A k-shortest paths based algorithm for multimodal time-dependent networks to compute alternative routes

Grégoire Scano, Marie-José Huguet, Sandra Ulrich Ngueveu

A k-shortest paths based algorithm for multimodal time-dependent networks to compute alternative routes

Grégoire Scano[1,2,4], Marie-José Huguet[1,2], Sandra Ulrich Ngueveu[1,3]

[1] Université de Toulouse, LAAS-CNRS, F-31400 Toulouse, France
[2] Université de Toulouse, INSA, LAAS, F-31062 Toulouse, France
[3] Univ de Toulouse, INP, LAAS, F-31400 Toulouse, France
[4] MobiGIS, F-31330 Grenade, France

Abstract. Usual computations of alternative routes and the measure of their similarities and/or differences do not embrace the variability of networks specifics and user preferences. Indeed, the definition and evaluation of the difference between paths is often embedded into algorithm internals and thus does not take into account that similar or dissimilar paths may vary depending on the user and/or the network. In this article, a generic method to generate alternative routes on FIFO time-dependent multimodal graphs with regular language constraints is presented. It relies on the computation, in a first stage, of the k-shortest paths before the enforcement of a distinction criteria based on word metrics and comparison procedures in a second stage. We first present a variant of a k-shortest paths algorithm taking into account both the multimodality and the time-dependency inherent to transportation networks. Then, we propose several methods for evaluating the differences between routes. Experiments are conducted in realistic cases of transportation networks and associated results show the relative efficiency and interest of the approach.

Keywords: k-shortest paths, multimodal time-dependent networks, alternative routes

1 Introduction

Users of transportation networks can be interested not only in the shortest path between the origin and the destination of their journey but also in alternative routes with respect to their personal preferences. In addition, if a relevant set of solutions can be presented instead of a single solution, it underlines the quality of the network, allowing to travel from place to place with different means, and thus can be used by transit authorities to monitor the consistency of the transport offer. However, handing out multiple routes supposes to be able to remove many almost identical propositions so as not to overwhelm the user and conceal or miss potentially relevant paths. It is important to select only relevant candidates for the user to skim through, which means selecting the best yet dissimilar

paths according to the user's own priorities.
Existing methods to compute alternative paths, such as plateau and penalty, have drawbacks because either they are based on a bidirectional search with optimal cost which is not efficient for time dependent networks and/or they assert the difference between two paths within the algorithm which implies an a priori knowledge.
In this paper, we propose a generic approach for generating alternative routes in time-dependent and multimodal transportation networks. We assume that the definition of the difference between two paths can vary depending on the network and/or the user. The proposed approach is decomposed in two stages. The first one relies on the computation of the k-shortest paths whereas the second one relies on filtering based on word metrics and comparison procedures. The combination of both produces alternative routes different enough according to a difference criterion and a threshold, while guaranteeing minimal costs.
The challenge is to compute alternatives while keeping the number of elements limited though as dissimilar as requested. We assume that the shortest path has to be presented because it carries the nominal cost as a reference for comparison to other solutions. Computationally, algorithms should run in polynomial time and with user tuning capabilities to compute alternatives.
To prove our point, we first define the problem under study and notations used throughout the article in section 2. Then, in section 3 we describe existing solutions. After presenting the proposed method in section 4, we focus on the first stage, computing the k-shortest paths in section 5 and then move, in section 6, to the selection of alternatives among those paths using word metrics. Finally, computational experiments validate the proof of concept in section 7.

2 Definitions and Problem Statement

Considering a finite set of vertices V, a finite set of edges E and a finite set of modes M, a multimodal transportation network is modeled with an edge-labeled graph $G = (V, E, \Sigma)$ where each edge is denoted (i, j, m) with $i \in V, j \in V$ and $m \in M$. A path $\bar{\pi}$ is a sequence of consecutive edges and π is the associated sequence of nodes. A path π is simple (resp. elementary) iff it contains at most once any arc (resp. vertex). The i^{th} vertex (resp. edge) in π (resp. $\bar{\pi}$) is denoted by $\pi[i]$ (resp. $\bar{\pi}[i]$) and $\pi(u, v)$ denotes the path from node u to node v. A cost function f_{uvm} is associated to each labeled edge (u, v, m) to represent the travel time. These costs may be static or time-dependent. In the latter case, $f_{uvm}(t)$ gives the travel time from u to v in mode m when leaving u at time t.

