Efficient Computation of Shortest Paths in Time-Dependent Multi-Modal Networks[1]

DOMINIK KIRCHLER, LIX, École Polytechnique; LIPN, Univ. Paris 13; Mediamobile, Ivry Sur Seine

LEO LIBERTI, LIX, École Polytechnique

ROBERTO WOLFLER CALVO, LIPN, Université Paris 13

We consider shortest paths on time-dependent multi-modal transportation networks where restrictions or preferences on the use of certain modes of transportation may arise. We model restrictions and preferences by means of regular languages. Methods for solving the corresponding problem (called the *regular language constrained shortest path problem*) already exist. We propose a new algorithm, called State Dependent ALT (`SDALT`), which runs considerably faster in many scenarios. Speed-up magnitude depends on the type of constraints. We present different versions of `SDALT` including uni-directional and bi-directional search. We also provide extensive experimental results on realistic multi-modal transportation networks.

Categories and Subject Descriptors: G.2.2 [**Discrete Mathematics**]: Graph Theory—*Path and circuit problems*

General Terms: Algorithms, Experimentation

Additional Key Words and Phrases: constrained shortest paths, regular languages, ALT, multi-modal, shortest path

ACM Reference Format:

acmjea ACM J. Exp. Algor. V, N, Article A (January YYYY), 46 pages.

DOI = 10.1145/0000000.0000000 http://doi.acm.org/10.1145/0000000.0000000

[1]This work considerably extends [Kirchler et al. 2011; Kirchler et al. 2012]

Authors' addresses: **D. Kirchler**, Mediamobile, 27, bld Hippolyte Marqus, 94200 Ivry sur Seine, France. Email: kirchler@lix.polytechnique.fr. **L. Liberti**, LIX, École Polytechnique, 91128 Palaiseau, France. Email: liberti@lix.polytechnique.fr. **R. Wolfler Calvo**, LIPN, Univ. Paris 13, Avenue J.B. Clément, 93430 Villetaneuse, France. Email: roberto.wolfler@lipn.univ-paris13.fr.

DOI 10.1145/0000000.0000000 http://doi.acm.org/10.1145/0000000.0000000

1. INTRODUCTION

Multi-modal transportation networks include roads, public transportation, bicycle lanes, etc. Shortest paths in such networks must satisfy some additional constraints: passengers may want to exclude some transportation modes, e.g., the bicycle when it is raining or the car at moments of heavy traffic. Furthermore, they may wish to pass by a particular location (e.g., a grocery shop), or limit the number of changes when using different modes of transportation. Feasibility also has to be assured: private cars or bicycles can only be used when they are available.

The *regular language constrained shortest path problem* (RegLCSP) deals with this kind of problem. It uses an appropriately labeled graph and a regular language to model constraints. A valid shortest path minimizes some cost function (distance, time, etc.) and, in addition, the word produced by concatenating the labels on the arcs along the shortest path must form an element of the regular language. In [Barrett et al. 2000], a systematic theoretical study of the more general *formal language constrained shortest path problem* can be found. It proposes a generalization of Dijkstra's algorithm ($\mathtt{D}_{\mathsf{RegLC}}$) to solve RegLCSP.

In recent years many scholars have worked on speed-up techniques for Dijkstra's algorithm [Dijkstra 1959] and shortest paths on continental-sized road networks can now be found in a few milliseconds [Delling et al. 2009b]. The $\mathtt{D}_{\mathsf{RegLC}}$ algorithm has received less attention. First attempts to adapt speed-up techniques of Dijkstra's algorithm to $\mathtt{D}_{\mathsf{RegLC}}$ are described in [Barrett et al. 2008].

Our Contribution. In this work, we adapt the `ALT` algorithm [Goldberg and Harrelson 2005] to $\mathtt{D}_{\mathsf{RegLC}}$ to speed up its performance. The `ALT` algorithm uses pre-processed data to guide Dijkstra's algorithm toward the target more efficiently. The idea is to adapt `ALT` to $\mathtt{D}_{\mathsf{RegLC}}$ by transferring some information on the regular language of the RegLCSP instance (which is known beforehand) to a preprocessing phase. So for each regular language, we produce specific preprocessed data which guide $\mathtt{D}_{\mathsf{RegLC}}$. We call this algorithm *State Dependent `ALT`* (`SDALT`) and we present uni-directional and bi-directional versions. We also show how to apply approximation. We provide experimental results on two realistic multi-modal transportation networks, of the French region Ile-de-France (which includes Paris and its suburbs) and of New York City. For both graphs we consider various transportation modes: walking, private car, private bike, and public transportation. For the network of Ile-de-France we also include rental bicycles, rental cars, and changing traffic conditions over the day. The experiments show that our algorithm performs better than D_{RegLC}, especially in cases where all modes of transportations have the same speed, or, more generally, that the constraints cause a major detour on the non-constrained shortest path. We observed speed-ups of a factor of 1.5 to 40 (up to a factor of 60 with approximation), in respect to D_{RegLC}.

2. RELATED WORK

Early works on the use of regular languages in the context of shortest path problems with applications to database queries include [Romeuf 1988; Mendelzon and Wood 1995; Yannakakis 1990]. In [Lozano and Storchi 2001] a regular language represented as a finite state automaton is used to model path constraints (called path viability) for the bi-objective multi-modal shortest path problem on a multi-modal transportation network.

Algorithmic and complexity-theoretical results on the use of various types of languages for the formal language constrained shortest path problem can be found in [Barrett et al. 2000]. The authors prove that the problem is solvable in deterministic polynomial time when regular languages are used and they provide a generalization of Dijkstra's algorithm (D_{RegLC}). Experimental data on networks including traffic information (modelled as time-dependent arc costs) can be found in [Barrett et al. 2002]. Another application on multi-modal time-dependent transportation networks can be found in [Sherali et al. 2003], [Sherali et al. 2006] introduces turn penalties.

Recently, much effort has been put into accelerating algorithms to solve the unimodal shortest path problem on large road networks, see [Delling et al. 2009b] for a comprehensive overview. It identifies three basic concepts common to most modern speed-up techniques: bi-directional search, goal-directed search, and contraction. It includes dynamic time-dependent graphs, which are used to model and elaborate real-time traffic conditions. The authors of [Delling et al. 2011] propose a highly flexible and fast algorithm supporting arbitrary cost functions and turn costs.

The ALT algorithm [Goldberg and Harrelson 2005] is a bi-directional, goal directed search technique based on the A^* search algorithm [Hart et al. 1968]. It uses lower bounds on the distance to the target to guide Dijkstra's algorithm. UniALT is the uni-directional version of the ALT algorithm. Efficient implementations of uniALT and ALT as well as experimental data on continental size road networks with time-dependent arc costs are given in [Nannicini et al. 2008].

An advantage of A^* and ALT is that they can easily be adapted to dynamic networks, such as road networks that are periodically updated with real time traffic information. Efficient algorithms including contractions and experimental results can be found in [Nannicini et al. 2008; Delling and Nannicini 2008].

In [Barrett et al. 2008], various basic speed-up techniques and their combinations including bi-directional and goal-directed search have been applied to $\mathsf{D}_{\mathsf{RegLC}}$ on rail and road networks (static arc costs, no time-dependency). The performance of the proposed algorithms depends on the network properties and on the restrictivity of the regular language.

An advantage of using regular languages is their flexibility: it is quite simple to forbid unfeasible types of paths, e.g., bicycle followed by metro followed by car, to assure that paths do not exceed a maximum number of transfers, or to exclude modes of transportation or certain types of road, e.g., toll roads. Unfortunately, it is not trivial to apply speed-up techniques for algorithms to solve uni-modal shortest path problems to $\mathsf{D}_{\mathsf{RegLC}}$. Therefore, some recent works isolate the public transportation network from road networks so that they can be treated individually and limit a priori the range of allowed types of paths [Delling et al. 2009a; Dibbelt et al. 2012].

The authors of [Delling et al. 2009a] assume that the road network is used only at the beginning and at the end of a path and public transportation is used in between.

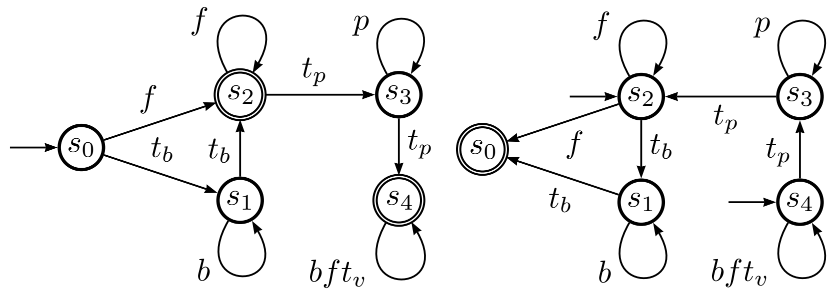
They apply Transit Node Routing to the road network and an adaption of `Dijkstra` to the public transportation network. In [Dibbelt et al. 2012], contraction has been applied only to arcs belonging to the road network of a multi-modal transportation network consisting of roads, public transport, and flight data. The sequence of modes of transportation can be chosen freely and is modeled by a regular language; no update of preprocessed data is needed for different regular languages. The authors report on speed-ups of over 3 orders of magnitude compared to $\mathtt{D}_{\mathsf{RegLC}}$.

