Project Number 288094

eCOMPASS

eCO-friendly urban **M**ulti-modal route **PlA**nning **S**ervices for mobile u**S**ers

STREP
Funded by EC, INFSO-G4(ICT for Transport) under FP7

eCOMPASS – TR – 056

Computing and Listing st-Paths in Public Transportation Networks

Kateina Bhmov, Mat Mihalk, Tobias Prger, Gustavo Sacomoto, Marie-France Sagot

November 2014

Computing and Listing *st*-Paths in Public Transportation Networks

Kateřina Böhmová[1], Matúš Mihalák[1], Tobias Pröger[1], Gustavo Sacomoto[2,3], and Marie-France Sagot[2,3]

[1] Institut für Theoretische Informatik, ETH Zürich, Switzerland
[2] INRIA Grenoble Rhône-Alpes, France
[3] UMR CNRS 5558 – LBBE, Université Lyon 1, France

Abstract. Given a set of directed paths (called *lines*) L, a *public transportation network* is a directed graph $G_L = (V_L, A_L)$ which contains exactly the vertices and arcs of every line $l \in L$. An *st*-route is a pair (π, γ) where $\gamma = \langle l_1, \ldots, l_h \rangle$ is a line sequence and π is an *st*-path in G_L which is the concatenation of subpaths of the lines $l_1, \ldots, l_h$, in this order. We study three related problems concerning traveling from s to t in G_L. We present an optimal algorithm for computing an *st*-route (π, γ) where $|\gamma|$ (i.e., the number of line changes plus one) is minimum among all *st*-routes. We show for the problem of finding an *st*-route (π, γ) that minimizes the number of different lines in γ, even computing an $o(\log |V|)$-approximation is NP-hard. Finally, given a constant integer β, we present an algorithm for listing all *st*-paths π for which a route (π, γ) with $|\gamma| \leq \beta$ exists, and show that the running time of this algorithm is polynomial with respect to the input and the output size.

1 Introduction

Motivation. Given a public transportation network and two locations s and t, a common goal is to find a fastest route from s to t, i.e. an *st*-route whose travel time is minimum among all *st*-routes. A fundamental feature of any public transportation information system is to provide, given s, t and a target arrival time t_A, a fastest *st*-route that reaches t no later than time t_A. This task can be solved by computing a shortest path in an auxiliary graph [13]. However, if delays occur in the network (which often happens in reality), then the goal of computing a *robust* *st*-route that likely reaches t on time, naturally arises.

The problem of finding robust routes received much attention in the literature (for a survey, see, e.g., [1]). Recently, Böhmova et al. [4] proposed a novel approach for computing robust routes which requires to list all *st*-routes explicitly. Each different route provides an alternative, and exploring all alternatives helps identifying the most robust routes. From a practical point of view it is undesirable to list *all* *st*-routes for two reasons: (1) the number of listed routes might be huge, leading to a non-satisfactory running time, and (2) many routes might be inacceptable for the user, e.g., because they use many more transfers than necessary. Having a huge number of transfers is not only uncomfortable,

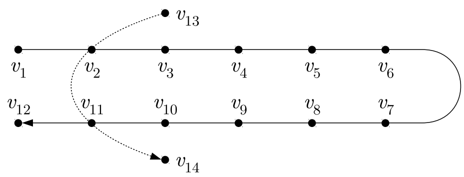

Fig. 1. Consider a bus network with one-way streets induced by a line $l_1 = \langle v_1, \ldots, v_{12} \rangle$ (solid) and a line $l_2 = \langle v_{13}, v_2, v_{11}, v_{14} \rangle$ (dotted). To travel from $s = v_1$ to $t = v_{12}$, it is reasonable to travel to use l_1 until v_2, after that use l_2 from v_2 to v_{11} and from there use l_1 again.

but usually also has a negative impact on the robustness of routes, since each transfer bears a risk of missing the next connection when vehicles are delayed.

Our Contribution. The main contribution of the present paper is an algorithm that lists all *st*-routes for which the number of transfers does not exceed a given constant. The running time of our algorithm is polynomial with respect to both the input as well as the output size. As a subroutine of this algorithm we need to compute a route with a minimum number of transfers. For this problem we also provide an efficient algorithm which might be of independent interest. Moreover, we show that finding a route with a minimum number of *different* lines cannot be approximated within $(1 - \varepsilon) \ln n$ unless NP $\subset$ TIME$(n^{\mathcal{O}(\log \log n)})$.

We note that for simplicity, we write *bus network* instead of *public transportation network*. We also note that for buses it is reasonable to consider directed networks (instead of undirected ones), because real transportation networks might contain one-way streets in which buses can only operate in a single direction. Such a situation occurs in, e.g., the city of Barcelona.