In a monomodal transportation networks, alternative routes are usually characterized in terms of number of separate nodes (or arcs). But in the context of multimodal transportation networks, two different routes in terms of nodes (or arcs) but using the same transportation mode for example, may be considered as similar by a user and different by another. As an illustration, let us consider the transportation network depicted in figure 1 which represents a simple multi-

modal graph with static cost containing six modes, *pedestrian* (p), *subway* (s_1, s_2) and *bus* (b_1, b_2), and spread over three "geographic" zones, *north* (n) in red, *center* (c) in green and *south* (s) in blue.

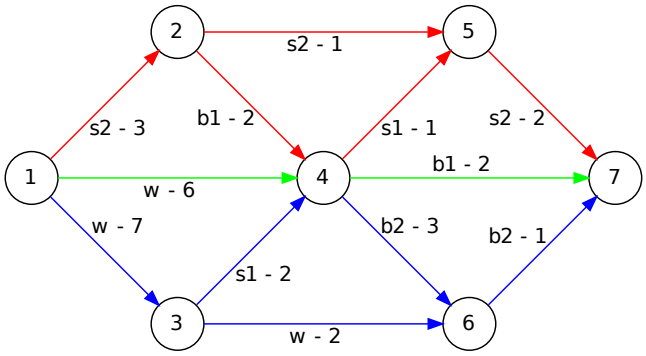

Fig. 1. Example graph

We consider three users A, B and C interested in alternative routes from node x_1 to node x_7, each having personal preferences on alternative itineraries. User A characterizes his itineraries with the sequence of transportation line used. In this case, two routes using for example the same subways in the same order will be considered similar even if the transfer stations are different. User B only focuses on the transportation mean utilized, regardless of the order or the specific line. In his case, taking a bus then a metro is equivalent to taking a metro then a bus, and the specific line of metro or bus used does not matter. Finally, user C is only concerned with the regions or districts traversed during the journey. We assume that all users are interested in paths with a level of difference of at least 1 with regards to their preferences. In this case, paths (1-2-4-7) and (1-2-4-5-7) are considered different by user A, but similar by users B and C. Likewise, paths (1-4-7) and (1-3-6-7) are considered similar by user B, but different by users A and C.

In this context, we developed a generic method able to adapt easily to various users' definitions of what constitute similar or different paths. The goal was to propose alternative paths to users in increasing order of cost, whilst ensuring that each new path was different enough, with regards to the user's own preferences, from the previous selected paths. The resulting methodology is detailed in Section 4.

3 Related works

Given a weighted graph $G = (V, E)$, an origin node o and a destination node d, the Shortest Path Problem (SPP) from o to d is solved in polynomial time with the well-known DIJKSTRA algorithm. In this algorithm, a label is associated to each node, each label containing the current shortest path from the origin to the corresponding node. Two main speed-up techniques were introduced to improve the efficiency of this algorithm, A* and bidirectional. In the A^* goal directed search, the DIJKSTRA algorithm is guided towards the destination using an estimate cost between the current node and the destination d. Obtaining the optimal solution at the end of such algorithm is guaranteed if the estimation is a lower bound of the exact cost. In a bidirectional algorithm, two algorithms start: one running from o to d (forward search) and the other one from d to o in the reverse graph (backward search). When a connection is found between the forward and the backward algorithms, a feasible solution is obtained. However, this solution may not be optimal and the two algorithms keep running until there is no better solution connecting the forward and the backward labels.

Some extensions of the SPP were proposed to deal with the time-dependency of travel times. It has been shown [8] that the resolution of the SPP with such a cost is polynomial iff the function is increasing every time an edge is added to the current path. The graph is then said to have the FIFO property because given any initial cost, a shortest path starting with a greater cost will have a greater final cost than any other path starting with a lower cost from the same departure node and arriving at the same final node. However, many efficient techniques based on bidirectional search cannot be easily extended in the time-dependent case as the exact starting time is only given at the origin.

In transportation networks, the sequence of modes corresponding to a path can be restricted to a language in order to match user constraints. The *regular language constrained shortest path problem* can be solved in polynomial time [4] using the D_{RegLC} algorithm presented in [3] that is an extension of the DIJKSTRA algorithm on the product graph of G and the automaton accepting L. Regarding bidirectional search and assuming that the automaton is deterministic, bidirectional methods despite correctness, may be exponentially complex if the reverse automaton is non deterministic since crossing an edge from a given state may result into several non dominated states.