The authors of [Rice and Tsotras 2010] use contraction on a continental size road network where roads are labeled according to their road type. A subclass of the regular languages, the Kleene languages, is used to constrain the shortest path. It can be used to exclude certain road types. Kleene Languages are less expressive than regular languages but contraction proves to be very efficient in such a scenario. The authors report on speed-ups of over 3 orders of magnitude compared to $\mathtt{D}_{\mathsf{RegLC}}$.

Overview. This paper is organized as follows. Section 3 defines the graph we are using to model the transportation network and gives more details about RegLCSP, $\mathtt{A}^*$, and `ALT`. Section 4 presents our new algorithm `SDALT`. Different versions of it are presented in Sections 5, 6, and 7. Its application to a realistic multi-modal transportation network and computational results are presented in Section 8.

3. PRELIMINARIES

Consider a *labeled*, directed graph $G = (V, A, \Sigma)$ consisting of a set of nodes $v \in V$, a set of labels $l \in \Sigma$, and a set of arcs $(i, j) \in A \subseteq V \times V$. The labels are used to mark arcs as, e.g., foot paths (label f), bicycle lanes (label b), bus networks (label p_b), etc. Function $\mathtt{Label}(i, j) : A \to \Sigma$ gives the label of an arc (i, j). Arc costs represent travel times. They are positive and time-dependent: $c : A \to (\mathbb{R}_+ \to \mathbb{R}_+)$, i.e., $c_{ij}(\tau)$ gives the travel times from node i to node j at time $\tau \geq 0$. We only use functions which satisfy the FIFO property as the time-dependent shortest path problem in FIFO networks is polynomially solvable [Kaufman and Smith 1993], whereas it is NP-hard in non-



(a) (b)

Fig. 1: Example of an automaton (left) and its backward automaton (right). Shortest paths start either by walking (label f) or by taking a private bicycle: transfer to private bicycle (t_b) and moving on bicycle network (b). Once the private bicycle is discarded (s_1), the path can be continued by walking or by taking public transportation (p). The trip may then be continued by using bicycle rental, by transferring at bicycle rental station to the bicycle network (t_v) or by walking.

FIFO networks [Orda and Rom 1990]. FIFO means that $c_{ij}(x) + x \leq c_{ij}(y) + y$ for all $x, y \in \mathbb{R}_+, x \leq y, (i,j) \in A$ or, in other words, that for any arc (i,j), leaving node i earlier guarantees that one will not arrive later at node j (also called the non-overtaking property).

A path p in G is a sequence of nodes $(v_1, \ldots, v_k)$ such that $(v_{i-1}, v_i) \in A$ for all $1 < i \leq k$. The cost of the path in a time-independent scenario is given by $c(p) = \sum_{i=2}^{k} c_{v_{i-1}v_i}$. In time-dependent scenarios, the cost or travel time $\gamma(p, \tau)$ of a path p departing from v_1 at time τ is recursively given by $\gamma((v_1, v_2), \tau) = c_{v_1 v_2}(\tau)$ and $\gamma((v_1, \ldots, v_j), \tau) = \gamma((v_1, v_{j-1}), \tau) + c_{v_{j-1}, v_j}(\gamma((v_1, v_{j-1}), \tau) + \tau)$.

3.1. Solving the RegLCSP

The *regular language constrained shortest path problem* (RegLCSP) consists in finding a shortest path from a source node r to a target node t with starting time τ_{start} on the labeled graph G by minimizing some cost function (in our case, travel time) and, in addition, the concatenated labels along the shortest path must form a word of a given regular language L_0. The regular language is used to model the constraints on the sequence of labels (e.g., exclusion of labels, predefined order of labels, etc.). Any

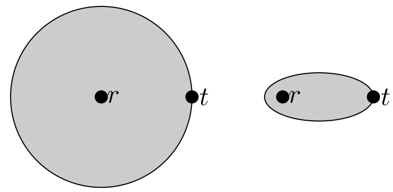


Fig. 2: Schematic search-space Dijkstra (left) and uniALT (right)

regular language L_0 can be described by a *non-deterministic finite state automaton* $\mathcal{A}_0 = (S, \Sigma, \delta, s_0, F)$, consisting of a set of states S, a set of labels Σ, a transition function $\delta : \Sigma \times S \rightarrow 2^S$, an initial state s_0, and a set of final states F (for examples, see Figures 1a and 6a).

To efficiently solve RegLCSP, a generalization of Dijkstra's algorithm (which we denote by $\texttt{D}_{\textsf{RegLC}}$ throughout this paper) has first been proposed in [Barrett et al. 2000]. The $\texttt{D}_{\textsf{RegLC}}$ algorithm can be seen as the application of Dijkstra's algorithm [Dijkstra 1959] to the product graph $G^{\times} = G \times S$ with tuples (v, s) as nodes for each $v \in V$ and $s \in S$ such that there is an arc $((v, s)(w, s'))$ between (v, s) and (w, s') if there is an arc $(i, j) \in A$ with label $l = \texttt{Label}(i, j)$ and a transition such that $s' \in \delta(l, s)$. To reduce storage space, $\texttt{D}_{\textsf{RegLC}}$ works on the *implicit* product graph $G^{\times}$ by generating all the neighbors which have to be explored only when necessary. Similarly to Dijkstra's algorithm, $\texttt{D}_{\textsf{RegLC}}$ can easily be adapted to the time-dependent scenario as shown in [Barrett et al. 2002].

Note some further notation we use throughout this paper: $\overleftrightarrow{S}(s, \mathcal{A})$ and $\overleftrightarrow{\Sigma}(s, \mathcal{A})$ return all states and labels reachable on an automaton $\mathcal{A}$ by starting at state s, backward and forward, respectively. E.g., in Figure 1a, $\overleftarrow{S}(s_2, \mathcal{A}_0) = \{s_0, s_1, s_3\}$, $\overrightarrow{\Sigma}(s_3, \mathcal{A}_0) = \{b, f, t_p, t_v, p\}$. The *backward automaton* of $\mathcal{A}_0$ is produced by reversing all arcs of $\mathcal{A}_0$, final states become initial states and initial states become final states (see Figure 1b). Furthermore, the concatenation of two regular languages L_1 and L_2 is the regular language $L_3 = L_1 \circ L_2 = \{v \circ w | (v, w) \in L_1 \times L_2\}$. E.g., if $L_1 = \{a, b\}$ and $L_1 = \{c, d\}$ then $L_1 \circ L_2 = L_3 = \{ac, ad, bc, bd\}$.

3.2. A* and ALT algorithm

The A* algorithm [Hart et al. 1968] is a goal directed search used to find the shortest path from a source node r to a target node t on a directed graph $G = (V, A)$ with time-independent, non-negative arc costs (without labels on arcs). A* is similar to Dijkstra's algorithm [Dijkstra 1959], which we shall denote by Dijkstra throughout this paper. The difference lies in the order of selection of the next node v to be settled. A* employs a key $k(v) = \tilde{d}(v)+\pi(v)$ where the *potential function* $\pi : V \to \mathbb{R}$ gives an under-estimation of the distance from v to t and $\tilde{d}(v)$ is the tentative distance from the source node r to node v. Note also that we denote by $d(r, t)$ the cost of the *shortest path* between nodes r and t. At every iteration, the algorithm selects the node v with the smallest key $k(v)$. Intuitively, this means that it first explores nodes which lie on the shortest estimated path from r to t. So the closer $\pi(v)$ is to the actual remaining distance, the faster the algorithm finds the target. Note that in the case where $\pi(v)$ gives an exact estimate, A* scans only nodes on shortest paths to t. In contrast, Dijkstra explores nodes in increasing distance from the source node r (see Figure 2).

In [Ikeda et al. 1994], it is shown that A* is equivalent to Dijkstra on a graph with *reduced arc costs* $c^{\pi}_{vw} = c_{vw} - \pi(v) + \pi(w)$. Dijkstra works well only for non-negative arc costs, so not all potential functions can be used. We call a potential function π *feasible*, if c^{π}_{vw} is positive for all $(v, w) \in A$. $\pi(v)$ can be considered a lower bound on the distance from v to t, if π is feasible and the potential $\pi(t)$ of the target is zero. Furthermore, if π' and π'' are feasible potential functions, then $\max(\pi', \pi'')$ is a feasible potential function [Goldberg and Harrelson 2005].

On a road network, the Euclidean distance or air distance from node v to node t can be used to compute $\pi(v)$ (if distance is to be minimized $\pi(v)$ is equal to the air distance and if travel time is to be minimized then $\pi(v)$ is equal to the air distance divided by the maximal travel speed). A significant improvement can be achieved by using landmarks and the triangle inequality [Goldberg and Harrelson 2005]. The main idea is to select a small set of nodes $\ell \in \mathcal{L} \subset V$, spread appropriately over the network, and precompute all distances of shortest paths $d(\ell, v)$ and $d(v, \ell)$ between these nodes

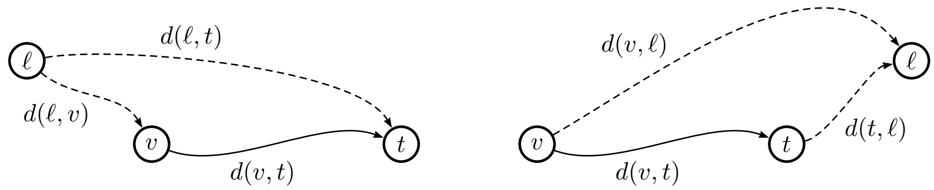


Fig. 3: Landmark distances for `uniALT`.