Related Work. Listing combinatorial objects (such as paths, cycles, spanning trees, etc.) in graphs is a widely studied field in computer science (see, e.g., [2]). The currently fastest algorithm for listing all *st*-paths in directed graphs was presented by Johnson [10] in 1975 and runs in time $\mathcal{O}((n + m)(\kappa + 1))$ where n and m are the number of vertices and arcs, respectively, and κ is the number of all *st*-paths (i.e., the size of the output). For undirected graphs, an optimal algorithm was presented by Birmelé et al. [3]. A related problem is the K-shortest path problem, which asks, for a given constant K, to compute the first K distinct shortest *st*-paths. Yen [16] and Lawler [12] studied this problem for directed graphs. Their algorithm uses Dijkstra's algorithm [6] and can be implemented to run in time $\mathcal{O}(K(nm + n^2 \log n))$ using Fibonacci heaps [9]. For undirected graphs, Katoh et al. [11] proposed an algorithm with running time $\mathcal{O}(K(m{+}n \log n))$. Both algorithms use space $\mathcal{O}(Kn{+}m)$. Eppstein [7] gave an $\mathcal{O}(K{+}m{+}n \log n)$ algorithm for listing the first K distinct shortest *st-walks*, i.e., paths where vertices are allowed to appear more than once. Recently, Rizzi et al. [15] studied a different parameterization of the K shortest path problem where they ask to list all st-paths with length at most α for a given α. The difference to the classical K shortest path problem is that the lengths (instead of the overall number) of the paths output is bounded. Thus, depending on the value of α, K might be exponential in the input size. The running time of the proposed algorithm coincides with the running time of the algorithm of Yen and Lawler for directed graphs, and with the running time of the algorithm of Katoh et al. for undirected graphs. However, the algorithm of Rizzi et al. uses only $\mathcal{O}(n+m)$ space which is linear in the input size. All these algorithms cannot directly be used for our listing problem, since we have the additional constraint to list only paths for which a route of length at most β exists.

2 Preliminaries

Mathematical Preliminaries. Given a directed graph $G=(V,A)$ and a vertex $v \in V$, the in- and out-neighborhoods of v are denoted by $N_G^+(v)$ and $N_G^-(v)$, respectively. A *walk* in G is a sequence of vertices $\langle v_0, \ldots, v_k \rangle$ such that $(v_{i-1}, v_i) \in A$ for all $i \in [1,k]$. For a walk $w = \langle v_0, \ldots, v_k \rangle$ and a vertex $v \in V$, we write $v \in w$ if and only if there exists an index $i \in [0,k]$ such that $v = v_i$. Analogously, for a walk $w = \langle v_0, \ldots, v_k \rangle$ and an edge $a = (u,v) \in A$, we write $a \in w$ if and only if there exists an index $i \in [1,k]$ such that $u = v_{i-1}$ and $v = v_i$. The length of a walk $w = \langle v_0, \ldots, v_k \rangle$ is k, the number of arcs in the walk, and is denoted by $|w|$. A walk w of length $|w| = 0$ is called *degenerate*, and *non-degenerate* otherwise. For two walks $w_1 = \langle u_0, \ldots, u_k \rangle$ and $w_2 = \langle v_0, \ldots, v_l \rangle$ with $u_k = v_0$, $w_1 \cdot w_2$ denotes the *concatenation* $\langle u_0, \ldots, u_k = v_0, \ldots, v_l \rangle$ of w_1 and w_2. A *path* is a walk $\pi = \langle v_0, \ldots, v_k \rangle$ such that $v_i \neq v_j$ for all $i \neq j$ in $[0,k]$, i.e. a path is a walk without crossings. Given a path $\pi = \langle v_0, \ldots, v_k \rangle$, every contiguous subsequence $\pi' = \langle v_i, \ldots, v_j \rangle$ is called a *subpath* of π. A path $\pi = \langle s = v_0, v_1, \ldots, v_{k-1}, v_k = t \rangle$ is called an st-path. Given two integers i, j, we define the function δ_{ij} (*Kronecker delta*) as 1 if $i = j$ and 0 if $i \neq j$.

Lines and Bus Networks. Given a set of non-degenerate paths (called *lines*) L, the *bus network induced by* L is the graph $G_L = (V_L, A_L)$ where V_L contains exactly the vertices v for which L contains a line l with $v \in l$, and A_L contains exactly the arcs a for which L contains a line l with $a \in l$. In the following, let $M_L = \sum_{l \in L} |l|$ denote the sum of the lengths of all lines. In the rest of this paper, we omit the index L from V_L, A_L and M_L to simplify the notation. We note that our definition of a bus network does not include travel times or timetables, since we are only interested in the structure of the network. Our modeling differs from classical graph-based models like the *time-expanded* or the *time-dependent* model which incorporate travel times explicitly by adding additional vertices or functions in the edges, respectively (see, e.g., [13, 14] for more information on these models). However, for finding robust routes with the approach in [4], the above definition is sufficient since travel times can be integrated at a later stage.

Given a path $\pi = \langle v_0, \ldots, v_k \rangle$ in G_L and a sequence of lines $\gamma = \langle l_1, \ldots, l_h \rangle$, we say that the pair (π, γ) is a *route* if π is equal to the concatenation of non-degenerate subpaths $\pi_1, \ldots, \pi_h$ of the lines $l_1, \ldots, l_h$, in this order. Notice that a

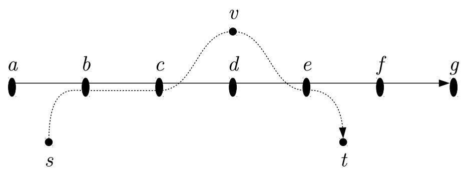

Fig. 2. Let $l = \langle a, b, c, d, e, f, g \rangle$ be a line (solid) and $\pi = \langle s, b, c, v, e, t \rangle$ be a path (dotted). Then, $l - \pi = \langle a, d, f, g \rangle$ is the disjoint union of the degenerate lines $\langle a \rangle$ and $\langle d \rangle$, and the non-generated line $\langle f, g \rangle$.