To the best of our knowledge, three main methods were proposed to consider alternative routes and rely on variants of the SPP.

The first method considers multiple objectives and aims to determine Pareto solutions. Instead of having one cost to optimize, a vector of costs is used yielding to a large number of candidate paths. This approach raises two issues related to the formulation of the parameters and the use of the results. Firstly, the objectives are obtained from the user and even if for certain objectives, such as the number of transfers combined with the cost of the path, a dedicated algorithm [1] could have better performances than Martins' [11] algorithm, this is not the case for most combinations of objectives. Secondly, the solution returned form a potentially large set which has to be processed in order to reduce the

number of solutions offered to the user. Therefore the selection process filtering candidates remains to be done, leaving the problem partly unsolved.
The plateau methods ([13],[2]) compute paths using a bidirectional search from both the origin and the destination. Then, the shortest path trees form simple paths at each node they intersect. An edge is in the plateau if the cost of both its source and target on the forward and backward paths tree are equal. As the number of plateaus can be rather important, they have to be filtered by selecting paths maximizing the number of edges in a plateau minus the number of edges in a path. The main drawback of plateaus lies into the impossibility of carrying efficiently the backward search with optimal costs in multimodal and time-dependent context as previously stated.
The penalty method ([2],[9]) consists in computing successive shortest paths, changing the cost of the edges after each iteration to make the previous solution paths less likely to be selected in further explorations. The key point of the procedure is the costs update which has to ensure that potential good solutions will not be dismissed. Various cost adaptation including *penalty-factor*, *multiple-increase* and *rejoin-penalty* can be combined to limit the possibility of skipping a good path lying next to a previously found route. For time-dependency network, the adaptation of traveling cost implies the modification of the whole timetable to keep the FIFO property.

4 Proposed approach for computing alternative routes

The proposed approach comprises two independent stages. The first stage relies on any algorithm that computes k single cost point to point elementary shortest paths, followed, in a second stage, by a selection procedure based on any given metric between paths.
Note that the value k given to the first algorithm is not the same as the k supplied by the user for the whole method. For instance, 3 alternative routes could be based upon either 3 or more paths. From now on, k will denote the number of alternatives and k' the argument of the k shortest paths algorithm.
The property of the paths returned by the first procedure is that the paths are ordered with respect to their cost. When the k^{th} elementary path is computed we know that there are no other elementary path with a lower cost.
First, we select the shortest path, i.e. the nominal route. Then, until k paths have been selected, we compute the difference from the current alternative routes to all other paths and add the path with minimal cost and having a difference higher than the threshold to the solution set.
Such a procedure gives alternative routes depending on the definition of the difference function supplied by the user and minimizing the cost of the paths. In other words, the k^{th} route is the route with minimal cost respecting the threshold difference to the $k-1$ paths.
Note that we could have used the difference measure inside the search but it would have turned the problem into a resource constrained shortest path problem (pseudo-polynomial at best).

The advantages of this method is that both algorithms can be enhanced or changed independently and then plugged back in without much efforts. For instance, speed up techniques using pre-computations to compute the shortest paths could be used to significantly improve the running time. Also, once the shortest paths have been computed it is possible to change the comparison procedure to obtain different results without recomputing the k paths.

In the two following sections, we present a new variant of k-shortest paths algorithms in the context of multimodal and time-dependent network and a method for selecting alternative routes based on usual distance metrics between words.

5 k-shortests paths for multimodal and time-dependent network

5.1 Existing k-shortest path algorithms for monomodal graphs

The k-shortest paths problem consists in computing a given number of non decreasing cost paths between two nodes. Depending on the type of the required paths, either unrestricted or simple, different types of algorithms exist in the literature.
Yen's algorithm [14] computes the k elementary shortest paths. It uses a shortest path algorithm as a subroutine and successively calls it from different origins after discarding from the graph previously used edges. The complexity of the algorithm is $k|V|P(|V|,|E|)$ where $P(|V|,|E|)$ is the complexity of the shortest path algorithm being used. An extension given by Lawler [10] reduces the number of iterations by keeping some information from earlier computations.
Computing k paths containing cycles is easier than simple paths since there is no need to track and prevent the appearance of loops within the paths. The algorithm of Eppstein [5] runs with an excellent complexity of $O(|E|+|V|+k)$. It can be observed that to compute the k^{th} path, only nodes in the $k^{th}-1$ path have to be visited. The REA algorithm [6] uses this fact to compute paths with cycles with a higher complexity of $O(|E|+k|V|log(|V|))$, but the running time in practice is better than that of Eppstein's. However, this idea can be applied to make a lazy version of the former algorithm [7] by delaying the construction of some parts of the intermediary graph. Such adaptation does not change the worst case complexity.
Unfortunately, Eppstein's algorithm cannot be extended efficiently to time-dependent graphs since it uses as a first step the computation of a backward shortest path tree.
Our goal is to compute k elementary shortest paths using algorithms which compute paths, eliminating cycles during and after the search. The rationale is that it might be faster to use algorithms with lower complexities and a little extra work since the graphs considered are quite large.