(also called *landmarks*) and any other node $v \in V$, by using `Dijkstra`. By using these *landmark distances* and the triangle inequality, $d(\ell, v) + d(v, t) \geq d(\ell, t)$ and $d(v, t) + d(t, \ell) \geq d(v, \ell)$, lower bounds on the distances between any two nodes v and t can be derived (see Figure 3). The potential function

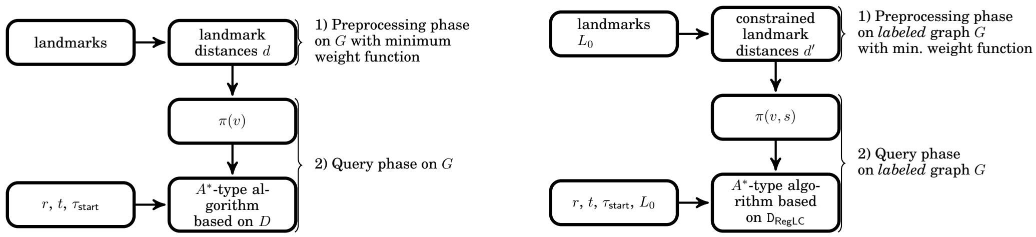
$$\pi(v) = \max_{\ell \in \mathcal{L}} (d(v, \ell) - d(t, \ell), d(\ell, t) - d(\ell, v)) \tag{1}$$

provides a lower bound for the distance $d(v, t)$ and is feasible. The $\mathtt{A}^*$ algorithm based on this potential function is called `ALT` [Goldberg and Harrelson 2005]. The authors propose a uni-directional and bi-directional variant of `ALT`.

As observed in [Delling and Wagner 2009], potentials stay feasible as long as arc weights only increase and do not drop below a minimal value. Based on this, the `ALT` algorithm can be adapted to the time-dependent scenario by selecting landmarks and calculating landmark distances by using the *minimum weight cost function* $c_{ij}^{\min} = \min_\tau c_{ij}(\tau)$. A crucial point is the quality of landmarks. Finding good landmarks is difficult and several heuristics exist [Goldberg and Harrelson 2005; Goldberg and Werneck 2005].

4. STATE DEPENDENT ALT

To speed up $\mathsf{D}_{\mathsf{RegLC}}$, [Barrett et al. 2008] employs among other techniques goal directed search ($\mathtt{A}^*$ search) and bi-directional search on a labeled graph with constant cost function. We go a step further and extend uni- and bi-directional `ALT` to speed-up $\mathsf{D}_{\mathsf{RegLC}}$. Note that we consider labeled graphs with time-dependent arc costs. Furthermore, we enhance the potential function by integrating information about the constraints which



(a) UniALT (b) SDALT

Fig. 4: Comparison uniALT and SDALT.

are modeled by the regular language L_0 (the corresponding automaton is marked as $\mathcal{A}_0 = (S, \Sigma, \delta, s_0, F)$), in a pre-processing phase. E.g., consider a transportation network; in case L_0 excludes a certain mode of transportation, say buses, we can anticipate this constraint by ignoring the bus network during the landmark distance calculation. We will show how to anticipate more complex constraints during the pre-processing phase and we will prove that our approach is correct and yields considerable speed-ups of D_{RegLC} in many scenarios. We will see that one difficulty is to assure feasibility of the potential function. Therefore, we will present two versions of SDALT: lsSDALT, which works with feasible potential functions; and lcSDALT, which also works in cases where the potential function is not always feasible. Furthermore, we will discuss three bi-directional versions of SDALT.

Let us first look at the general structure of the algorithm. The algorithm SDALT, similar to ALT, consists of a *preprocessing phase* and a *query phase* (see Figure 4). The main differences consist in the way landmark distances are calculated and on SDALT being based on D_{RegLC} and not on Dijkstra. Potentials depend on the pair (v, s).

Query phase. The query phase deploys a D_{RegLC} algorithm enhanced by the characteristics of the ALT algorithm. As priority queue Q we use a binary heap. The pseudo code in Algorithm 1 works as follows: the algorithm maintains, for every visited node (v, s) in the product graph $G^{\times}$, a tentative distance label $\tilde{d}(v, s)$ (between source node (r, s_0) and node (v, s)) and a parent node $p(v, s)$. It starts by computing the key $k(r, s_0) = \pi(r, s_0)$ for the source node (r, s_0) and by inserting it into Q (line 3). At every iteration, the algorithm extracts the node (v, s) in Q with the smallest key (it is *settled*) and *relaxes* all outgoing arcs (line 9), i.e., it checks and possibly updates the key and tentative distance label for every node (w, s'), where $s' \in \delta(\texttt{Label}(v, w), s)$. More precisely, a new temporary distance label $\tilde{d}_{\mathsf{tmp}} = \tilde{d}(v, s) + c_{vw}(\tau_{\mathsf{start}} + \tilde{d}(v, s))$ is compared to the currently assigned tentative distance label (line 10). If it is smaller, it either calculates the key $k(w, s') = \pi(r, s_0) + \tilde{d}_{\mathsf{tmp}}$ and inserts (w, s') into the priority queue or decreases its key (line 14, 18). Note that it is necessary to calculate the potential of the node (w, s') only the first time it is visited. The cost of arc (v, w) might be time-dependent and thus has to be evaluated for time $\tau_{\mathsf{start}} + \tilde{d}(v, s)$. The algorithm terminates when a node (t, s') with $s' \in F$ is settled. The resulting shortest path can be produced by following the parent nodes backward starting from (t, s').

Algorithm 1 Pseudo-code `SDALT`.

Input: labeled graph $G = (V, A, \Sigma)$, source r, target t, start time τ_{start}, regular language $L_0 \subseteq \Sigma^*$ represented as automaton $\mathcal{A}_0$

1 **function** SDALT($G, r, t, \tau_{\mathsf{start}}, L_0$)
2 $\quad \tilde{d}(v, s) \leftarrow \infty$, $p(v, s) \leftarrow -1$, $\pi_{v,s} \leftarrow 0$, $\forall (v, s) \in V \times S$
3 $\quad$ pathFound $\leftarrow$ false, $\tilde{d}(r, s_0) \leftarrow 0$, $k(r, s_0) \leftarrow \pi(r, s_0)$, $p(r, s_0) \leftarrow -1$
4 $\quad$ insert (r, s_0) in priority queue Q
5 $\quad$ **while** Q is not empty **do**
6 $\quad\quad$ extract (v, s) with smallest key k from Q
7 $\quad\quad$ **if** $v == t$ and $s \in F_0$ **then**
8 $\quad\quad\quad$ pathFound $\leftarrow$ true
9 $\quad\quad\quad$ **break**
10 $\quad\quad$ **for** each (w, s') s.t. $(v, w) \in \mathcal{A}_0 \wedge s' \in \delta(\texttt{Label}(v, w), s)$ **do**
11 $\quad\quad\quad \tilde{d}_{tmp} \leftarrow \tilde{d}(v, s) + c_{vw}(\tau_{\mathsf{start}} + \tilde{d}(v, s))$ ▷ time-dependency
12 $\quad\quad\quad$ **if** $\tilde{d}_{tmp} < \tilde{d}(w, s')$ **then**
13 $\quad\quad\quad\quad p(w, s') \leftarrow (v, s)$
14 $\quad\quad\quad\quad \tilde{d}(w, s') \leftarrow \tilde{d}_{tmp}$
15 $\quad\quad\quad\quad$ **if** (w, s') not in Q **then** ▷ insert
16 $\quad\quad\quad\quad\quad \pi_{w,s'} \leftarrow \pi(w, s')$
17 $\quad\quad\quad\quad\quad k(w, s') \leftarrow \tilde{d}(w, s') + \pi_{w,s'}$
18 $\quad\quad\quad\quad\quad$ insert (w, s') in Q
19 $\quad\quad\quad\quad$ **else** ▷ decrease
20 $\quad\quad\quad\quad\quad k(w, s') \leftarrow \tilde{d}(w, s') + \pi_{w,s'}$
21 $\quad\quad\quad\quad\quad$ decreaseKey (w, s') in Q

Preprocessing phase. Preprocessed distance data is used to guide the search algorithm. This data is produced as follows. First, as done for `ALT`, a set of landmarks $\ell \in \mathcal{L} \subset V$ is selected by using the *avoid* heuristic [Goldberg and Harrelson 2005]

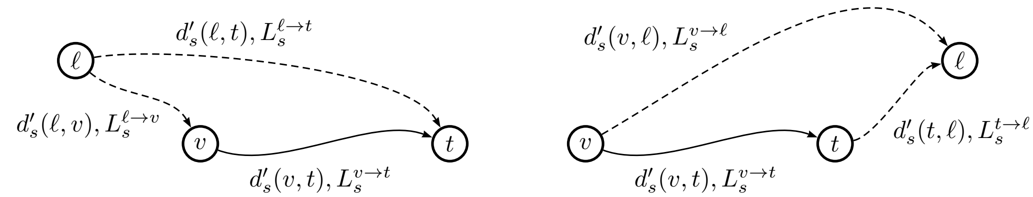


Fig. 5: Landmark distances for `SDALT`, $L_s^{i\to j}$ represents the regular language which constrains the shortest paths from (i, s) to (j, s'), $s' \in F$.

