Adaptations of k-Shortest Path Algorithms for Transportation Networks

Grégoire Scano, Marie-José Huguet, Sandra Ulrich Ngueveu

Adaptations of k-Shortest Path Algorithms for Transportation Networks

(presented at the 6^{th} IESM Conference, October 2015, Seville, Spain)

Grégoire Scano* † §, Marie-José Huguet* † and Sandra Ulrich Ngueveu* ‡
*CNRS, LAAS, 7 avenue du colonel Roche, F-31400 Toulouse, France
†Univ Toulouse, INSA, LAAS, F-31400 Toulouse, France
‡Univ Toulouse, ENSEEIHT, LAAS, F-31400 Toulouse, France
§MobiGIS, ZAC Grenade Sud, Rue de l'Autan, F-31330 Grenade, France

Email: {gscano, huguet, sungueve}@laas.fr

***Abstract*—The computation of the k-shortest paths, should they be elementary or not, has been extensively investigated in the literature, yielding to extremely performant algorithms. For elementary paths, the best known algorithm to this day is the algorithm of Yen enhanced by the extension of Lawler, while for the search of non-elementary paths, the algorithm with the best complexity is due to Eppstein but is outperformed in practice by the Recursive Enumeration Algorithm. In the context of transportation networks, graphs are time dependent, meaning that the cost of an edge depends on the time at which it is crossed. If for each edge one cannot arrive later if he departs earlier, the network is said to respect the FIFO property. Under this hypothesis, the usual Dijkstra shortest path algorithm is still polynomial. Additionally, since each edge is associated to a transportation type, one may want to restrict a path to be in a regular language. To find a shortest path under this constraint a polynomial algorithm, called DRegLC, works on the product of the network and the graph representing an automaton accepting the regular language.**

In this paper, some k-shortest paths algorithms are adapted to be used on such transportation networks with a regular language constraint. Also, the computation of the k-shortest elementary paths is considered using k-shortest non elementary paths algorithms, deleting loops while searching if possible. To address this approach, a new algorithm is presented to speed-up the search of elementary paths while scanning as few paths containing loops as possible.

I. Introduction and Problem statement

Computing efficiently the shortest path in the context of transportation networks, including multimodality and time-dependency, has been the subject of intensive research in the last ten years. However, 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 a set of viable paths. Moreover, public transport authorities and city planners may be interested by information about the number of paths and associated durations or costs, connecting two points or areas, in order to improve their transport offer.

This can be done by considering k-shortest paths methods producing a set of paths ordered by their costs. The k-shortest paths problem is a well-known problem and it was studied from two points of view depending on the application: producing a set of paths without cycles or producing paths with no restrictions on cycles. The former set is a subset of the later. In the literature, paths with no restriction on cycles are simply denoted paths with cycles. The k-shortest paths algorithms producing paths with cycles have lower worst-case complexities than algorithms producing only cycle-free paths. In the multimodal transportation context, the aim is to produce k-shortest paths from a given origin to a given destination without any cycle and going through the public transportation network and the pedestrian network. Obviously, in transportation applications, users are not interested in paths having cycles. However, due to the difference in algorithms complexity when considering paths with or without cycles and due to the specific topology of transportation networks (with bus lines for instance), both approaches have merits. Of course, as we do not consider a path with cycles as an admissible solution for users, those paths have to be removed from the set of paths generated. Therefore, when considering k-shortest paths with cycles, one would have to produce a high number of paths to obtain the expected number of paths without cycles.

Let $\mathcal{G}(\mathcal{V}, \mathcal{A})$ be a directed graph, where $\mathcal{V}$ is a set of n nodes and $\mathcal{A}$ a set of m arcs and let us consider a static positive cost $c_{i,j}$ on each arc (i, j). A path is defined by a sequence of consecutive arcs (or by the sequence of their corresponding nodes). On such graphs, given an origin node o, and a destination node d, solving the Shortest Path problem consists in finding the path from o to d having the minimal cost and solving the k-Shortest Paths problem consists in finding a set of k paths from o to d having minimal costs if possible: no path outside of this set has a cost lower than any path of the set. The k-Shortest Paths problem on static weighted graphs was already studied in the literature, but the specific case of transportation graphs was not studied.