line might occur multiple times in γ (see Figure 1); however, we assume that any two consecutive lines in γ are different. For every $i \in \{1, \ldots, h-1\}$, we say that a *line change* between the lines l_i and l_{i+1} occurs. The *length* of the route (π, γ) is $|\gamma|$, i.e. the number of line changes plus one. Given two vertices $u, v \in V$, a uv-route is a route (π, γ) such that π is a uv-path. A *minimum* uv-route has smallest length among all uv-routes in G_L, and we define the L-*distance* $d_L(u, v)$ from u to v as the length of a minimum uv-route. For a path π and a line $l \in L$, let $l - \pi$ be the union of (possibly degenerate) paths that we obtain after removing every vertex $v \in \pi$ and its adjacent arcs from l (see Figure 2). For simplicity, we also call each of these unions of paths a *line*, although they might be disconnected and/or degenerated. However, we note that all algorithms in this paper also work for disconnected and/or degenerate lines. Given a path π and a set L of lines, let $L - \pi = \{l - \pi \mid l \in L\}$ denote the set of all lines in which every vertex from π has been removed. Analogously to our previous definitions, given a path π and a graph G, we define $G - \pi$ as the graph from which every vertex $v \in \pi$ and its adjacent arcs have been removed.

Problems. An algorithm that systematically lists all or a specified subset of solutions of a combinatorial optimization problem is called a *listing algorithm*. The *delay* of a listing algorithm is the maximum of the time elapsed until the first solution is output and the times elapsed between any two consecutive solutions are output.

Problem 1 (Finding a minimum st-route). Given a bus network $G_L = (V, A)$ and two vertices $s, t \in V$, find a minimum route from s to t.

Problem 2 (Finding an st-route with a minimum number of different lines). Given a bus network $G_L = (V, A)$ and two vertices $s, t \in V$, find a route from s to t that uses a minimum number of different lines from L.

A natural listing problem is to list all possible st-routes. However, this formulation has the disadvantage that the number of possible solutions is huge, and that there might exist many redundant solutions since a path π can give rise to multiple distinct routes (it is enough that some arc of π is shared by two lines). Moreover, from a practical point of view, also routes that contain many line changes are undesirable. Thus, we formulate the listing problem as follows.

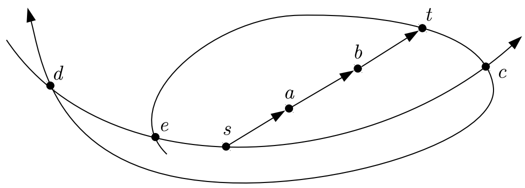

Fig. 3. Consider a bus network induced by the lines $l_1 = \langle s, a\rangle$, $l_2 = \langle a, b\rangle$, $l_3 = \langle b, t\rangle$, $l_4 = \langle d, e, s, c\rangle$ and $l_5 = \langle e, t, c, d\rangle$. The route $r_1 = (\langle s, a, b, t\rangle, \langle l_1, l_2, l_3\rangle)$ is an optimal solution for Problem 1. It uses three different lines and two transfers. However the optimal solution for Problem 2 is the route $r_2 = (\langle s, c, d, e, t\rangle, \langle l_4, l_5, l_4, l_5\rangle)$ which uses only two different lines but three transfers.

Problem 3 (Listing β-bounded st-paths). Given a bus network $G_L = (V, A)$, two vertices $s, t \in V$, and $\beta \in \mathbb{N}$, output all st-paths π such that there exists at least one route (π, γ) with length at most β.

3 Finding an optimal solution

As a preliminary result we show that for undirected lines (i.e., undirected connected graphs where every vertex has degree 2 or smaller) and undirected bus networks, Problems 1 and 2 are equivalent and can be solved in time $\Theta(M)$. After that we show that for directed lines and directed bus networks (as defined in Section 2), Problem 1 can easily be solved using Dial's algorithm [5] on an auxiliary graph while Problem 2 is NP-hard to approximate.

For undirected networks, Problems 1 and 2 essentially are easy because lines can always be traveled in both directions. Of course, this does not hold in the case of directed graphs. Figure 3 gives an example of a directed bus network where the optimal solutions for the two problems differ.

3.1 Undirected lines and undirected bus networks

Theorem 1. *If all lines in L were undirected and G_L was the undirected induced bus network, then Problems 1 and 2 coincide and can be solved in time $\Theta(M)$.*

Proof. Let $r = (\pi, \gamma)$ with $\pi = (\pi_1, \ldots, \pi_h)$ and $\gamma = (l_1, \ldots, l_h)$ be an optimal solution to Problem 2. We first show that there always exists an optimal solution $\bar{r} = (\bar{\pi}, \bar{\gamma})$ that uses every line in $\bar{\gamma}$ exactly once. Suppose that some line l occurred multiple times in γ. Let i be the smallest index such that $l_i = l$, and let j be the largest index such that $l_j = l$. Let v be the first vertex on π_i (i.e., the first vertex on the subpath served by the first occurrence of l), and let w be the last vertex on π_j (i.e., the last vertex on the subpath served by the last occurrence of l). Let π_{sv} be the subpath of π starting in s and ending in v, π_{vw} be a subpath of l from v to w, and π_{wt} be the subpath of π starting in w and ending in t. The route $r' = (\pi', \gamma')$ with $\pi' = \pi_{sv} \cdot \pi_{vw} \cdot \pi_{wt}$ and $\gamma' = (l_1, \ldots, l_{i-1}, l, l_{j+1}, \ldots, l_h)$

is still an st-route, it uses the line l exactly once, and overall it does not use any new line. Thus, repeating the above argument for every line l that occurs multiple times, we obtain a route $\bar{r} = (\bar{\pi}, \bar{\gamma})$ which uses every line in $\bar{\gamma}$ exactly once and which is still an optimal solution to Problem 2.