(Note that we calculated the landmarks on the walking network, as all our paths begin and end by walking). Then the costs of the shortest paths between all $v \in V$ and each landmark ℓ are determined. Here lies one of the major differences between `SDALT` and `ALT`: different from `ALT`, `SDALT` uses $\mathsf{D_{RegLC}}$ instead of `Dijkstra` to determine landmark distances and works on $G^{\times}$, instead of G. This way, it is possible to constrain the cost calculation by some regular languages which we derive from L_0. We refer to the travel time of the shortest path from (i, s) to (j, s'), $s' \in F$, which is *constrained* by the regular language $L_s^{i\to j}$, as *constrained distance* $d'_s(i, j)$ and to the constrained distances calculated during the preprocessing phase between nodes and landmarks as *constrained landmark distances*. $L_s^{i\to j}$ represents the regular language which constrains the shortest paths from (i, s) to (j, s'), for some $s' \in F$ and which has distance $d'_s(i, j)$. The constrained landmark distances are used to calculate the potential function $\pi(v, s)$ and to provide a lower bound on the distance $d'_s(v, t)$:

$$\pi(v, s) = \max_{\ell \in \mathcal{L}}(d'_s(\ell, t) - d'_s(\ell, v), d'_s(v, \ell) - d'_s(t, \ell)) \qquad (2)$$

Note that $d'_s(v, t)$ is constrained by $L_s^{v\to t} = L_0^s$. L_0^s is the regular expression of $\mathcal{A}_0^s$ which is equal to $\mathcal{A}_0$ except that the initial state s_0 is replaced by s. Intuitively, $L_s^{v\to t}$ represents the *remaining* constraints to be considered for the shortest path from an arbitrary node (v, s) to the target. In the next section, we provide different methods on how to choose $L_s^{\ell\to t}$, $L_s^{\ell\to v}$, $L_s^{v\to \ell}$, and $L_s^{t\to \ell}$ used to constrain the calculation of $d'_s(\ell, t)$, $d'_s(\ell, v)$, $d'_s(v, \ell)$, and $d'_s(t, \ell)$, for all $s \in S$ (see Figure 5).

Constrained landmark distances. The only open question now is how to produce good bounds to guide SDALT efficiently toward the target. This means, more formally, how to choose the regular languages $L_s^{\ell\to t}$, $L_s^{\ell\to v}$, $L_s^{v\to\ell}$, and $L_s^{t\to\ell}$ used to constrain the calculation of $d'_s(\ell,t)$, $d'_s(\ell,v)$, $d'_s(v,\ell)$, and $d'_s(t,\ell)$ in order that $d'_s(\ell,t) - d'_s(\ell,v)$ and $d'_s(v,\ell) - d'_s(t,\ell)$ are valid lower bounds for $d'_s(v,t)$ (see Figure 5 and Equation 2). A first answer gives Proposition 4.1:

PROPOSITION 4.1. *For all $s \in S$, if the concatenation of $L_s^{\ell\to v}$ and $L_s^{v\to t}$ is included in $L_s^{\ell\to t}$, then $d'_s(\ell,t) - d'_s(\ell,v)$ is a lower bound for the distance $d'_s(v,t)$. Similar, if $L_s^{v\to t} \circ L_s^{t\to\ell} \subseteq L_s^{v\to\ell}$ then $d'_s(v,\ell) - d'_s(t,\ell)$ is a lower bound for $d'_s(v,t)$.*

PROOF. (i) Suppose that $d'_s(\ell,t) - d'_s(\ell,v)$ is not a lower bound for the distance $d'_s(v,t)$ for some $s \in S$ and $L_s^{\ell\to v} \circ L_s^{v\to t} \subseteq L_s^{\ell\to t}$. We have $d'_s(\ell,t) - d'_s(\ell,v) > d'_s(v,t)$. Let $w_1 \in L_s^{\ell\to v}$ and $w_2 \in L_s^{v\to t}$ be the words produced by concatenating the labels on the arcs of the shortest path with cost $d'_s(\ell,v)$ and $d'_s(v,t)$, respectively. The fact that $d'_s(\ell,t) - d'_s(\ell,v)$ is greater than $d'_s(v,t)$ or $d'_s(\ell,v) + d'_s(v,t)$ is smaller than $d'_s(\ell,t)$ means that the word $w_1 \circ w_2$ is not included in $L_s^{\ell\to t}$ because $d'_s(\ell,t)$ is the cost of a shortest path. But this means $L_s^{\ell\to v} \circ L_s^{v\to t} \not\subseteq L_s^{\ell\to t}$. (ii) The same can be proven in a similar way for $d'_s(v,\ell) - d'_s(t,\ell)$. □

Proposition 4.1 is based on the observation that the distance of the shortest path from ℓ to t (v to ℓ) must not be greater than the distance of the shortest path from ℓ to v to t (v to t to ℓ). We now give three procedures to determine the regular languages $L_s^{\ell\to t}$, $L_s^{\ell\to v}$, $L_s^{v\to\ell}$, $L_s^{t\to\ell}$, which satisfy Proposition 4.1, in order to gain valid distance bounds for a generic node (v,s) of $G^\times$ (see also Table I):

Procedure 1. The language produced by Procedure 1 allows every combination of labels in Σ.

Procedure 2. The language produced by Procedure 2 depends on the state s of the node (v,s). It allows every combination of labels in Σ except those labels for which there is no longer any transition between states which are reachable from state s.

Table I: With reference to a generic RegLCSP where the shortest path is constrained by regular language L_0 ($\mathcal{A}_0 = (S, \Sigma, \delta, s_0, F)$) the table shows three procedures to determine the regular language to constrain the distance calculation for a generic node (v, s) of the product graph $G^{\times}$.

Procedure and regular language and/or NFA			
1		$L_s^{v\to\ell} = L_s^{t\to\ell} =$	$L_s^{\ell\to v} = L_s^{\ell\to t} = L_{\mathsf{proc1}} = \{\Sigma^*\}$
		$L_{\mathsf{proc1}} : \mathcal{A}_{\mathsf{proc1}} =$	$(\{s\}, \Sigma, \delta : \{s\} \times \Sigma \to \{s\}, s, \{s\})$
2		$L_s^{v\to\ell} = L_s^{t\to\ell} =$	$L_s^{\ell\to v} = L_s^{\ell\to t} = L_{\mathsf{proc2,s}} = \{\overrightarrow{\Sigma}(s, \mathcal{A}_0)^*\}$
		$L_{\mathsf{proc2,s}} : \mathcal{A}_{\mathsf{proc2,s}} =$	$(\{s\}, \overrightarrow{\Sigma}(s, \mathcal{A}_0), \delta : \{s\} \times \overrightarrow{\Sigma}(s, \mathcal{A}_0) \to \{s\}, s, \{s\})$
3	a)	$L_s^{\ell\to v} : \mathcal{A}_s^{\ell\to v} =$	$(S, \Sigma, \delta, s_0, s)$
	b)	$L_s^{\ell\to t} : \mathcal{A}_s^{\ell\to t} =$	$(S, \Sigma, \delta, s_0, F \cap \overleftarrow{S}(s, \mathcal{A}_0))$
	c)	$L_s^{v\to\ell} : \mathcal{A}_s^{v\to\ell} =$	$(S, \Sigma, \delta, s, F)$
	d)	$L_s^{t\to\ell} : \mathcal{A}_s^{t\to\ell} =$	$(S, \Sigma, \delta, F \cap \overleftarrow{S}(s, \mathcal{A}_0), F \cap \overleftarrow{S}(s, \mathcal{A}_0))$
	f)	[Optional] Clean $\mathcal{A}_s^{\ell\to v}, \mathcal{A}_s^{\ell\to t}, \mathcal{A}_s^{v\to\ell}, \mathcal{A}_s^{t\to\ell}$ from all transitions and states which are not reachable.	

Procedure 3. The language produced by Procedure 3 produces four distinct languages for a node (v, s) of G^x. To compute the bound $d'_s(\ell, t) - d'_s(\ell, v)$ the distance calculation of $d'_s(\ell, t)$ is limited by *all* constraints of $\mathcal{A}_0$, i.e., it is constrained by $\mathcal{A}_0$, and that of $d'_s(\ell, v)$ is constrained by the part of the constraints on $\mathcal{A}_0$ occurring *before* state s. Similar, to compute the bound $d'_s(v, \ell) - d'_s(t, \ell)$, the distance calculation of $d'_s(v, \ell)$ is limited by all constraints on $\mathcal{A}_0$ occurring *after* state s, and that of $d'_s(t, \ell)$ may only use labels on self-loops on final states. We modify the initial and final states and then remove from the automaton all transitions and states that are no longer reachable. If constrained shortest paths cannot be found because landmarks are not reachable from r or t, then it suffices to relax L_0 into a new language L'_0, e.g., by adding self-loops, and then apply Procedure 3 to L'_0.

