.12.

23 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.

24 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 1–12, Dagstuhl, Germany, 2016. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/OASIcs.ATMOS.2016.10.