The above argument can also be applied to show that every optimal solution (π, γ) to Problem 1 uses every line in γ exactly once. Now it easy to see that there exists a solution to Problem 1 with exactly k line changes if and only if there exists a solution to Problem 2 with exactly $k+1$ different lines. Therefore, Problems 1 and 2 are equivalent. They can efficiently be solved as follows. For a given bus network $G_L = (V, A)$, consider the vertex-line incidence graph $G' = (V \cup L, A')$ where

$$A' = \{\{v, l\} \mid v \in V \wedge l \in L \wedge \text{line } l \text{ contains vertex } v\}. \tag{1}$$

Using a breadth-first search we can find a shortest st-path $\langle s, l_1, v_1, \dots, v_{k-1}, l_k, t\rangle$ in G'. Let $\gamma = (l_1, \dots, l_k)$ be the sequence of lines in this path. Now we use a simple greedy strategy to find a path π in the bus network G_L such that π is the concatenation of subpaths of $l_1, \dots, l_k$: we start in s, follow l_1 in an arbitrary direction until we find the vertex v_1; if v_1 is not found, we traverse l_1 in the opposite direction until we find v_1. From v_1 we search v_2 on line l_2, and continue in the same fashion until we reach vertex t on line l_k. Now the pair (π, γ) is a route with a minimum number of transfers (and, with a minimum number of different lines).

We have $|V \cup L| \in \mathcal{O}(M)$ and $|A'| \in \Theta(M)$, thus the breadth-first search runs in time $\Theta(M)$. Furthermore, G' can be constructed from G_L in time $\Theta(M)$. Thus, for undirected lines and undirected bus networks, Problems 1 and 2 can be solved in time $\Theta(M)$. □

3.2 Directed lines and directed bus networks

To solve Problem 1 for a directed bus network $G_L = (V, A)$, we first construct a weighted auxiliary graph $\Gamma[G_L] = (V[\Gamma], A[\Gamma])$ such that $V \subseteq V[\Gamma]$, and for any two vertices $s, t \in V$ the cost of a shortest st-path in $\Gamma[G_L]$ is exactly $d_L(s, t)$. For a given vertex $v \in V$, let $L_v \subseteq L$ be the set of all lines that contain v. We add every vertex $v \in V$ to $V[\Gamma]$. Additionally, for every vertex $v \in V$ and every line $l \in L_v$, we create a new vertex v_l and add it to $V[\Gamma]$. The set $A[\Gamma]$ contains three different types of arcs:

1) For every arc $a = (u, v)$ in a line l, we create a *traveling* arc (u_l, v_l) with cost 0. These arcs are used for traveling along a line l.
2) For every vertex v and every line $l \in L_v$, we create a *boarding* arc (v, v_l) with cost 1. These arcs are used to board the line l at vertex v.
3) For every vertex v and every line $l \in L_v$, we create a *leaving* arc (v_l, v) with cost 0. These arcs are used to leave the line l at vertex v.

Figure 4 shows an example of the graph construction.

Theorem 2. *Problem 1 is solvable in time $\Theta(M)$.*

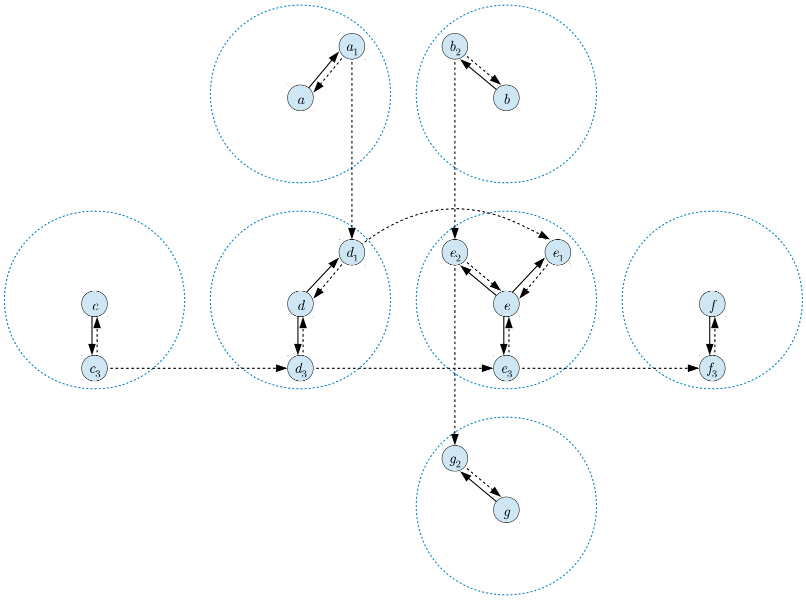

Fig. 4. Consider a bus network G_L with the vertices $V = \{a, \ldots, g\}$ induced by the lines $l_1 = \langle a, d, e \rangle$, $l_2 = \langle b, e, g \rangle$ and $l_3 = \langle c, d, e, f \rangle$. The figure shows the graph $\Gamma[G_L]$ for this bus network. Dotted lines represent arcs of cost 0, solid lines represent arcs of cost 1. The dotted circles represent meta-vertices of the corresponding stations.