Consider, e.g., a transportation network offering different modes of transportation. Procedures 1 and 2 are based on the intuition that modes of transportation that are excluded by L_0 (Procedure 1), or are excluded from a certain state s onward (Procedure 2), should not be used to compute the bounds. Procedure 3 goes a step further with the aim to incorporate into the preprocessed data not only the exclusion of modes of transportation but also specific information from L_0, i.e., having to maintain a certain sequence of modes of transportation, or limitations on the number of changes of modes of transportation which can be made during the trip.

5. LABEL SETTING SDALT

One condition that the A* and ALT algorithm work correctly is that reduced costs are positive, i.e., the potential function is feasible. In this section, we present three methods on how to produce feasible potential functions for SDALT. We call the version of SDALT which uses such potential functions Label Setting SDALT (lsSDALT) as it guaranties that when a node (v, s) is extracted from the priority queue (the node is settled), then it will not be visited again. Note that here *label* refers to the distance label of the algorithm and not to the labels on arcs, which indicate the mode of transportation.

Feasible potential functions. We present three methods on how to produce potential functions which are feasible: a basic method (bas), an advanced method (adv), and a specific method (spe). The *basic method (bas)* applies Procedure 1 to determine the constrained distance calculation. All nodes $(v, s), s \in S$ have the same lower bound on the distance to the target node. The *advanced method (adv)* applies Procedure 2 and thus produces different constrained landmark distances and consequently different lower bounds for nodes (v, s) with different states $s \in S$. Feasibility is guaranteed by using a slightly modified potential function:

$$\pi_{\mathsf{adv}}(v, s) = \max\{\pi(v, s_x) | s_x \in \overleftarrow{S}(s, \mathcal{A}_0)\}.$$

Finally, the third method, the *specific method (spe)*, applies Procedure 3. Potentials are feasible as proven by Proposition 5.1.

PROPOSITION 5.1. *By using the regular languages produced by applying Procedure 3 (see Table I) for the constrained landmark distance calculation for all nodes* (v, s)*, the potential function* $\pi(v, s)$ *in Equation 2 is feasible.*

PROOF.

If $\pi(v,s)$ is feasible, then the reduced cost c^{π}_{ij} is non-negative for all arcs of graph $G^{\times}$.
(i) Let us look at the potential function $\pi_1(v,s) = d'_s(\ell,t) - d'_s(\ell,v)$ first. In reference to the two arbitrary nodes (f,s_f) and (g,s_g) and arc (f,g), let us suppose $\pi(v,s)$ is not feasible and that the reduced cost is $c_{fg}(\tau) - \pi(f,s_f) + \pi(g,s_g) < 0$. We have that $c_{fg}(\tau) + (d'_{s_g}(\ell,t) - d'_{s_g}(\ell,g)) < (d'_{s_f}(\ell,t) - d'_{s_f}(\ell,f))$. Let us consider two cases.

(1) (case 1) If $s_f = s_g = s$, then $c_{fg}(\tau) + d'_s(\ell,f) < d'_s(\ell,g)$. But as $d'_s(\ell,g)$ is a shortest path and $s \in \delta(l,s)$, this is a contradiction.
(2) (case 2) If $s_g \neq s_f$ then as for (3b), $\mathcal{A}^{\ell\rightarrow t}_{s_f}$ includes $\mathcal{A}^{\ell\rightarrow t}_{s_g}$ we have $d'_{s_f}(\ell,t) \leq d'_{s_g}(\ell,t)$. So we have that $c_{fg}(\tau) + d'_{s_f}(\ell,f) < d'_{s_g}(\ell,g)$. But as, for rules (3a), $\mathcal{A}^{\ell\rightarrow g}_{s_g}$ includes all states and transitions of $\mathcal{A}^{\ell\rightarrow f}_{s_f}$ plus the transition $\delta(l,s_f) = s_g$, and as $d'_{s_g}(\ell,g)$ is a shortest path, this is again a contradiction.

(ii) Let us now look at the potential function $\pi_2(v,s) = d'_s(v,\ell) - d'_s(t,\ell)$. In reference to the two arbitrary nodes (f,s_f) and (g,s_g) and arc $a = ((f,s_f)(g,s_g))$ let us suppose $\pi(v,s)$ is not feasible and that $c_{fg}(\tau) - \pi(f,s_f) + \pi(g,s_g) < 0$. We have that $c_{fg}(\tau) + (d'_{s_g}(g,\ell) - d'_{s_g}(t,\ell)) < (d'_{s_f}(f,\ell) - d'_{s_f}(t,\ell))$. Let us consider two cases.

(1) (case 1) If $s_f = s_g = s$, then $c_{fg}(\tau) + d'_s(g,\ell) < d'_s(f,\ell)$. But as $d'_s(f,\ell)$ is a shortest path and $s \in \delta(l,s)$, this is a contradiction.
(2) (case 2) If $s_g \neq s_f$ then as for 3c and 3d, $\mathcal{A}^{t\rightarrow\ell}_{s_f}$ is included in $\mathcal{A}^{t\rightarrow\ell}_{s_g}$ we have $d'_{s_f}(t,\ell) \geq d'_{s_g}(t,\ell)$. Thus $c_{fg}(\tau) + d'_{s_g}(\ell,g) < d'_{s_f}(\ell,f)$. But as, for (3c), $\mathcal{A}^{f\rightarrow\ell}_{s_f}$ includes all states and transitions of $\mathcal{A}^{g\rightarrow\ell}_{s_g}$ plus the transition $\delta(l,s_f) = s_g$, and as $d'_{s_f}(f,\ell)$ is a shortest path, this again is a contradiction.

Thus $\pi_1(v,s) = d'_s(\ell,t) - d'_s(\ell,v)$ is feasible and $\pi_2(v,s) = d'_s(v,\ell) - d'_s(t,\ell)$ is feasible. Hence, $\pi(v,s) = \max_{\ell\in\mathcal{L}}(d'_s(\ell,t) - d'_s(\ell,v), d'_s(v,\ell) - d'_s(t,\ell))$ is feasible. □

For an example of how these three methods are applied, see Figure 6. We call the versions of `lsSDALT` which apply these three methods `bas_ls`, `adv_ls`, and `spe_ls`. We introduce a fourth *standard* version called `std` to evaluate `lsSDALT`. It does not constrain the landmark distance calculation by any regular language and can be seen as the application of plain `uniALT` to $\mathsf{D}_{\mathsf{RegLC}}$.

Correctness. In the case the potential function $\pi(v, s)$ is feasible, all characteristics that we discussed for `uniALT` also hold for `SDALT`, which can be seen as an A^* search on the product graph $G^\times$ which uses the potential function $\pi(v, s)$. Hence, `lsSDALT` is correct and always terminates with the correct constrained shortest path.

PROPOSITION 5.2. *If solutions exist,* `lsSDALT` *finds a shortest path.*

Complexity and memory requirements . Complexity of `lsSDALT` is equal to the complexity of $\mathsf{D}_{\mathsf{RegLC}}$, which is equal to the complexity of `Dijkstra` on the product graph $G^\times$: $O(m \log n)$; $m = |A||S|^2$ and $n = |V||S|$ are the number of arcs and nodes of $G^\times$. The amount of memory needed to hold the distance data computed during the preprocessing phase varies in function of the chosen method. Memory requirements for `std` and `bas_ls` are proportional to $|\mathcal{L}| \times |V|$. They are up to an additional factor $|S|$ and $4 \times |S|$ higher for `adv_ls` and `spe_ls`, respectively.

Calculation of potential function. Note that the calculation of the potential function introduces a strong algorithmic overhead for `lsSDALT`. The number of calculated bounds to compute the potential function $\pi(v, s)$ varies in function of the chosen method. The number of calculated bounds grows linearly to the number of relaxed arcs for `bas_ls` and `spe_ls`. For `adv_ls`, the number of calculated bounds in worse case scenario is an additional factor $|S|$ higher.

6. LABEL CORRECTING SDALT

The algorithm `lsSDALT` works correctly only if reduced arc costs are non-negative. It turns out, however, that by violating this condition often tighter lower bounds can be produced and required memory space can be reduced. At least in our scenario, this compensates the additional computational effort required to remedy the disturbing effects of the use of negative reduced costs on the underlying `Dijkstra` algorithm and in addition results in shorter query times and lower memory requirements. This is why we propose a version of `SDALT`, which can handle negative reduced costs. The major impact of this is that *settled* nodes may be re-inserted into the priority queue for re-examination (*correction*). In our setting, the number of arcs with non-negative reduced arc costs is limited and we can prove that the algorithm may stop once the target node is extracted from the priority queue. Note that in our scenario there are no negative cycles as arc costs are always non-negative. We name the new algorithm Label Correcting `SDALT` or shortly `lcSDALT`.

Query. The algorithm `lcSDALT` is similar to `lsSDALT` with the difference being that it allows re-insertion of a node (v, s) into the priority queue Q. Note that it is necessary to calculate the potential of a node (v, s) only the first time it is inserted in Q (see Algorithm 2, the missing lines are the same as in Algorithm 1).