A Transportation Network is not modeled using static weighted graphs, but it is based on labeled directed graphs $\mathcal{G}_{\mathcal{L}}(\mathcal{V}, \mathcal{A}, \Sigma)$ consisting of a set of n nodes $v \in \mathcal{V}$, a set of m labeled arcs $(i, j, l) \in \mathcal{A} \subseteq \mathcal{V} \times \mathcal{V} \times \Sigma$, and a set of p labels $l \in \Sigma$. Each triplet (i, j, l) represents an arc from node i to node j having label l. The labels on arcs represent transportation modes, such as foot, car, bus, metro, etc. Moreover a positive cost corresponding to the travel time is associated to each arc. Costs may be time-dependent (considering for example timetables or frequency of public transportation lines) and $c_{i,j,l}(t)$ gives the cost of an arc (i, j, l) when arriving at i at time t.

The integration of multimodality through labels or time-dependent costs increases the complexity of Shortest Paths

algorithms because it restricts the application of some efficient techniques such as bidirectional ones. To the best of our knowledge, there is no dedicated algorithm taking into account both multimodality and time-dependency, for solving the k-Shortest Paths Problem in the context of transportation networks.
The rest of the paper is organized as follows. In Section II, we recall the k-Shortest Paths problems previously studied for static weighted directed graphs and the associated algorithms. Then, in Section III, we introduce three algorithms designed for solving the k-Shortest Paths problem in Transportation Networks. Last, in Section IV, we evaluate our algorithms on a real transportation network, before concluding in Section V.

II. Related Work

In a static weighted graph $\mathcal{G}(\mathcal{V}, \mathcal{A})$, given an origin node o and a destination node d, the Shortest Path problem (SP) from o to d is solved in polynomial time with the well-known DISJKSTRA 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 bidirectionnal search.
Hart [1] proposed the A^* algorithm, where a DISJKSTRA algorithm is guided towards the destination informed by 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. Its complexity is the same as of DISJKSTRA if the triangle equality is valid on any node with respect to this estimate.
A bidirectional algorithm, explained by Nicholson, [2] is not more informed than DISJKSTRA but two searches are conducted from origin to destination (forward search) and conversely on the reverse graph (backward search). When a connection is found between the forward and the backward searches, a feasible solution is obtained. Such solution is optimal if and only if its cost is less or equal to the sum of the minimum forward and backward costs of the partial paths that remain unexplored.
Some extensions of the SP problem were proposed to deal with the time-dependency of travel times. It has been shown in [3] that the resolution of the SP problem with such a cost is polynomial iff the arcs in the graph respect the FIFO property (any path starting with a greater cost from a given node 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 to take into account user or mode constraints. The *regular language constrained shortest path problem* can be solved in polynomial time [4] using the D_{RegLC} algorithm presented in [5] that is an extension of the DISJKSTRA algorithm on the product graph of $\mathcal{G}$ (a directed weighted graph) and the automaton accepting the given regular language 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 backward an arc from a given state may result into several non dominated states.

A path in a graph is a sequence of consecutive arcs and will be denoted $\overline{\pi}$, the associated sequence of nodes will be denoted by π. A path $\overline{\pi}$ is said to be simple iff it contains at most once any arc. A path $\overline{\pi}$ is an elementary path iff it is simple and has no cycle, i.e., no node occurs more than once. The i^{th} node (resp. arc) in π (resp. $\overline{\pi}$) is denoted $\pi[i]$ (resp. $\overline{\pi}[i]$) and $\pi(i,j)$ (resp. $\overline{\pi}(i,j)$) denotes the path from node i to node j.

The k-shortest paths ($k-$SP) problem consists in computing a given number of non decreasing cost paths between two nodes. Depending on the type of required paths (elementary or unrestricted), different types of algorithms exist in the literature.
Yen's algorithm [6] 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 $O(k.n.SP(n,m))$ where $SP(n,m)$ is the complexity of the shortest path algorithm being used.
Computing k *paths without any restriction on cycles*, also known as *paths with cycles*, is easier than computing elementary paths since there is no need to track and prevent the occurrence of loops within the paths. The algorithm of Eppstein [7] runs with an excellent complexity of $O(m + n.log(n) + k.log(k))$. 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, which is not possible since the costs of arcs depend on the starting time, and therefore can only be known during a forward exploration.
It can be observed that to compute the k^{th} path, only nodes in the $k^{th} - 1$ path have to be explored. The REA algorithm [8] uses this fact to compute paths with cycles with a higher complexity of $O(m + k.n.log(n))$, but the running time in practice is better than that of Eppstein's. In addition, this idea can be applied to make a lazy version of the former algorithm [9] by delaying the construction of some parts of the intermediary graph. Such adaptation does not change the worst case complexity.