Proof. Let $G_L = (V, A)$ be a bus network and $s, t \in V$ be arbitrary. We compute the graph $\Gamma[G_L]$ and run Dial's algorithm [5] on the vertex s. Let π_{st} be a shortest st-path in $\Gamma[G_L]$. It is easy to see that the cost of π_{st} is exactly $d_L(s, t)$. Furthermore, π_{st} induces an st-path in G_L by replacing every traveling arc (v_l, w_l) by (v, w), and ignoring the arcs of the other two types. Analogously the line sequence can be extracted from π_{st} by considering the lines l of all boarding arcs (v, v_l) in π_{st} (or, alternatively, by considering the lines l of all leaving arcs (v_l, v) in π_{st}).

For every vertex v served by a line l, $\Gamma[G_L]$ contains at most two vertices (namely, v_l and v), thus we have $|V[\Gamma]| \in \mathcal{O}(M)$. Furthermore, $A[\Gamma]$ contains every arc a of every line, and exactly two additional arcs for every vertex v_l. Thus we obtain $|A[\Gamma]| \in \mathcal{O}(M)$. Since the largest edge weight is $C = 1$ and Dial's algorithm runs in time $\mathcal{O}(|V[\Gamma]|C + |A[\Gamma]|)$, Problem 1 is solvable in time $\mathcal{O}(M)$ which is optimal since the input has size $\Theta(M)$. □

In contrast to the previous Theorem, we will show now that finding a route with a minimum number of different lines is NP-hard to approximate.

Theorem 3. *Problem 2 cannot be approximated within* $(1 - \varepsilon) \ln n$ *unless NP* $\subset$ *TIME*$(n^{\mathcal{O}(\log \log n)})$.

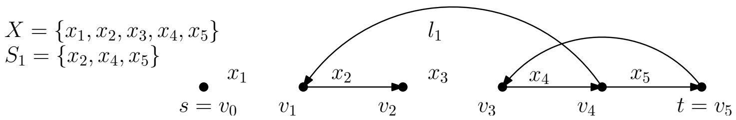

Fig. 5. The correspondence between a set $S_1 \subseteq X$ and a line l_i of the bus network.

Proof. We construct an approximation preserving reduction from SetCover. The reduction is similar to the one presented in [17] for the minimum-color path problem. Given an instance $I = (X, \mathcal{S})$ of SetCover, where $X = \{x_1, \ldots, x_n\}$ is the ground set, and $\mathcal{S} = \{S_1, \ldots, S_m\}$ is a family of subsets of X, the goal is to find a minimum cardinality subset $\mathcal{S}' \subseteq \mathcal{S}$ such that the union of the sets in $\mathcal{S}'$ contains all elements from X.

We construct from I a set of lines L that induces a bus network $G_L = (V, A)$ as follows. The set L consists of m lines and induces $n + 1$ vertices. The vertex set $V = \{v_0, v_1, \ldots, v_n\}$ contains one vertex v_i for each element x_i of the ground set X, plus one additional vertex v_0. Let $V^O = \langle v_0, v_1, \ldots, v_n \rangle$ be the order naturally defined by V. The set of lines $L = \{l_1, \ldots, l_m\}$ contains one line for each set in $\mathcal{S}$. For a set $S_i \in \mathcal{S}$, consider the set of vertices that correspond to the elements in S_i and order them according to V^O to obtain $\langle v_{i_1}, v_{i_2}, \ldots, v_{i_r} \rangle$. Now we define the line l_i as $\langle v_{i_r-1}, v_{i_r}, v_{i_{(r-1)}-1}, v_{i_{(r-1)}}, \ldots, v_{i_1-1}, v_{i_1} \rangle$. Observe that the set of arcs A induced by L contains two types of arcs. There are arcs of the form (v_{i-1}, v_i) for some $i \in [1, n]$. These are the only arcs in A whose direction agrees with the order V^O, and we refer to them as *forward* arcs. For all the other arcs $(u, v) \in A$ we have $u > v$ with respect to the order V^O, and we refer to these arcs as *backward* arcs. We note that every line l_i is constructed so that the forward arcs of l_i correspond to those elements of X that are contained in S_i, and the backward arcs connect the forward arcs, in the order opposite to V^O (see Figure 5), thus making the lines connected.

Now, for $s = v_0$ and $t = v_n$, we show that an st-route with a minimum number of different lines in the given bus network G_L provides a minimum SetCover for I, and vice versa. Since t is after s in the order V^O, and the only forward arcs in G_L are of the form (v_{i-1}, v_i) for some i, it follows that any route from s to t in G_L goes via all the vertices, in the order V^O. Thus, for each st-route $r = (\pi, \gamma)$, there exists an st-route $r' = (\pi', \gamma')$ which does not use any additional lines to those used in r, but contains no backward arc. That is, γ' is a subsequence of γ, and $\pi' = \langle v_0, v_1, \ldots, v_n \rangle$. In particular, there exists an st-route that minimizes the number of different lines, and its path is $\langle v_0, v_1, \ldots, v_n \rangle$. Clearly, if a line l_i is used in the st-route r, all the forward arcs in l_i correspond to the arcs of the path in r and in this way the line l_i "covers" these arcs. Since there is a one to one mapping between the lines and the sets in $\mathcal{S}$, by finding an st-route with k different lines, one finds a solution of the same size to the original SetCover. Similarly each solution of size k to the original SetCover can be mapped to an st-route with k lines. Thus the above reduction is approximation preserving, and based on the inapproximability of SetCover [8] this concludes the proof. □