Algorithm 2 Pseudo-code `lcSDALT`

```
if (w, s') not in Q and never visited then          ▷ insert
    π_{w,s'} ← π(w, s')
    k(w, s') ← d̃(w, s') + π_{w,s'}
    insert (w, s') in Q
else if (w, s') not in Q then                        ▷ re-insert
    k(w, s') ← d̃(w, s') + π_{w,s'}
    insert (w, s') in Q
else                                                 ▷ decrease
    k(w, s') ← d̃(w, s') + π_{w,s'}
    decreaseKey (w, s') in Q
```

Correctness. The algorithm `lcSDALT` is based on D_{RegLC} and `uniALT`. It suffices to prove that the algorithm may stop as soon as the target node (t, s'), $s' \in F$ is extracted from the priority queue (see Lemma 6.1 and Proposition 6.2). Note that $\pi(t, s') = 0$, $s' \in F$, that $d^*(v, s)$ is the distance of the shortest path from (r, s_0) to (v, s), and that there are no negative cycles as arc costs are always non-negative.

LEMMA 6.1. *The priority queue always contains a node (i, s') with key $k(i, s') = d^*(i, s') + \pi(i, s')$ which belongs to the shortest path from (r, s_0) to (t, s'') where $s'' \in F, s' \in S$.*

PROOF. Let $q^* = (p_1 = (r, s_0), \ldots, p_m = (t, s''))$ be the shortest path from (r, s_0) to (t, s'') on $G^\times$ (constrained by L_0). At the first step of the algorithm, node $p_1 = (r, s_0)$ is inserted in the priority queue with key $k(r, s) = d^*(r, s) + \pi(r, s) = \pi(r, s)$. When node p_n with $k(i, s) = d^*(i, s) + \pi(i, s)$ for some $n \in \{1, \ldots, m\}$ is extracted from the priority queue, at least one new node $p_{n+1} = (j, s')$ with $\tilde{d}(j, s') = d^*(j, s') = d^*(i, s) + c_{(i,s)(j,s')}(\tau)$ is inserted in the queue by lines 18, 21, 24. □

PROPOSITION 6.2. *If solutions exist,* `lcSDALT` *finds a shortest path.*

PROOF. Let us suppose that a node (t, s), where $s \in F$, is extracted from the priority queue but its distance label is not optimal, so $\tilde{d}(t, s) \neq d^*(t, s)$. Node (t, s) has key $k(t, s_f) = \tilde{d}(t, s_f) + \pi(t, s) \neq d^*(t, s)$. By Lemma 6.1, this means that there exists some node (i, s') in the priority queue on the shortest path from (r, s_0) to (t, s) which has not been settled because its key $k(i, s') > k(t, s)$. This means $k(i, s') = d^*(i, s') + \pi(i, s') > \tilde{d}(t, s) + \pi(t, s) = k(t, s)$, which is a contradiction. □

Constrained landmark distances. The methods (bas), (adv), and (spe) may be used with `lcSDALT`. However, `lcSDALT` produces a slight overhead in respect to `lsSDALT` as it unnecessarily checks if newly inserted nodes in Q have previously been extracted from the priority queue (line 18). Now we present two new methods which can only be used with `lcSDALT`, as reduced costs may be negative: an adapted version of (adv) which we call (adv_{lc}) and an adapted version of (spe) which we call (spe_{lc}). We name the versions of `lcSDALT` which apply these two methods `adv_lc` and `spe_lc`.

- *(adv_{lc}).* Equal to (adv), this method applies Procedure 2 to all nodes (v, s) of $G^\times$. Different to (adv) it uses Equation 2 as potential function and thereby considerably reduces the number of potentials to be calculated.
- *(spe_{lc}).* The method (spe) applies the regular languages constructed by applying Procedure 3 for *each* state of L_0. This is space-consuming and bounds for nodes with certain states may be worse than those produced by Procedure 2. This is why we introduce a more flexible new method (spe_{lc}) which provides the possibility to freely

choose for each state between the application of Procedure 2 and Procedure 3. This also provides a trade-off between memory requirements and performance improvement as Procedure 2 consumes less space than Procedure 3. The right calibration for a given L_0 and the choice of whether to use Procedure 2 or 3 is determined experimentally. See Figure 6 for an example.

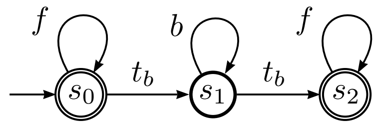
Complexity and memory requirements. Complexity of `lcSDALT` when a feasible potential function is used is equal to the complexity of `lsSDALT`. If the potential function is non-feasible the key of a node extracted from the priority queue could not be minimal, hence already extracted nodes might have to be *re-inserted* into the priority queue at a later point and re-examined (*corrected*). The algorithm `lcSDALT` can handle this but in this case its complexity is similar to the complexity of the Bellman-Ford algorithm (plus the time needed to manage the priority queue): $O(mn \log n)$; $m = |A||S|^2$ and $n = |V||S|$ are the number of arcs and nodes of $G^{\times}$. The amount of memory needed to hold the distance data computed during the preprocessing phase for `spe_lc` and `adv_ls` in worse case is equal to `spe_ls` and `adv_ls`, respectively.

7. BI-DIRECTIONAL SDALT

In this section, we discuss the bi-directional version of the `SDALT` algorithm. We introduce the approaches for bi-directional search for `Dijkstra` and `ALT` described in [Pohl 1971; Nannicini et al. 2008; Goldberg and Harrelson 2005] and we describe how we adapted them to `SDALT`.

Query. In general, bi-directional `SDALT` (`biSDALT`) works as follows. It alternates between running a `lsSDALT` query from source (r, s_0) to target $(t, s'), s' \in F$ (forward search) and a second `lsSDALT` query from all $(t, s'), s' \in F$ to (r, s_0) (backward search). Note that the backward search works on the *backward automaton* (see Figure 1 for an example).

The potential function for the backward search, π_B (see Figure 7), is a slight modification of the potential function for the forward search, π_F (equal to Equation 2):



(a) $\mathcal{A}_0$: Automaton allows walking (label f) and biking (label b), transitions with label t_b model the transfer between walking and biking. Once the bike is discarded (state s_2) it may not be used again. Automaton has states $S = \{s_0, s_1, s_2\}$, initial state s_0, final states $F = \{s_0, s_2\}$, and labels $\Sigma = \{f, b, t_b\}$.

$$L_0 : f^*|(f^* t_b b^* t_b f^*)$$

(b) $\mathcal{A}_0$ expressed as a regular expression. The vertical bar | represents the boolean *or* and the asterisk * indicates that there are zero or more of the preceding element.

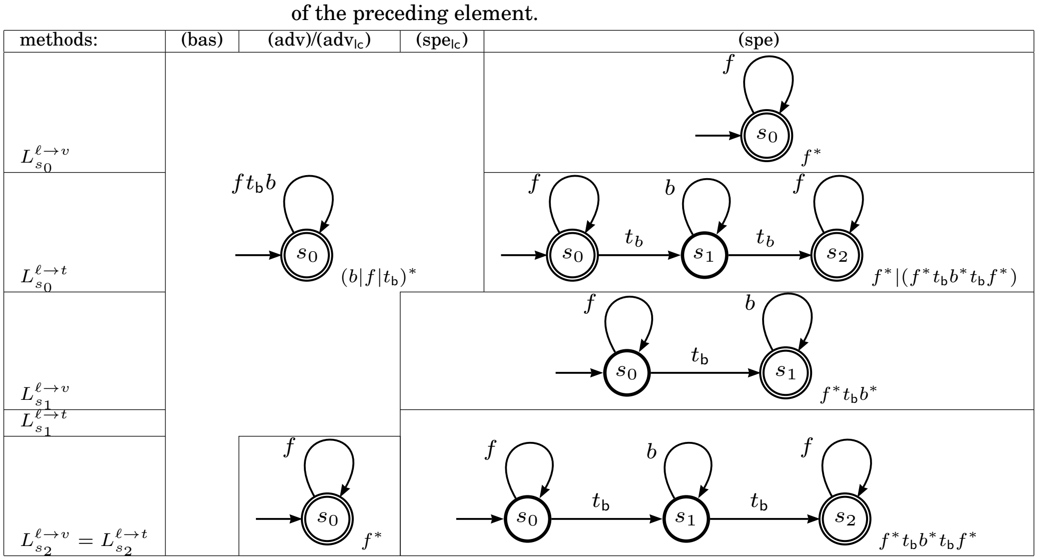


Fig. 6: Example of a regular language L_0 and its representation as an automaton (Figure 6a) and regular expression (Figure 6b). The table lists the languages used to constrain the landmark distance calculation for the different methods. E.g., for (bas) all $(b|f|t)^*$, for (adv): $L_{s_0}^{\ell\to v} = L_{s_0}^{\ell\to t} = L_{s_1}^{\ell\to v} = L_{s_1}^{\ell\to t} : (b|f|t)^*$, $L_{s_2}^{\ell\to v} = L_{s_2}^{\ell\to t} : f^*$.

$$\pi_F(v, s) = \max_{\ell\in\mathcal{L}}(d'_s(\ell, t) - d'_s(\ell, v), d'_s(v, \ell) - d'_s(t, \ell)) \tag{3}$$

$$\pi_B(v, s) = \max_{\ell\in\mathcal{L}}(d'_s(\ell, v) - d'_s(\ell, r), d'_s(r, \ell) - d'_s(v, \ell)) \tag{4}$$

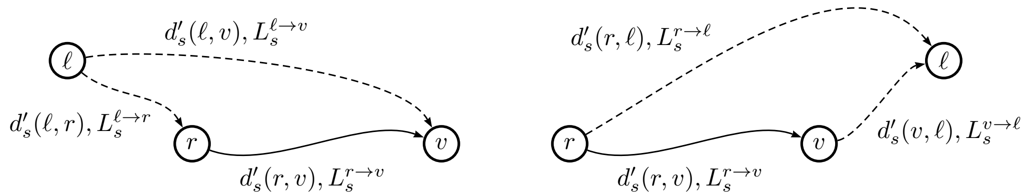


Fig. 7: Landmark distances for backward search.