5.2 Adaptation proposed to multimodal time-dependent graphs

In the following, the computation of the k-shortests paths is always done in the context of a multimodal time-dependent network. Thus, we consider any regular language and the non deterministic automaton that accepts it.
In the case of Yen's algorithm, the extension is straightforward using the D_{RegLC} algorithm as the subprocedure instead of the DIJKSTRA algorithm. For Eppstein's algorithm, the modification is not possible since the algorithm requires the computation of a backward shortest path tree at the beginning. Instead, we adapted the REA algorithm that computes k shortest paths with cycle. For this purpose, we propose in algorithm 1 a forward implementation in place of the recursive calls in order to cut short cycles. Moreover, this procedure is directly adapted for edge-labeled graph $G = (V, E, \Sigma)$ using the product graph $V \times S$ where S is the number of states of the automaton accepting the regular language and, with time-dependent cost function f on each arc respecting the FIFO property.

Indeed, after testing REA, we found that the number of cycles using short cycles (≤ 2) is very large, hiding non cycles solutions after the 10000^{th} first paths on non trivial instances. If continuing the algorithm once a certain number of paths were found to compute more paths is not mandatory, the number of elements kept for each node can be limited to k so as to narrow the search.

In addition, REA allows loops to happen on the origin or the destination but looping through the destination makes no sense since we are interested in the earliest arrival time, and going through the origin multiple times would just shift the departure time. If multiple departure times are to be considered for alternatives, one should launch the computation of the k^{th} shortest paths several times with a predetermined departure time shift not to miss potential solutions.

6 Selection of minimum cost dissimilar paths

The goal of the selection process is to extract a partition of k paths among the precomputed set of elementary paths. This partition contains elements which are as dissimilar as required by the user preference but which have a minimal cost.
The procedure to select the alternative routes is summarized in algorithm 2 and consists in initializing the pool of selected paths with the minimum cost path and then iteratively adding in the pool the minimum cost path that satisfies the difference criterion, with a given threshold, in comparison to the other selected paths of the pool.

Algorithm 1 k^{th} shortest cost from s to t with loops longer than l on G using cost f

Input: $G = (V, E)$, $s \in V$, $t \in V$, $k \in \mathbb{N}^*$, $f : E \times E \to \mathbb{R}_+$, $l \in \mathbb{N}$
Output: (k^*, c^*) cost c^* of the maximum k^* shortest path found

$H \leftarrow \{(s, 0, 0)\}$ ▷ Candidates set
$S \leftarrow V$ ▷ Currently available nodes
$P \leftarrow \emptyset$ ▷ Set of predecessors
$r^* = (0, 0)$ ▷ Default return value
$K[V] \leftarrow 0$ ▷ Number of paths found on nodes
while $H \neq \emptyset \wedge K[t] \neq k$ **do**
    $(u, c, n) \leftarrow min_{H, u \in S}(c)$ ▷ Select best cost candidate which node is available
    $H \leftarrow H \setminus (u, c, n)$
    $S \leftarrow S \setminus \{u\}$ ▷ Make the node unavailable for the moment
    $K[v] \leftarrow K[v] + 1$
    if $u \neq t$ **then**
        for $v \in \{w \in V \setminus \{s\} \mid (u, w) \in G\}$ **do**
            if $v \notin P^l(u, n)$ **then** ▷ Successor is not in the l^{th} predecessors of u
                $H \leftarrow H \cup (v, c + f(u, v), K[v])$
                $P(v, K[v]) \leftarrow (u, n)$
            end if
        end for
    else
        $r^* \leftarrow (c, n)$ ▷ Set new current solution
        $(w, n') \leftarrow (t, n)$
        while $w \neq s$ **do** ▷ Backtrace the path to node s
            $S \leftarrow S \cup \{w\}$ ▷ Make the nodes on the path available
            $(w, n') \leftarrow P(w, n')$
        end while
    end if
end while
return r^*