In the context of multimodal and time-dependent graphs (i.e. transportation networks), there is no dedicated k-shortest paths algorithms and there is no comparison of the both types of algorithms (with and without cycles). Due to the specificities of such graphs, we propose to evaluate the efficiency of both approaches (with and without cycles) and to do that we present, in the next Section, two extensions and one new variant of k-shortest paths algorithms.

III. Proposed Algorithms

In the following, the computation of the k-shortest paths is always done in the context of a multimodal time-dependent network. Thus, we consider any regular language and the automaton that accepts it to represent mode constraints (if the automaton is non deterministic, we determine it). Our goal is to

determine k-shortest paths without cycles for a given origin o, a given destination d and a given start time t at the origin. We consider the two types of $k-$SP approaches: with or without cycles. In the case of $k-$SP with cycles, paths that contain cycles are disregarded after the search and some cycles may also be eliminated during the search.

A. Extension for k-shortest paths without cycles

In Yen's algorithm [6], the shortest path is computed first (using DIJKSTRA procedure), then each node of this shortest path is considered to compute another shortest path to the destination. This principle is successively applied on each node of previously obtained paths. When computing k-shortest paths using such method, some nodes and edges need to be removed from the graph (to find different paths) and are ignored by the search by tapping directly into DIJKSTRA internal data structures. The Lawler's extension [10] prevents from recomputing previously enumerated paths.
For time-dependent and multimodal graphs, the extension of Yen's (or Lawler's) algorithm is straightforward using the D_{RegLC} algorithm as the sub-procedure instead of the DIJKSTRA algorithm and produces a set of paths without cycle.

B. Extension for k-shortest paths with cycles

Eppstein's algorithm solves as a first step the shortest path tree rooted at the destination. It then builds an intermediate tree structure representing every possible deviation along the shortest path tree with respect to arcs in the graph. To allow multiple deviations to happen along a path, leaves are connected to the root. Then, a k shortest path algorithm can be applied on the resulting compacted tree yielding to much better performances in finding the shortest paths since it just combines deviations in every possible way instead of exploring them multiple times on the graph. However, when considering time-dependency, it was already shown that backward search is based on lower bounds and then re-computation, and therefore increases computation times. Moreover, taking into account multimodality may require non-deterministic automaton to represent mode constraints, leading to an additional increase of computation time.

Instead of Eppstein's algorithm, we consider the adaptation of the REA algorithm that also computes k shortest paths with cycles. The REA algorithm, starts with a standard one-to-all shortest path algorithm. Then, it works in a recursive way and starts at the destination node to compute the next shortest path. Each node produces then a set of labels corresponding to paths. This list of labels is initialized when the second shortest path is required by extending the computed paths coming from nodes not in the shortest path to the considered node. It is then extended by recursive computations of shortest paths from nodes on the previous shortest path.

The adaptation of this algorithm for taking into account both multimodality and time-dependency is immediate.

C. New variant of k-shortest paths algorithm with cycles

In addition, we propose another algorithm close to REA but working in an iterative way to compute k shortest paths with cycles between an origin o and a destination d. This algorithm, denoted as Iterative Enumeration Algorithm or IEA, computes the shortest path from o to all of the other nodes using a DISJKSTRA-like procedure. The main difference is that several labels can be stored on each node, even if the corresponding node was already visited and therefore marked. Moreover, another mark and an index are directly associated to each label on each node. A label is marked if it has already been selected and propagated. Of course, priority is given to the labels of minimal cost. The index on the label represents an identifier of the label (to re-compute the produced path); indexes are produced in increasing order.
For a given node x, one associates a Boolean mark B_x to state whether this node was already visited or not, and a label l_x defined by $l_x = (c, p(r'), r, b)$ where c is the current cost from the origin to x, p the predecessor node, r' the index of the label on node p, r the index of label on node x and b a Boolean mark to state whether this label was already considered or not.

Initially, all nodes are unmarked (not visited), the cost of all labels is set to infinity except for the origin that has a label with cost 0 (and index 1), and all labels are unmarked. The main steps of the IEA algorithm are the following:

1) select the unmarked node having the unmarked label with the lowest cost, mark this node and this label as visited;
2) add new labels for each successor nodes, even if these nodes are marked, and number the produced labels;
3) when the destination node is selected in step 1 and marked (a label for this node is also marked), store the solution and increase the count on the number of produced paths. Then, unmark all nodes belonging to the shortest path (note that marks remain on labels).
4) restart at step 1 with the new set of unmarked nodes and labels. This set is composed only of nodes belonging to the previously obtained path and of nodes not visited during the previous search.