4 Listing all solutions

Motivation. A naïve approach for solving Problem 3 is to use the algorithm in [4] to generate all feasible line sequences γ, i.e., all line sequences γ with $|\gamma| \leq \beta$ for which G_L contains an st-path π such that (π, γ) is a route. After that, we could compute the corresponding paths (there might be more than one) for each feasible γ. However, this approach has two main drawbacks. First, the running time of the algorithm is exponential in β, independently of K, the number of listed paths. Second, for every path π there might be many line sequences γ such that (π, γ) is route in G_L. Since we want to output every path π at most once, we need to store $\Omega(K)$ many paths. In the following, we present a polynomial delay algorithm that uses only $\mathcal{O}(m)$ space where m is the number of edges in G_L. It is important to stress that the order in which the solutions are output by the algorithm is fixed, but arbitrary.

Improved Idea. Let $\mathcal{P}^{\beta}_{st}(L)$ denote the set of all st-paths π such that there exists a route (π, γ) with length at most β in the bus network G_L. Our algorithm works similar to a depth first search and recursively partitions the solution space (i.e., $\mathcal{P}^{\beta}_{st}(L)$) at every call until the considered subspace is a singleton (i.e., contains exactly one solution) and in that case outputs the corresponding path. At a generic recursive step (u, π_{su}, G), let u be some vertex (initially, $u = s$), let π_{su} be the su-path discovered so far (initially, $\pi_{su} = \langle s \rangle$), and let G be the graph that we obtain after removing all vertices in π_{su} except u from G_L (initially, $G = G_L$). By $\mathcal{P}_{\beta}(\pi_{su})$ we denote the set of all st-paths to be listed by the current recursive call on (u, π_{su}, G), i.e. the subset of paths in $\mathcal{P}^{\beta}_{st}(L)$ that have prefix π_{su}. To bound the overall running time of the algorithm, we maintain the invariant that the current partition (i.e., $\mathcal{P}_{\beta}(\pi_{su})$) contains at least one solution.
Invariant: (I) There exists at least one ut-path π_{ut} in G that extends π_{su} so that it belongs to $\mathcal{P}^{\beta}_{st}(L)$, i.e. $\pi_{su} \cdot \pi_{ut} \in \mathcal{P}^{\beta}_{st}(L)$.
Base case: When $u = t$, output the st-path π_{su}.
Recursive rule: We observe that the set $\mathcal{P}_{\beta}(\pi_{su})$ is the union of the disjoint sets $\mathcal{P}_{\beta}(\pi_{su} \cdot a)$ for each arc $a = (u, v) \in G$ outgoing from u. Thus we perform a recursive call on $(v, \pi_{su} \cdot a, G - \langle u \rangle)$ for every arc $a \in G$ for which $\mathcal{P}_{\beta}(\pi_{su} \cdot a)$ is not empty. This additional condition is required to maintain the invariant (I).

Checking whether to recurse or not. We recurse on $(v, \pi_{su} \cdot a, G - \langle u \rangle)$ only if the invariant (I) is satisfied, i.e., if $\mathcal{P}_{\beta}(\pi_{su} \cdot a)$ is non-empty. For checking this condition, we first set $L' = L - \pi_{su}$ and $G' = G_L - \pi_{su} = G_{L'}$. Let $d_{G_L}(\pi_{su}, a, l_i)$ be the length of a minimum route $(\pi_{su} \cdot a, \gamma)$ in G_L such that l_i is the last line of γ. Let $d^{L'}_{G'}(v, t, l_j)$ be the L'-distance from v to t in G' such that l_j is the first line used. For a vertex $v \in V$, let $L_v \subseteq L$ be the set of all lines that contain an outgoing arc from v. Analogously, for an arc $a \in A$, let L_a be the set of all lines that contain a. Now, the set $\mathcal{P}_{\beta}(\pi_{su} \cdot a)$ is not empty if and only if

$$\min \left\{ d_{G_L}(\pi_{su}, a, l_i) - \delta_{ij} + d^{L'}_{G'}(v, t, l_j) \mid l_i \in L_a \text{ and } l_j \in L_v \right\} \leq \beta. \qquad (2)$$

Basically, $\min\{d_{G_L}(\pi_{su}, a, l_i) - \delta_{ij} + d^{L'}_{G'}(v, t, l_j) \mid l_i \in L_a \text{ and } l_j \in L_v\}$ is the length of the minimum route that has prefix $\pi_{su} \cdot a$.