Algorithm 2 Alternative routes selection among candidate paths

Input: k the number of alternatives, P the set of candidate paths, c the function associating a path to a cost, d the difference function, t the threshold
Output: A the alternative routes

$P' \leftarrow P$ ▷ Initialize the set of candidates
repeat
    $A \leftarrow A \cup \{\arg\min_{p \in P'} c(p)\}$ ▷ Select the best candidate
    $P' = \{p \in P - A \mid \forall s \in A, t \leq d(p, s)\}$ ▷ Update candidates different enough from previously selected elements
until $|A| = k \vee P' = \emptyset$ ▷ k paths were found or no more paths are candidates
return A

To satisfy the requirement of genericity and adaptability to the user preferences, the difference functions were chosen based on the assignment of words to paths, and then the evaluation of the difference between paths as the difference between their respective words. Two word metrics were applied: the edit distance and the occurence of patterns.

Table 1 summarizes all existing paths from x_1 to x_7 on the graph of figure 1, ranked in increasing cost order. For each path, are specified the sequence of nodes visited, the cost, the transportation modes used (which form a basic word), the location of edges used (which form a basic word) and finally the result of the word assignment functions of the three users (A, B, C) that were identified in Section 2. For example, for path $p_1 = (1-2-5-7)$, which is the shortest path, user A will assign the word s_2, corresponding to the only transportation mode used by this path. For user B, who focuses on the transportation mean and not the specific lines, the word assignment function produces letter s if at least one subway is used, p for pedestrian and b for bus. Since the order of usage does not matter, the letters of each word are reordered in lexicographical order to facilitate future comparisons. As a result, the word bs is assigned to path π_2 even though the subway was used before the bus. For user C, only the geographical location of the used edges matter, as a consequence path π_2 is assigned the word nc, resulting from the concatenation of nnc which reflects the geographic location of each arc used, and path π_3 is assigned only n, even though the path uses four different transportation modes.

Table 1. Set of paths

path	nodes	cost	modes	location	A	B	C
π_1	1-2-5-7	6	$s_2s_2s_2$	nnn	s_2	s	n
π_2	1-2-4-7	7	$s_2b_1b_1$	nnc	s_2b_1	bs	nc
π_3	1-2-4-5-7	8	$s_2b_1s_1s_2$	$nnnn$	$s_2b_1s_1s_2$	bs	n
π_4	1-4-7	8	pb_1	cc	pb_1	bp	c
π_5	1-2-4-6-7	9	$s_2b_1b_2b_2$	$nnss$	$s_2b_1b_2$	bs	ns
π_6	1-3-6-7	10	ppb_2	sss	pb_2	bp	s
π_7	1-3-4-7	11	ps_1b_1	ssc	ps_1b_1	bps	sc
π_8	1-3-4-5-7	12	$ps_1s_1s_2$	$ssnn$	ps_1s_2	ps	sn
π_9	1-3-4-6-7	13	$ps_1b_2b_2$	$ssss$	ps_1b_2	bps	s

6.1 Patterns

The evaluation of word metrics using patterns recognition is based on the identification of the number of occurences of certain tuples in the words considered. For instance, considering pairs, each path is associated with all the pairs of consecutive edges it uses. In this case, the word *path* for example will be characterized by the pairs *pa*, *at*, and *th*. When comparing two words, the number of pairs in common are counted and divided by the total number of pairs in both

words. The lower the results, the more dissimilar the words are. Note that this metric is normalized and can only vary between 0 and 1. The complexity of the procedure is $O(N \log N)$ where N is the total length of both words.