The algorithm stops when the required number of paths is obtained or when there is no remaining unmarked node or when there is no remaining unmarked labels. Note that in this algorithm, we can also suppress cycles going through the origin by do not allow to unmark the origin in step 3 (and do not produce new labels on the origin in step 2).

Let us consider the example given in figure 1 with 4 nodes. The origin is x_1 and the destination x_4. In this small example, there are 4 different paths without cycle: $\pi^1 = \{x_1, x_2, x_4\}$ with cost 4, $\pi^2 = \{x_1, x_2, x_3, x_4\}$ with cost $c(\pi^2) = 4$, $\pi^3 = \{x_1, x_3, x_2, x_4\}$ with cost $c(\pi^3) = 10$ and $\pi^4 = \{x_1, x_3, x_4\}$ with cost $c(\pi^4) = 11$.

In the proposed algorithm node x_1 received a label $l_{x_1} = (0, \emptyset, 1, false)$ and it is not visited ($v_{x_1} = false$). In table I, we detail the labels produced on each node. At the beginning, the origin x_1 is selected, marked as visited, its label is also marked as visited and it is propagated through x_2 and x_3 (line1). In line 2, the unmarked node x_2 has the unmarked label with lowest cost (1), it is marked as visited, its label is also marked and its successors receive new labels (nodes x_3 and x_4). In line 3, node x_3 is selected as it is unmarked and has the lowest cost (3) and produce new labels on node x_2

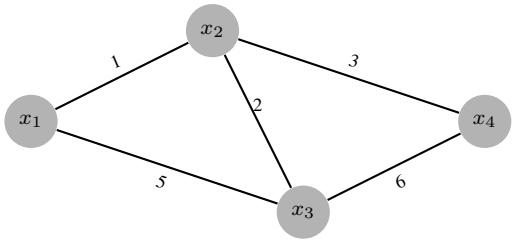

Fig. 1. Toy example for the computation of k-shortest paths

TABLE I. LABELS PRODUCED DURING THE IEA ALGORITHM

step	x_2	x_3	x_4
1	(F; {(1, x_1(1), 1, F)})	(F; {(5, x_1(1), 1, F)})	-
2	(**T**; {(1, x_1(1), 1, **T**)})	(F; {(5, x_1(1), 1, F), (3, x_2(1), 2, F)})	(F, {(4, x_2(1), 1, F)})
3	(**T**; {(1, x_1(1), 1, **T**), (5, x_3(2), 2, F)})	(**T**; {(5, x_1(1), 1, F), (3, x_2(1), 2, **T**)})	(F, {(4, x_2(1), 1, F), (9, x_3(2), 2, F)})
4	(**T**; {(1, x_1(1), 1, **T**), (5, x_3(2), 2, F)})	(**T**; {(5, x_1(1), 1, F), (3, x_2(1), 2, **T**)})	(**T**, {(4, x_2(1), 1, **T**), (9, x_3(2), 2, F)})
5	(**T**; {(1, x_1(1), 1, **T**), (5, x_3(2), 2, **T**) })	(**T**; {(5, x_1(1), 1, F), (3, x_2(1), 2, **T**), (7, x_2(2), 3, F) })	(F, {(4, x_2(1), 1, **T**), (9, x_3(2), 2, F), (8, x_2(2), 3, F) })
6	(**T**; {(1, x_1(1), 1, **T**), (5, x_3(2), 2, **T**) })	(**T**; {(5, x_1(1), 1, F), (3, x_2(1), 2, **T**), (7, x_2(2), 3, F) })	(**T**, {(4, x_2(1), 1, **T**), (9, x_3(2), 2, F), (8, x_2(2), 3, **T**) })
8	(F; {(1, x_1(1), 1, **T**), (5, x_3(2), 2, **T**), (7, x_3(1), 3, F) })	(**T**; {(5, x_1(1), 1, **T**), (3, x_2(1), 2, **T**), (7, x_2(2), 3, F) })	(F, {(4, x_2(1), 1, **T**), (9, x_3(2), 2, F), (8, x_2(2), 3, **T**), (11, x_3(1), 4, F) })
9	(**T**; {(1, x_1(1), 1, **T**), (5, x_3(2), 2, **T**), (7, x_3(1), 3, **T**) })	(**T**; {(5, x_1(1), 1, **T**), (3, x_2(1), 2, **T**), (7, x_2(2), 3, F), (9, x_2(3), 4, F) })	(F, {(4, x_2(1), 1, **T**), (9, x_3(2), 2, F), (8, x_2(2), 3, **T**), (11, x_3(1), 4, F), (10, x_2(3), 5, F) })
10	(**T**; {(1, x_1(1), 1, **T**), (5, x_3(2), 2, **T**), (7, x_3(1), 3, **T**) })	(**T**; {(5, x_1(1), 1, **T**), (3, x_2(1), 2, **T**), (7, x_2(2), 3, F), (9, x_2(3), 4, F) })	(**T**, {(4, x_2(1), 1, **T**), (9, x_3(2), 2, **T**), (8, x_2(2), 3, **T**), (11, x_3(1), 4, F), (10, x_2(3), 5, F) })