As π_F and π_B are not *consistent* (i.e., $\pi_F + \pi_B \neq$ const.), we have no guarantee that the shortest path is found when the two searches first meet [Goldberg and Harrelson 2005]. We discuss the non time-dependent and the time-dependent case below.

Non time-dependent case. For networks without time-dependent arc costs, the authors of [Pohl 1971] propose a symmetric lower bounding algorithm. When applied to the product graph $G^\times$, it works as follows. Every time the forward or backward search relaxes a node (v, s) which has already been relaxed by the opposite search, it checks whether the cost of the path $(r, s_0) - (v, s) - (t, s_f)$ is smaller than that of the best shortest path (whose cost is μ) found so far. If this is the case, we update μ. The search stops when one of the searches is about to settle a node (v, s) with key $k(v, s) \geq \mu$, or when the priority queues of both searches are empty. The authors of [Goldberg and Harrelson 2005] enhance this algorithm further: when either of the searches relaxes a node (v, s) which has been settled by the opposite search, then the search does nothing with (v, s) (pruning).

Time-dependent case. For networks with time-dependent arc costs, the algorithm becomes more complicated. The symmetric lower bounding algorithm may stop as soon as a node (v, s) with $k(v, s) \geq \mu$ is found, because for every settled node the backward search produces correct shortest path distances to the target. In the time-dependent scenario, arc costs depend on the arrival time at the arc. But for the backward search the exact starting time from the target is not known. The authors of [Nannicini et al. 2008] propose to use the minimum weight arc cost for the backward search and to use the backward query only to restrict the search space of the forward query. Their algorithm is similar to the symmetric lower bounding algorithm. Again μ is checked and recorded at every iteration, μ is the sum of the costs of paths $(r, s_0) - (v, s)$ (forward search) and $(v, s) - (t, s')$, $s' \in F$ (backward search). Note that the cost of path $(v, s) - (t, s')$, is re-evaluated by considering the correct time-dependent arc costs. When either search settles a node (v, s) with key $k(v, s) \geq \mu$ then only the backward search stops. The forward search continues but only visits nodes already settled by the backward search. Pruning applies only to the backward search. The authors of [Nannicini et al. 2008] prove correctness and propose the following two improvements:

Approximation. The algorithm produces approximate shortest paths of factor K if the backward search is stopped as soon as a node (v, s) with $k(v, s) \leq K \cdot \mu$ is found.

Tight Potential Function. In order to enhance the potential function of the backward search, information from the forward search is used. The potential function for the backward search becomes

$\pi^*_B(w, s) = \max\{\pi_B(w, s), \tilde{d}(v', s') + \pi_F(v', s') - \pi_B(w, s)\}$.

At predefined checkpoints, i.e., whenever the current distance exceeds $\frac{K \cdot \pi_F(r, s_0)}{10}$, $k \in \{1, \ldots, 10\}$, the node (v', s'), $s' \in S$, $v' \in V$, that was settled most recently by the forward search is memorized. At the checkpoints the backward queue is flushed and all the keys are recalculated. This guarantees feasibility.

We include these improvements in our algorithm and call this new version of SDALT $\texttt{bi}_{v0}$. As time-dependent arcs are limited in our scenario, depending on the regular language L_0, we propose a first variation of $\texttt{bi}_{v0}$ that combines the symmetric lower-bounding algorithm with the time-dependent version. To do this, we set a flag on nodes visited by the backward search indicating that the node has been reached *exclusively* by using time-independent arcs. If a node with flag=1 is reached by the forward search the termination condition of the symmetric lower-bound algorithm applies. We call this version of the algorithm $\texttt{bi}_{v1}$. Note that the bi-directional algorithm only works correctly (pruning of backward search, approximation, tight potential function) if both π_B and π_F are feasible. However, whenever a node already settled by the backward search is visited by the forward search, the potential function π_F can be enhanced by using the distance already calculated by the backward search. In the second variation of $\texttt{bi}_{v0}$, which we call $\texttt{bi}_{v2}$, as soon as the backward search stops we switch to `lcSDALT` for the forward search and use the potential $\pi_F(v,s) = \tilde{d}(v,s)$ for every visited node; $\tilde{d}(v,s)$ is the distance label for node (v,s) of the backward search. This improves potentials and prevents the computation of bounds. However, this new potential function is not feasible and therefore the forward search has to switch lo `lcSDALT`.

Constrained landmark distances and potential function. The potential function for the backward search is constructed semi-symmetrically to the potential function of the forward search. We want to choose the regular languages for $L_s^{\ell\to v}$, $L_s^{\ell\to r}$, $L_s^{r\to\ell}$, $L_s^{v\to\ell}$ used to constrain the calculation of $d'_s(\ell,v)$, $d'_s(\ell,r)$, $d'_s(r,\ell)$, $d'_s(v,\ell)$ in order that $d'_s(\ell,v)-d'_s(\ell,r)$, $d'_s(r,\ell)-d'_s(v,\ell)$ be valid lower bounds for $d'_s(r,v)$ (see Figure 7). Similar to Proposition 4.1, the following Proposition 7.1 gives first indications.

PROPOSITION 7.1. *For all $s \in S$, if the concatenation of $L_s^{\ell\to r}$ and $L_s^{r\to v}$ is included in $L_s^{\ell\to v}$ ($L_s^{\ell\to r} \circ L_s^{r\to v} \subseteq L_s^{\ell\to v}$), then $d'_s(\ell,v) - d'_s(\ell,r)$ is a lower bound for the distance $d'_s(r,v)$. Similarly, if $L_s^{r\to v} \circ L_s^{v\to\ell} \subseteq L_s^{r\to\ell}$ then $d'_s(r,\ell) - d'_s(v,\ell)$ is a lower bound for $d'_s(v,t)$.*

Table II summarizes three procedures on how to determine $L_s^{\ell\to v}$, $L_s^{\ell\to r}$, $L_s^{r\to\ell}$, $L_s^{v\to\ell}$ for the backward search. The *basic method* (bas_B) applies Procedure 1B to determine the constrained distance calculation and is equal to Procedure 1. The *advanced method* (adv_B) applies procedure 2B and thus produces different constrained landmark distances for nodes with different states. Feasibility is again guaranteed by using a slightly modified potential function:

$$\pi_{\text{adv\_B}}(v,s) = \max\{\pi(v,s_x) | s_x \in \overleftarrow{S}(s,\mathcal{A}_0)\}$$

Finally, the *specific method* (spe_B) applies procedure 3B.

Note that when using any of the methods, (bas), (adv), or (spe), for the forward search, any of the methods defined for the backward search, (bas_B), (adv_B), or (spe_B)

Table II: With reference to a generic RegLCSP where the shortest path is constrained by regular language L_0 ($\mathcal{A}_0 = (S, \Sigma, \delta, s_0, F)$) the table shows three procedures to determine the regular language to constrain the distance calculation for a generic node (v, s) of the product graph $G^\times$ for the backward query.

proc.		regular language and/or NFA	
1B		equal to Procedure 1	
2B		$L_s^{\ell\to v} = L_s^{\ell\to r} =$	$L_s^{r\to\ell} = L_s^{v\to\ell} = L_{\mathsf{proc2,s}} = \{\overleftarrow{\Sigma}(s, \mathcal{A}_0)^*\}$
		$L_{\mathsf{proc2,s}} : \mathcal{A}_{\mathsf{proc2,s}} =$	$(\{s\}, \overleftarrow{\Sigma}(s, \mathcal{A}_0), \delta : \{s\} \times \overleftarrow{\Sigma}(s, \mathcal{A}_0) \to \{s\}, s, \{s\})$
3B	a)	$L_s^{\ell\to r} : \mathcal{A}_s^{\ell\to r} =$	$(S, \Sigma, \delta, s_0, s_0)$
	b)	$L_s^{\ell\to v} : \mathcal{A}_s^{\ell\to v} =$	$(S, \Sigma, \delta, s_0, s)$
	c)	$L_s^{r\to\ell} : \mathcal{A}_s^{r\to\ell} =$	$\mathcal{A}_0$
	d)	$L_s^{v\to\ell} : \mathcal{A}_s^{v\to\ell} =$	$(S, \Sigma, \delta, s, F \cap \overleftarrow{S}(s, \mathcal{A}_0))$
	e)	[Optional] Clean $\mathcal{A}_s^{\ell\to r}, \mathcal{A}_s^{\ell\to v}, \mathcal{A}_s^{r\to\ell}, \mathcal{A}_s^{t\to\ell}$ of all transitions and states which are not reachable	

can be used. We provide experimental data for the combinations (bas)-(bas$_{\mathsf{B}}$), (adv)-(adv$_{\mathsf{B}}$), and (spe)-(spe$_{\mathsf{B}}$), and called the algorithms `bas-bi`$_{\mathsf{vx}}$, `adv-bi`$_{\mathsf{vx}}$, and `spe-bi`$_{\mathsf{vx}}$, respectively, where x $\in \{1, 2, 3\}$. Preliminary results for the other combinations did not differ greatly, however, it shall be noted that they provide the possibility to further balance the trade-off between memory requirements and performance improvement.