To apply this metric taking into account the user's preferences, paths are compared by evaluating the pattern ratio of their respective user words. The user can enforce the level of dissimilarity he wants by setting the threshold value $\overline{\tau_p} \in]0, 1]$, in which case two paths will be considered dissimilar iff the pattern ratio is less or equal to $\overline{\tau_p}$. Note that in the case of pattern recognition based on n-tuples, it is not possible to evaluate words of less than n letters. Consequently, for the pattern recognition based on pairs, it is not possible to evaluate and compare single digit words because the ratio leads to $\frac{0}{0}$, which is undetermined. To counter that, we modify each word by adding a dummy letter (here ϵ) at the beginning and at the end of each user word. For example, user A assigns the word $\epsilon s_2 \epsilon$ to path π_1 and word $\epsilon s_2 b_1 b_2 \epsilon$ to path π_5. As a consequence, the pair pattern ratio between the two words (and thus between the two corresponding paths) is $\frac{2}{6} = \frac{1}{3}$. Similarly, for user C, the pair pattern ratio between paths π_2 and π_5 is $\frac{2}{6} = \frac{1}{3}$ while the pair pattern ratio between paths π_5 and π_8 is $\frac{0}{6}$.

Table 2 summarizes the resulting ordered set of alternative paths obtained on the graph 1 using patterns recognition for each of the users introduced in Section 2 and in function of $\overline{\tau_p}$. Let us focus on user C and threshold value $\tau = \frac{1}{3}$ for

Table 2. Alternative paths using patterns recognition

Threshold $\overline{\tau_p}$	A	B	C
1	all	all	all
1/2	(π_1, π_2, π_4, π_6, π_7, π_8)	(π_1, π_2, π_4)	(π_1, π_2, π_4, π_5, π_6, π_7, π_8)
1/3	(π_1, π_4, π_5, π_6, π_7)	(π_1, π_4)	(π_1, π_4, π_6)
0	(π_1, π_4)	(π_1, π_4)	(π_1, π_4, π_6)

example. The set is initialized with the shortest path π_1. Path π_2 is disregarded because its pair pattern ratio with path π_1 is 0.4, higher than the threshold. Path π_3 is disregarded because it is equivalent to π_1. Path π_4 is selected because the pair pattern ratio with π_1 is 0, lower or equal to $\overline{\tau_p}$. At this stage, the set of selected paths is $\{\pi_1, \pi_4\}$. Path π_5 is disregarded because its pair pattern ratio with path π_1 is 0.4, higher than $\overline{\tau_p}$. Path π_6 is selected because its pair pattern ratio with π_1 is 0 and its pair pattern ratio with π_4 is 0, both values lower or equal to $\overline{\tau_p}$. At this stage, the set of selected paths is $\{\pi_1, \pi_4, \pi_6\}$. Path π_7 is disregarded because its pair pattern ratio with path π_6 is 0.4, higher than $\overline{\tau_p}$. Path π_8 is disregarded because its pair pattern ratio with path π_6 is 0.4, higher than $\overline{\tau_p}$. Finally, π_9 is disregarded because it is equivalent to π_6. Therefore, the set of alternative paths to produce for user C when he sets a threshold value $\overline{\tau_p} = \frac{1}{3}$ is, in increasing cost order, $\{\pi_1, \pi_4, \pi_6\}$.

6.2 Edit distance

The edit distance is a widely spread method and computationally very efficient to measure the similarity of words. It consists in computing the number of atomic operations necessary to transform one word into another. A user might vary the level of dissimilarity enforced by setting the threshold value $\underline{\tau_e} \in \mathbb{N}$, in which case two paths will be considered dissimilar iff the edit distance between their corresponding words exceeds or equals $\underline{\tau_e}$.

Table 3 summarizes the resulting ordered set of alternative paths obtained on the graph 1 using edit distance for each of the users (A, B, C) introduced in Section 2 and in function of $\underline{\tau_e}$.

Table 3. Alternative paths using edit distance

Threshold $\underline{\tau_e}$	A	B	C
0	all	all	all
1	all	(π_1, π_2, π_4, π_7, π_8)	(π_1, π_2, π_4, π_5, π_6, π_7, π_8)
2	(π_1, π_3, π_4, π_5, π_6, π_8)	(π_1, π_4)	(π_1, π_4, π_6)
3	(π_1, π_3, π_4, π_9)	(π_1, π_4)	(π_1, π_7)

Let us focus on user C and threshold value $\tau = 2$ for example. The set is initialized with the shortest path π_1. Path π_2 is disregarded because its edit distance with π_1 is one, less than $\underline{\tau_e}$. Path π_3 is disregarded because it is similar to π_1. Path π_4 is selected because the edit distance to π_1 is two, greater or equal to $\underline{\tau_e}$. At this stage, the set of selected paths is $\{\pi_1, \pi_4\}$. Path π_5 is disregarded because its edit distance to π_1 is one, less than τ. Path π_6 is selected because its edit distance to π_1 is two and its edit distance to π_4 is two, both values greater or equal to $\underline{\tau_e}$. At this stage, the set of selected paths is $\{\pi_1, \pi_4, \pi_6\}$. Path π_7 is disregarded because its edit distance to π_6 is one, less than τ. Path π_8 is disregarded because its edit distance to π_6 is only one. Finally, path π_9 is disregarded because it is similar to π_6. Therefore, the set of alternative paths to produce for user C if when he sets a threshold value $\underline{\tau_e} = 2$ is, in increasing cost order, $\{\pi_1, \pi_4, \pi_6\}$..