and x_4. Then, in line 4, the only unmarked node is x_4, and the algorithm obtains the first path with cost 4, this label is marked on x_4. Nodes belonging to this path ($\pi^1 = \{x_1, x_2, x_4\}$) are unmarked, except the origin.
The algorithm re-starts with two unmarked nodes x_2 and x_4. The node having the unmarked label with the lowest cost is x_2 (cost 5). In line 5, this node and its label are marked and produce new labels on nodes x_3 and x_4. Node x_3 is marked and cannot be selected. The only unmarked node is x_4 with cost 8 (line 6) producing the second path: $\pi^2 = \{x_1, x_2, x_3, x_2, x_4\}$ (path with a cycle).
Then all nodes except the origin are unmarked and the algorithm re-starts. The node having the lowest cost is x_3 that is marked and new labels are produced on x_2 and x_4 (line 8). In line 9, node x_2 is selected (cost 7) and produces new labels on x_3 and x_4. Then, in line 9, node x_4 has the lowest cost (9) and a new path is obtained: $\pi^3 = \{x_1, x_2, x_3, x_4\}$. The algorithm re-starts again with all nodes set as not-visited.

In Algorithm 1, we detail the proposed Iterative Enumeration Algorithm integrating cut-cycles with the following variables :

- L: the set of labels for each node and each index,
- B: the set of marks for each node,
- H: the heap of candidate labels,
- K: the counter of found solutions for each node,
- lc: the length of cut-cycle,
- S: the solution set.

At the beginning, the heap is initialized with the label of the origin o having a given cost c_0. Then, while the number of solutions k' has not been found, the algorithm selects the best available label in H and as it gets extended iff it does not introduce any cycle of length lc. When a path is found, i.e. the node solution is unstacked but with a value of k' lower than the given argument, every node in the associated path is made available for the next step.
The complexity of this algorithm is the same as of REA, i.e. $O(m+k.n.log(n))$. The IEA algorithm can be directly adapted for time-dependent and multimodal graphs.

Algorithm 1 Iterative Enumeration Algorithm

Require: $G = (V, E)$, o, d, k', $f : E \times E \to \mathbb{R}_+$, lc, c_0
Ensure: $S = \{(k*, c*)\}$
1: ▷ Initialize
2: $L_{x,0} \leftarrow (\infty, \emptyset, 0, false),\ \forall x \in V$
3: $L_{o,0} \leftarrow (c_0, \emptyset, 0, false)$
4: $B_x \leftarrow lc,\ \forall x \in V$
5: $H \leftarrow L_{o,0}$
6: $K_x \leftarrow 0,\ \forall x \in V$
7: $S \leftarrow \emptyset$
8: ▷ Select the unmarked label with minimal cost, mark the node and the label
9: **while** $H \neq \emptyset \wedge K_d \neq k'$ **do**
10: $l_x = (c, p(r'), r, b) \leftarrow min_{B_x \succ 0}(H)$
11: $H \leftarrow H \setminus \{l_x\}$
12: $B_x \leftarrow B_x - 1$
13: $b \leftarrow true$
14: $K_x \leftarrow K_x + 1$
15: **if** $x \neq d$ **then**
16: ▷ Produce sucessor labels l_y
17: **for** $y \in \{z \in V \setminus \{o\} \mid (y, z) \in E\}$ **do**
18: ▷ check cycles
19: $(cycle, count) \leftarrow (false, lc)$
20: $l_z = (-, q_z(r'_z), -, -) \leftarrow l_x$
21: **while** $count \neq 0$ **do**
22: **if** $z = x$ **then**
23: $(cycle, count) \leftarrow (true, 0)$
24: **end if**
25: $(count, l_z) \leftarrow (count - 1, L_{q_z, r'_z})$
26: **end while**
27: **if** $\neg cycle$ **then**
28: $l_y \leftarrow (c + f(x, y), x(r), |L_x| + 1, false)$
29: $H \leftarrow H \cup \{l_y\}$
30: **end if**
31: **end for**
32: **else**
33: ▷ unmark the obtained path
34: $S \leftarrow S \cup (K_x, c)$
35: $l_y = (-, q_y(r'_y), r_y, -) \leftarrow l_x$
36: **while** $y \neq o \wedge r_y \neq 1$ **do**
37: $B_y \leftarrow B_y + 1$
38: $l_y \leftarrow L_{q_y, r'_y}$
39: **end while**
40: **end if**
41: **end while**
42: **return** S