Computing $d_{G_L}(\pi_{su}, a, l_i)$ *and* $d^{L'}_{G'}(v, t, l_j)$. We can use the solution for Problem 1 to compute the values $d_{G_L}(\pi_{su}, a, l_i)$ and $d^{L'}_{G'}(v, t, l_j)$. The values $d_{G_L}(\pi_{su}, a, l_i)$ need to be computed only for arcs $a = (u, v) \in A$ with $v \notin \pi_{su}$ (i.e., only for arcs from u to a vertex $v \in N^-_{G_L}(u) \cap G'$), and only for lines $l_i \in L_a$. Consider the graph G'' that contains every arc from π_{su} and every arc $(u, v) \in A$ with $v \notin \pi_{su}$, and that contains exactly the vertices incident to these arcs. Now we compute $H = \Gamma[G'']$ and run Dial's algorithm on the vertex s. For every $v \in N^-_{G_L}(u) \cap G'$ and every line $l_i \in L_{(u,v)}$, the length of a shortest path in H from s to v_{l_i} is exactly $d_{G_L}(\pi_{su}, (u, v), l_i)$. For computing $d^{L'}_{G'}(v, t, l_j)$, we can consider the L'-distances from t in the reverse graph G'^R (with all the arcs and lines in L' reversed). Considering G' instead of G_L ensures that lines do not use vertices that have been deleted in previous resursive calls of the algorithm. Thus we compute $\Gamma[G'^R]$ and run Dial's algorithm on the vertex t. Then, the length of a shortest path in $\Gamma[G'^R]$ from t to v_{l_j} is exactly $d^{L'}_{G'}(v, t, l_j)$.

Algorithm. Algorithm 1 implements the recursive partition strategy. To limit the space consumption of the algorithm, we do not pass the graph G' as a parameter to the recursive calls, but compute it at the beginning of each recursive call from the current prefix π_{su}. For the same reason, we do not perform the recursive calls immediately in step 8, but first create a list $V_R \subseteq V$ of vertices for which the invariant (I) is satisfied, and only then recurse on $(v, \pi_{su} \cdot v)$ for every $v \in V_R$.

Algorithm 1: ListPaths(u, π_{su})

1 **if** $u = t$ **then** Output(π_{su}); **return**
2 $L' \leftarrow L - \pi_{su}$; $G' \leftarrow G_L - \pi_{su}$
3 Compute $d_{G_L}(\pi_{su}, (u, v), l_i)$ for each $v \in N^-_{G_L}(u) \cap G'$ and $l_i \in L_{(u,v)}$
4 Compute $d^{L'}_{G'}(v, t, l_j)$ for each $v \in N^-_{G_L}(u) \cap G'$ and $l_j \in L_v$
5 $V_R \leftarrow \emptyset$
6 **for** $v \in N^-_{G_L}(u) \cap G'$ **do**
7     $d \leftarrow \min\{d_{G_L}(\pi_{su}, (u, v), l_i) + d^{L'}_{G'}(v, t, l_j) - \delta_{ij} \mid i \in L_{(u,v)} \text{ and } l_j \in L_v\}$
8   **if** $d \leq \beta$ **then** $V_R \leftarrow V_R \cup \{v\}$
9 **for** $v \in V_R$ **do**
10   ListPaths$(v, \pi_{su} \cdot (u, v))$

Theorem 4. *Algorithm 1 has delay $\mathcal{O}(nM \log M)$, where n is the number of vertices in G_L. The total time complexity is $\mathcal{O}(nM \log M \cdot K)$, where is K the number of returned solutions. Moreover, the space complexity is $\mathcal{O}(m)$ where m is the number of arcs in G_L.*

Proof. We first analyze the cost of a given call to the algorithm without including the cost of the recursive calls performed inside. Theorem 2 states that steps 3 and 4 can be performed in time $\mathcal{O}(M)$. We will now show that steps 6–8 can be implemented in time $\mathcal{O}(M \log M)$. Notice that for a fixed prefix π_{su} and a fixed vertex $v \in N^{-}_{G_L}(u) \cap G'$, for computing the minimum in step 7, we need to consider only the values $d_{G_L}(\pi_{su}, (u,v), l_i)$ that are minimum among all $d_{G_L}(\pi_{su}, (u,v), \cdot)$, and only the values $d^{L'}_{G'}(v,t,l_j)$ that are minimum among all $d^{L'}_{G'}(v,t,\cdot)$. Let $\Lambda_v \subseteq L_{(u,v)}$ be the list of all lines l_i for which $d_{G_L}(\pi_{su}, (u,v), l_i)$ is minimum among all $d_{G_L}(\pi_{su}, (u,v), \cdot)$. Analogously, let $\Lambda'_v \subseteq L_v$ be the list of all lines l_j for which $d^{L'}_{G'}(v,t,l_j)$ is minimum among all $d^{L'}_{G'}(v,t,\cdot)$. Let

$$\mu_v = \min\left\{d_{G_L}(\pi_{su}, (u,v), l_i) \mid l_i \in \Lambda_v\right\} \tag{3}$$

$$\mu'_v = \min\left\{d^{L'}_{G'}(v,t,l_j) \mid l_j \in \Lambda'_v\right\} \tag{4}$$