Many algorithms exists to compute the edit distance, we use Myers' algorithm [12] with the memory refinement. The algorithm performs in $O(ND)$ using $O(N)$ memory where N is the sum of the lengths and D the edit distance between the two words.

7 Computational evaluation

7.1 Instances, Programming and Parameter Settings

The tests were carried out on an Intel(R) Core(TM) i5-3337U CPU @ 1.7GHz with 6144 KB cache and 3GB main memories and running Linux 3.2.0.4-amd64.

All algorithms were implemented in C++ and compiled with gcc with optimization level 2.

Regarding the instances, we used as transportation network a multimodal graph modeling the city of Toulouse (France). All transportation data used are freely available data : the road network corresponds to the OpenStreetMap datasets and was provided by GeoFabrik and our public transportation network is based on The General Transit Feed Specification format. Once converted into an edge-labeled multimodal graph, it contains 75837 nodes, 484426 road edges and 43318 public transport edges. All combinations of transportation modes are authorized, including pedestrian. A generic automaton that allows bus and subway is used and the departure time is set to 9:AM.

7.2 Evaluation of the proposed k-shortest path algorithm for multimodal and time-dependent graph

In the first part of experiments, we aim to evaluate the efficiency of the proposed k-shortest path algorithm on multimodal and time-dependent networks. We consider a realistic multimodal and time-dependent graph from Toulouse (France), and computations were performed with the value of k' ranging from 100 to 400 with a step of 100. For each value of k', the average computing times were evaluated on 1000 randomly generated pairs origin-destination. The same random generator seed is used to keep the sequence of points unchanged from one instance to another. The results are given in table 4 in which column k' reports the number of paths generated, column k reports the number of elementary paths that were found among the k' paths and column *time* gives the CPU time in milliseconds.

Table 4. Experimental results for the proposed algorithm

k'	100		200		300		400	
cycles cut	time (ms)	k	time (ms)	k	time (ms)	k	time (ms)	k
0	$\lneq 100$	13	$\lneq 100$	20	$\lneq 100$	28	$\lneq 100$	34
1	113	95	134	186	170	276	318	365
2	116	97	140	190	175	283	310	374
3	116	98	141	195	177	291	297	386
4	116	99	147	196	189	293	357	389
5	130	99	173	197	243	295	483	393

We observe that the proposed algorithm is efficient. If the CPU time grows with the length of eliminated cycles, it is however still less than 500 ms in our more complex experiments ($k' = 400$, and length cycle equals to 5).

7.3 Alternative routes

In the second part of experiments, we evaluate the selection of alternative routes. We consider that a set of $k = 491$ shortest paths was previously obtained (with

$k' = 500$ and length cycle reduction equals to 5). We used 4 models to associate paths with words so as to enumerate the alternatives a user may define.

- a Maps an arc to its type as a letter
- b Associates each mode to a letter and writes a letter only once every time a mode is used no matter how long the transportation system is used
- c Same as b but writes the transportation lines instead of the modes

For each model, Table 5 summarizes the average number of alternatives given a threshold varying between 1 to 5 for the edit distance.

Table 5. Experimental results for alternative routes based on the edit distance

threshold	a	b	c
0	490.86	490.86	490.86
1	5.56	1.017	490.86
2	3.01	1.014	433.52
3	2.20	1.010	414.04
4	1.79	1.009	330.45
5	1.57	1.001	324.29

With a threshold of 0, i.e. no filtering, the maximum executing time peaked at 137 ms so execution times for higher thresholds are lower. In addition, for a same threshold of 0, all users have the same number of paths 490 since no filtering is applied. Then, when the threshold increases, we can observe an important deviations between users. User a seems to be the ideal case where a good number of solutions (5.56) are found for a single difference and then slowly decreases as the constraint increases. However, user b cannot make any differences between paths. And user c cannot make a fine statement about which path to pick because it has too much information about the paths.
For patterns based selection, models a and c perform poorly. On the contrary, user b has a relevant set computed in less than 500 ms in average (the CPU time using patterns recognition is higher than the CPU time with edit distance). For a threshold of 2, it has 1.85 and 2.28 paths in average for a pattern length of 2 and 3 respectively. Then, for a difference of 3 or higher it has 1.57 and 1.85. So the longer the pattern is, the more paths are selected.