D. Cutting cycles during the search

When using REA or IEA algorithms, one produces (with a low worst-case complexity) k shortest paths with cycles, then due to the application, paths with cycles are disregarded. We consider here, how to adapt these two algorithms to suppress some cycles during the search.

1) Embedding cut cycles in REA algorithm: The REA algorithm produces paths with cycles via the origin and the destination nodes. Considering our application, we propose to modify REA algorithm to forbid cycles going through the origin by not allowing recursive calls on the origin node. However, concerning cycles at the destination, only part of them could be removed by preventing every other node to pull path stubs from the destination when k equals to 2, i.e. at the initialization of paths on a node. Also, for every other path, even if cycles can be detected using local storage, removing a cycle within a recursive call would cause the evaluation to stall on the computation of another path with a different k value, yielding to an impractical approach.

2) Embedding cut cycles in IEA algorithm: In the IEA algorithm, we propose to suppress some cycles during the search by checking whether a successor node is or not in the predecessor list of the node having the label with the minimal cost. The size of the predecessor list directly impacts the size of removed cycles. However, when introducing this cycles removal, the shortest paths may appear in a non-increasing order. For instance, let us consider the example given in figure 2 in which we only consider the direct predecessors of each node when extending a label. The origin is x_1 and the destination x_2. In this example, there are 5 different paths without cycle: $\pi^1 = \{x_1, x_2\}$ with cost 3, $\pi^2 = \{x_1, x_3, x_2\}$ with cost $c(\pi^2) = 7$, $\pi^3 = \{x_1, x_4, x_5, x_3, x_2\}$ with cost $c(\pi^3) = 13$, $\pi^4 = \{x_1, x_3, x_5, x_2\}$ with cost $c(\pi^4) = 14$ and $\pi^5 = \{x_1, x_4, x_5, x_2\}$ with cost $c(\pi^5) = 16$.

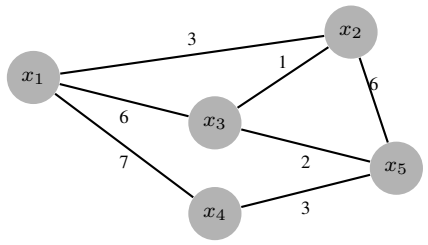

Fig. 2. Heuristic cycles cut

When computing the first path, the IEA algorithm produces the following labels: $\{(3, x_1(1), 1)\}$ on x_2, $\{(6, x_1(1), 1)\}$ on x_3, and $\{(7, x_1(1), 1)\}$ on x_4. Then the path having cost equals to 3 is obtained on the destination x_2. At the next iteration for the computation of the second path, labels $\{(6, x_1(1), 1)\}$ on x_3, and $\{(7, x_1(1), 1)\}$ on x_4 are marked and new labels are produced: $\{(8, x_3(1), 1)\}$ on x_5, $\{(7, x_3(1), 2)\}$ on x_2 and $\{(10, x_3(1), 1)\}$ on x_5. The path having cost equals to 7 is then obtained on the destination. During the following iteration, the label $\{(8, x_3(1), 1)\}$ on x_5 cannot be extended on x_3 as x_3 is the predecessor of this label but it is extended on node x_2 with label $\{(14, x_5(1), 3)\}$ and on node x_4 with label $\{(11, x_5(1), 2)\}$. In this iteration the path with cost 14 is obtained (and not the path having cost 13). During the next iteration, the label $\{(10, x_3(1), 1)\}$ on x_5 is extended to x_3 to obtain the label $\{(12, x_5(2), 2)\}$ and the label $\{(16, x_5(2), 4)\}$ on x_2, but there is no new label on x_4 as it is a predecessor of the considered label. Then, the path having cost 13 is obtained. At the last iteration, the path with cost 16 is produced.