be the minimum values of $d_{G_L}(\pi_{su}, (u,v), \cdot)$ and $d^{L'}_{G'}(v,t,\cdot)$, respectively. Both values as well as the lists Λ_v and Λ'_v can be computed in steps 3 and 4, and their computation only takes overall time $\mathcal{O}(M)$. Now the expression in step 7 evaluates to $\mu_v + \mu'_v$ if $\Lambda_v \cap \Lambda'_v = \emptyset$, and to $\mu_v + \mu'_v - 1$ otherwise. Assuming that Λ_v and Λ'_v are ordered in increasing order by the index of the contained lines l_i, it can easily be checked with $|\Lambda_v| + |\Lambda'_v| \leq |L_{(u,v)}| + |L_v|$ many comparisons if their intersection is empty or not. Using this method, each of the values $d_{G_L}(\pi_{su}, \cdot, \cdot)$ and $d^{L'}_{G'}(\cdot, t, \cdot)$ is accessed exactly once (when computing Λ_v and Λ'_v), and since each of these values has a unique corresponding vertex in the graphs H and $\Gamma[G'^R]$, there exist at most $\mathcal{O}(M)$ many such values. Thus, the running time of the steps 6–8 is bounded by $\mathcal{O}(M \log M)$ which is also an upper bound on the running time of Algorithm 1 (ignoring the the recursive calls). Notice that we only obtain the upper bound $\mathcal{O}(M \log M)$ instead of $\mathcal{O}(M)$ because the lists Λ_v and Λ'_v have to be sorted.

We now look at the structure of the recursion tree. The height of the recursion tree is bounded by n, since at every level of the recursion tree a new vertex is added to the current solution and any solution has at most n vertices. In that way, the path between any two leaves in the recursion tree has at most $2n$ nodes. Since each recursive call outputs a solution, the time elapsed between two solutions being output is $\mathcal{O}(nM \log M)$.

For analyzing the space complexity, observe that L', G' and the values $d_{G_L}(\pi_{su}, (u,v), l_i)$ and $d^{L'}_{G'}(v,t,l_j)$ can be removed from the memory after step 8 since they are not needed any more. Thus, we only need to store the lists V_R between the recursive calls. Consider a path in the recursion tree, and for each recursive call i, let u^i be the vertex u and V^i_R be the list V_R of the i-th recursive call. Since V^i_R contains only vertices adjacent to u^i and u^i is never being considered again in *any* succeeding recursive call $j > i$, we have

$$\sum_i |V^i_R| \leq m, \tag{5}$$

which proves the space complexity of $\mathcal{O}(m)$. $\square$

Acknowledgements. This work has been partially supported by the Swiss National Science Foundation (SNF) under the grant number 200021 138117/1, and by the EU FP7/2007-2013 (DG CONNECT.H5-Smart Cities and Sustainability), under grant agreement no. 288094 (project eCOMPASS). Kateřina Böhmová is a recipient of a Google Europe Fellowship in Optimization Algorithms, and this research is supported in part by this Google Fellowship. Gustavo Sacomoto is a recipient of a grant from the European Research Council under the European Community's Seventh Framework Programme (FP7/2007-2013)/ERC grant agreement n° [247073]10 SISYPHE.

References

1. Bast, H., Delling, D., Goldberg, A., Müller-Hannemann, M., Pajor, T., Sanders, P., Wagner, D., Werneck, R.: Route planning in transportation networks (2014)
2. Bezem, G., Leeuwen, J.v.: Enumeration in graphs. Tech. Rep. RUU-CS-87-07, Utrecht University (1987)
3. Birmelé, E., Ferreira, R.A., Grossi, R., Marino, A., Pisanti, N., Rizzi, R., Sacomoto, G.: Optimal listing of cycles and st-paths in undirected graphs. In: SODA. pp. 1884–1896 (2013)
4. Böhmová, K., Mihalák, M., Pröger, T., Šrámek, R., Widmayer, P.: Robust routing in urban public transportation: How to find reliable journeys based on past observations. In: ATMOS. pp. 27–41 (2013)
5. Dial, R.B.: Algorithm 360: shortest-path forest with topological ordering. Commun. ACM 12(11), 632–633 (1969)
6. Dijkstra, E.W.: A note on two problems in connexion with graphs. Numerische Mathematik 1(1), 269–271 (1959)
7. Eppstein, D.: Finding the k shortest paths. SIAM J. Comput. 28(2), 652–673 (1998)
8. Feige, U.: A threshold of ln n for approximating set cover. J. ACM 45(4), 634–652 (1998)
9. Fredman, M.L., Tarjan, R.E.: Fibonacci heaps and their uses in improved network optimization algorithms. J. ACM 34(3), 596–615 (1987)
10. Johnson, D.B.: Finding all the elementary circuits of a directed graph. SIAM J. Comput. 4(1), 77–84 (1975)
11. Katoh, N., Ibaraki, T., Mine, H.: An efficient algorithm for k shortest simple paths. Networks 12(4), 411–427 (1982)
12. Lawler, E.L.: A procedure for computing the k best solutions to discrete optimization problems and its application to the shortest path problem. Mgmt. Sci. 18, 401–405 (1972)
13. Müller-Hannemann, M., Schulz, F., Wagner, D., Zaroliagis, C.D.: Timetable information: Models and algorithms. In: ATMOS. pp. 67–90 (2004)
14. Pyrga, E., Schulz, F., Wagner, D., Zaroliagis, C.D.: Efficient models for timetable information in public transportation systems. ACM J. Exp. Algor. 12 (2007)
15. Rizzi, R., Sacomoto, G., Sagot, M.: Efficiently listing bounded length st-paths. CoRR abs/1411.6852 (2014)
16. Yen, J.Y.: Finding the k shortest loopless paths in a network. Mgmt. Sci. 17, 712–716 (1971)
17. Yuan, S., Varma, S., Jue, J.P.: Minimum-color path problems for reliability in mesh networks. In: INFOCOM. pp. 2658–2669 (2005)