8 Conclusion

In this paper we proposed an approach based on the computation of a large number of paths to select a few of them as alternatives.
To compute the paths, we adapted one of the best k-shortest paths algorithm to eliminate cycles of a specific length with a good average running time and generating an acceptable number of paths without cycles.
The definition of the difference between two paths, though supposed to be an

operation on some strings, is quite general and may suit specific needs. Experiments revealed that the transformation of a path into a word as well as the algorithm used to measure the difference between words have a great impact on the results, not to mention particularities of the network and associated automaton.
Thus, even if the results are encouraging, any application in a real word environment supposes the ability for the user to specify his criterion and the measure between words he wants to use, which is for the moment to be conceived.

9 Acknowledgments

This work was motivated and supported by the MobiGIS company. We would especially like to thank Sylvain Gaudan and Christophe Lapierre for their help and comments.

References

1. Christian Artigues, Marie-José Huguet, Fallou Gueye, Frédéric Schettini, and Laurent Dezou. State-based accelerations and bidirectional search for bi-objective multimodal shortest paths. *Transportation Research Part C: Emerging Technologies*, 27:233–259, 2013.
2. Roland Bader, Jonathan Dees, Robert Geisberger, and Peter Sanders. Alternative route graphs in road networks. In Alberto Marchetti-Spaccamela and Michael Segal, editors, *Theory and Practice of Algorithms in (Computer) Systems*, volume 6595 of *Lecture Notes in Computer Science*, pages 21–32. Springer Berlin Heidelberg, 2011.
3. Chris Barrett, Keith Bisset, Martin Holzer, Goran Konjevod, Madhav Marathe, and Dorothea Wagner. *Engineering Label-Constrained Shortest-Path Algorithms*, volume 5034 of *AAIM '08*, chapter 5, pages 27–37. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008.
4. Chris Barrett, Riko Jacob, and Madhav Marathe. Formal-language-constrained path problems. *SIAM J. Comput.*, 30(3):809–837, May 2000.
5. David Eppstein. Finding the k shortest paths. *SIAM J. Comput.*, 28(2):652–673, February 1999.
6. Víctor M. Jiménez and Andrés Marzal. Computing the k shortest paths: A new algorithm and an experimental comparison. In *Proceedings of the 3rd International Workshop on Algorithm Engineering*, WAE '99, pages 15–29, London, UK, UK, 1999. Springer-Verlag.
7. Víctor M. Jiménez and Andrés Marzal. A lazy version of eppstein's k shortest paths algorithm. In *In: WEA*, pages 179–190. Springer, 2003.
8. David E. Kaufman and Robert L. Smith. Fastest paths in Time-Dependent networks for intelligent Vehicle-Highway systems application. *Journal of Intelligent Transportation Systems*, 1(1):1–11, 1993.
9. Moritz Kobitzsch, Marcel Radermacher, and Dennis Schieferdecker. Evolution and evaluation of the penalty method for alternative routes. In *In ATMOS*, 2013.
10. Eugene L. Lawler. A procedure for computing the k best solutions to discrete optimization problems and its application to the shortest path problem. *Management Science*, 18(7):401–405, 1972.

11. E. Martins. On a multicriteria shortest path problem. *European Journal of Operational Research*, 16(2):236–245, 1984.
12. Eugene W. Myers. An o(nd) difference algorithm and its variations. *Algorithmica*, 1:251–266, 1986.
13. Andreas Paraskevopoulos and Christos Zaroliagis. Improved Alternative Route Planning. In Daniele Frigioni and Sebastian Stiller, editors, *13th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems*, volume 33 of *OpenAccess Series in Informatics (OASIcs)*, pages 108–122, Dagstuhl, Germany, 2013. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.
14. Jin Y. Yen. Finding the k shortest loopless paths in a network. *Management Science*, 17(11):712–716, July 1971.