With this example, one can see that the paths are not produced in an increasing order of their cost. Then, using cut-cycles in the IEA algorithm leads to a heuristic approach to obtain the k-shortest paths when limiting the number of produced paths.

TABLE II. RESULTS FOR YEN' EXTENSION

k	time	cost	dev c	hops	dev h	max time
10	23 849	81.403	0.015	90.65	4.16	50 712
20	49 276	81.404	0.093	91.17	4.44	102 012
30	76 052	81.406	0.155	91.63	4.51	153 787
40	100 371	81.407	0.171	91.84	4.36	205 532
50	125 666	81.408	0.199	91.93	4.47	257 166
100	248 875	81.412	0.301	92.88	4.53	519 595

IV. EXPERIMENTAL RESULTS

The tests were carried out on an Intel(R) Core(TM) i5-3337U CPU @ 1.7GHz with 6144 KB cache and 3GB main memory, running Linux 3.2.0.4-amd64. All algorithms were implemented in C++ and compiled with GCC.

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 data sets 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 75 837 nodes, 484 426 road edges and 43 318 public transport edges. All combinations of public transportation modes are authorized, including pedestrian. A generic automaton that allows walk, bus and subway is used and the departure time is set to 9 AM.

We first evaluate the extension of Yen algorithms by computing k-shortest paths without cycles for 40 instances randomly chosen in the network, i.e. different origins and destinations. For each pair origin-destination, we test different values of k. For each value of k, we give in table II, the CPU time in milliseconds (time) in average over the 40 instances, the average cost (cost) defined as the travel time in minutes from the origin to the destination, the number of hops of paths as well as their respective standard deviations (dev c and dev h).

For Yen's algorithm, the average cost of the paths increases slightly with the number of paths generated. The computation time evolution bears out the linear complexity in k since the solving time is almost 24 seconds when k equals 10 versus 248 when k equals 100. In addition, because Yen's algorithm calls the subroutine exactly the number of hops contained in the previous path, the computation time is limited by the average number of hops times the value of k times the time required by D_{RegLC} to solve the shortest path (56 ms in average over all considered instances).

Concerning algorithms computing shortest paths with cycles, we consider another type of experiments. We use the

TABLE III. RESULTS FOR ALGORITHM WITH CYCLES

Nb	REA-0/REA-1				IEA-0				IEA-1			
k'	k	time	cost	dev	k	time	cost	dev	k	time	cost	dev
100	1.18	205.8	81.40	0.0	1.13	589	81.40	0.0	69.78	511	81.41	0.33
200	1.18	260.5	81.40	0.0	1.15	815.5	81.40	0.0	124.13	667.8	81.42	0.52
300	1.18	251.5	81.40	0.0	1.15	1174.3	81.40	0.0	172.45	854	81.42	0.56
400	1.18	256.8	81.40	0.0	1.15	1818.5	81.40	0.0	222.48	998	81.42	1.38

same set of 40 instances but we fix the number of paths with cycles (denoted by k') and extract paths without cycles from the set of computed paths. We then consider the following algorithms; the extension of the initial version of REA (named REA-0), the proposed adaptation to cut cycles going through the origin, named REA-1, the exact variant of the proposed IEA algorithm, named IEA-0 and the heuristic variant where short cycles are cut (cycles of length 2), named IEA-1.

In table III, we give the results of these algorithms in average on the considered instances and for different values of k'. For each algorithm, in the first column, we note k the number of obtained paths without cycles. For the considered values of k', and the transportation network used, algorithms REA-0 and REA-1 obtain the same results. One can note that algorithms REA-0, REA-1 and IEA-0 were not able to produce many paths without cycles and that algorithms IEA are slower than algorithms REA. However, algorithm IEA-1 successfully produced multiple paths without cycles, but with similar costs. The CPU time used by all of these algorithms is far below the time used by the Yen algorithm.

We tried to increase the number of paths with cycles allowed in each algorithms up to $k' = 10000$. For REA-0 or REA-1, the number of paths without cycles is still equal to 1.18 (with 673.8 ms for CPU time). Regarding, IEA-0 and IEA-1, they are limited by the number of labels allowed at each node (the number is bounded by k') and the memory size needed is too high for the values of k' on the machine used.

In Table IV, we show the impact of the cut cycles procedure in IEA algorithms. In IEA-c, cycles having length $c + 1$ are not allowed. The table shows that the number of paths without cycle grows with the length of forbidden cycles and that CPU time remains reasonable (less than 2 seconds for $k' = 600$). Also, it can be observed that the computation time increases slowly with the value of k'. Notwithstanding being a heuristic, IEA produces paths which costs are in average not far at all from those returned by Yen's algorithm. Even if the values of k are not similar, since IEA finds not all k values when k' is provided, the values obtained for $k = 100$ in Yen (81.412) is really close to the one obtained by IEA-$\{1, 2, 3\}$ for $k' = 100$. To sum up, it is possible to produce, in a short computing time, a large number of paths without cycles for transportation networks using REA-c.

V. CONCLUSION

Beside the implementation of usual k shortest paths algorithms and their adaptation to time-dependent transportation networks, an iterative version of the recursive enumeration algorithm allowing to cut cycles of a given size is proposed. This heuristic adaptation is relatively fast and supports the scheme of filtering paths with cycles to obtain elementary paths, while deteriorating as less as possible the cost of the solutions.
However, this algorithm is limited in terms of memory and is not suited for a large number of paths since the number of labels per node increases proportionally. Also, if IEA is interesting for the considered graph, there is still interrogations about its behavior on other topologies as well as its concurrency with a more refined version of Yen's algorithm.
Finally, the goal of generating that many number of elementary paths is to skim them to select viable alternatives to the shortest path.

ACKNOWLEDGMENTS

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

REFERENCES

[1] P. Hart, N. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *Systems Science and Cybernetics, IEEE Transactions on*, vol. 4, no. 2, pp. 100–107, Feb. 1968. [Online]. Available: http://dx.doi.org/10.1109/tssc.1968.300136

[2] T. A. J. Nicholson, "Finding the shortest route between two points in a network," *The Computer Journal*, vol. 9, no. 3, pp. 275–280, Nov. 1966. [Online]. Available: http://dx.doi.org/10.1093/comjnl/9.3.275

[3] D. E. Kaufman and R. L. Smith, "Fastest paths in Time-Dependent networks for intelligent Vehicle-Highway systems application," *Journal of Intelligent Transportation Systems*, vol. 1, no. 1, pp. 1–11, 1993.

[4] C. Barrett, R. Jacob, and M. Marathe, "Formal-language-constrained path problems," *SIAM J. Comput.*, vol. 30, no. 3, pp. 809–837, May 2000.

[5] C. Barrett, K. Bisset, M. Holzer, G. Konjevod, M. Marathe, and D. Wagner, *Engineering Label-Constrained Shortest-Path Algorithms*, ser. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2008, vol. 5034, pp. 27–37.

[6] J. Y. Yen, "Finding the k shortest loopless paths in a network," *Management Science*, vol. 17, no. 11, pp. 712–716, Jul. 1971.

[7] D. Eppstein, "Finding the k shortest paths," *SIAM J. Comput.*, vol. 28, no. 2, pp. 652–673, Feb. 1999.

[8] V. Jimenez and A. Marzal, "Computing the k shortest paths: A new algorithm and an experimental comparison," in *Proceedings of the 3rd International Workshop on Algorithm Engineering*, ser. WAE '99. London, UK, UK: Springer-Verlag, 1999, pp. 15–29.

[9] V. M. Jimenez and A. Marzal, "A lazy version of Eppstein's k shortest paths algorithm," in *2nd Int. Conf. on Experimental and Efficient Algorithms (WEA)*, 2003, pp. 179–191.

[10] E. L. Lawler, "A procedure for computing the k best solutions to discrete optimization problems and its application to the shortest path problem," *Management Science*, vol. 18, no. 7, pp. 401–405, 1972.

TABLE IV. IMPACT OF CUT-CYCLE PROCEDURE

Nb	IEA-1				IEA-2				IEA-3			
k'	k	time	cost	dev	k	time	cost	dev	k	time	cost	dev
100	69.78	511	81.409	0.33	79.63	512	81.412	0.39	90.8	581.8	81.415	0.50
200	124.13	667.8	81.415	0.52	143.28	622.8	81.416	0.54	173.85	739.3	81.424	0.74
300	172.45	854	81.417	0.56	205.38	786	81.421	0.78	252.15	892.8	81.436	1.81
400	222.48	998	81.424	1.38	263.25	949.5	81.434	1.71	326.55	1015	81.454	2.97
500	271.7	1330.5	81.434	1.79	322	1191.8	81.441	1.87	401.2	1171.3	81.469	3.45
600	319.0	1659.8	81.439	1.89	377.08	1698.3	81.447	1.94	469.85	1577.3	81.475	3.51