Correctness. The variants of `biSDALT` are based on the principles outlined in [Nannicini et al. 2008; Goldberg and Harrelson 2005] and Section 6.

PROPOSITION 7.2. *If solutions exist, the variants of `biSDALT` find a shortest path.*

Memory requirements. Memory requirements to hold preprocessing data for `bas-bi`$_{\mathsf{vx}}$ and `spe-bi`$_{\mathsf{vx}}$ are equal to memory requirements of `bas_ls` and `spe_ls`, because of symmetry in the calculation of the potential function for forward and backward search. For `adv-bi`$_{\mathsf{vx}}$ memory requirements in worst case are a factor 2 higher as memory requirements for `adv_ls`.

8. EXPERIMENTAL RESULTS

The algorithms are implemented in C++ and compiled with GCC 4.1. A binary heap is used as priority queue. Similar to the `ALT` algorithm presented in [Nannicini et al.

2008], periodical additions of landmarks (max. 6 landmark) take place. Experiments are run on an Intel Xeon (model W3503), clocked at 2.4 Ghz, with 12 GB RAM.

For the evaluation of the versions of SDALT two multi-modal transportation networks have been used: IDF (Ile-de-France) and NY (New York City). Note that we did not consider real time traffic information, perturbations on public transportation, or information about available rental cars or bicycles at rental stations. However, SDALT is robust to variations in the graph and so this information can be included as long as minimum travel times do not change.

The network IDF is based on road and public transportation data of the French region Ile-de-France (which includes the city of Paris and its suburbs). It consists of four layers: bicycle, walking, car, and public transportation. Each arc has exactly one associated label, e.g., f for arcs representing foot paths, p_r for rail tracks, c_t for toll roads. Each layer is connected to the walking layer through transfer arcs. See the schematic representation in Figure 8. The cost of transfer arcs represent the time needed to transfer from one layer to another (e.g., the time needed to unchain and mount a bicycle). The graph consists of circa 3.9M arcs and 1.2M nodes. Dimensions of the graph and a list of all used labels are given in Table III. See [Pyrga et al. 2007] for more information about graph models of a multi-modal network and time-dependency.

Data of the public transportation network has been provided by STIF[2]. It includes geographical information, as well as timetable data on bus lines, tramways, subways and regional trains. We use the realistic time-dependent model as presented in [Pyrga et al. 2007]. The public transportation layer is reachable from the walking layer through transfer arcs (label t_p) which connect each public transportation station (metro stations, bus stops, etc.) to the nearest node from the walking layer.

Data for the car layer is based on road and traffic information provided by Mediamobile[3]. Arc labels and costs (travel times) are set according to the road type (motorway, side street, etc). Circa 15% of the road arcs have a time-dependent cost function to rep-

[2]Syndicat des Transports IdF, www.stif.info, data for scientific use (01/12/2010)
[3]www.v-trafic.fr, www.mediamobile.fr

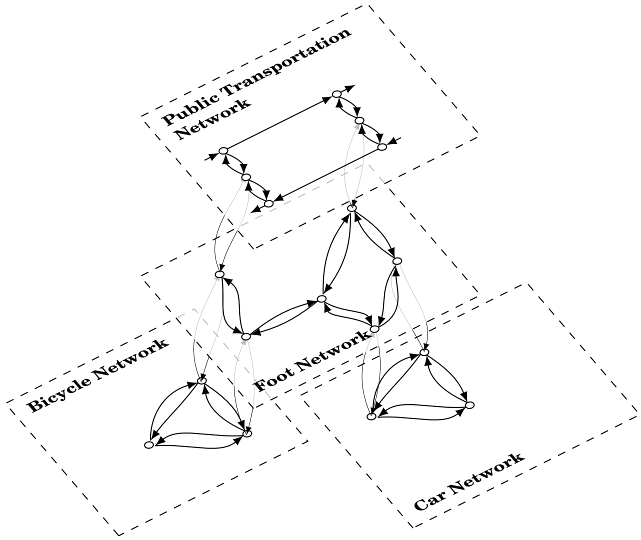


Fig. 8: Multi-modal graph.

resent changing traffic conditions throughout the day. Transfers from the car layer to the walking layer are possible at uniformly distributed transfers arcs (label t_c) between close nodes of the two layers (except for nodes belonging to low road classes, i.e., highways, motorways) or, if a rental car is used, at car rental stations[4] (label t_a). Car rental stations are located in Paris and its surroundings and cars are always assumed to be available.

The walking as well as the bicycle layer are based on road data (walking paths, cycle paths, etc.) extracted from geographical data freely available from OpenStreetMap[5]. Arc cost equals walking or biking time (pedestrians 4km/h, bikers 12km/h). Arcs are replicated and inserted in each of the layers if both walking and biking are possible. Rental bicycle stations are located mostly in the area of Paris[6], they serve as connection points between the walking layer and the bicycle layer, as rental bicycles have to be picked up at and returned to bicycle rental stations (label t_v). We suppose that rental bicycles are always available. The private bicycle layer is connected to the walking layer at common street intersections (label t_b).

[4]Autolib', www.autolib.eu
[5]See www.openstreetmap.org
[6]Vélib', www.velib.paris.fr

Table III: Ile-de-France (IDF) transportation network: sizes

layer	nodes	arcs	labels
walking	275 606	751 144	f (all arcs except 2x20 arcs with labels z_{f_1} and z_{f_2})
public transportation	109 922	292 113	p_b (bus, 72 512 arcs), p_m (metro, 1 746), p_t (tram, 1 746), p_r (train, 8 309), p_c (connection between stations, 32 490), p_w (walking paths inside stations, 176 790 (omitted in automata and regular expressions for simplicity)), time-dependent 82 833
bicycle	250 206	583 186	b
car	613 972	1 273 170	c_t (toll roads, 3 784), c_f (fast roads, 16 502), c_p (paved roads except toll and fast roads, 1 212 957), c_u (unpaved roads, 27 979), 2x20 arcs with labels z_{c_1} and z_{c_2}, time-dependent 188 197
transfers	-	1 109 922	access to car layer by private car t_c (493 601) and by rental car at rental car stations t_a (524), access to bike layer by rental bike t_v (1 198) and by private bike t_b (493 601), access to public transportation at stations t_p (38 848)
Tot	1 249 706	3 980 887	time-dependent arcs 271 030 (7 687 204 time points)

Table IV: New York (NY) transportation network: sizes

layer	nodes	arcs	labels
walking	104 737	317 888	f (all arcs except 2x20 arcs with labels z_{f_1} and z_{f_2})
public transportation	43 856	78 932	p_b (bus, 23 784 arcs), p_m (metro, 1 702), p_t (train, 348), p_c (connection between stations, 142), p_w (walking paths inside stations, 52 956 (omitted in automata and RE)), time-dependent arcs 25 834
bicycle	104 737	317 888	b
car	100 529	276 521	all paved roads c_p except 2x20 arcs with labels z_{c_1} and z_{c_2} and all non-time-dependent
transfers	-	442 796	access to car layer by private car t_c (201 058), access to bike layer by private bike t_v (209 474), access to public transportation at stations t_p (32 264)
Tot	353 859	1 436 141	time-dependent arcs 25 834 (3 572 498 time points)

The NY network is composed of data of the road and public transportation system of New York City. It consists of four layers: bicycle, walking, car, and public transportation. It is constructed in the same way as the graph of Ile-de-France and we use the same labels to mark modes of transportation. We use geographical data from OpenStreetMap for the car, walking, and cycling layers. The public transportation layer is based on data freely available from the Metropolitan Transportation Authority[7]. See Table IV for detailed information.

In addition, in both graphs, we introduced two times twenty arcs with labels z_{f_1} and z_{f_2} between nodes of the foot layer, and two times twenty arcs with labels z_{c_1} and z_{c_2} between nodes of the car layer. They represent arcs close to locations of interest, and are used to simulate the problem of reaching a target and in addition passing by any pharmacy, grocery shop, etc.

[7]MTA, www.mta.info/developers (01/08/2012)

Test instances. To test the performance of the algorithms, we recorded runtimes for 500 test instances for 26 RegLCSP scenarios. Scenarios have been chosen with the intention to represent real-world queries, which may arise when looking for constrained shortest paths on a multi-modal transportation network. 11 scenarios have simple constraints which only exclude modes of transportation. The remaining 15 scenarios have more complex constraints (constraints on number of changes, sequence of modes of transportation, e.g., bicycle followed by public transportation followed by rental bicycles). These scenarios have been derived from six base-automata (I, II, III, IV, V, VI) by varying the involved modes of transportation, see Figures 9, 11, 13, 15, 17, and 19. The regular expressions of all 26 scenarios can be found in Tables V and VII.

Source node r, target node t, and start time τ_{start} are picked at random, r and t always belong to the walking layer. Thus all paths start and end by walking. For all scenarios we use the same 32 landmarks determined by using the *avoid* heuristic [Goldberg and Harrelson 2005]. The determination of the landmarks took approximately 3 minutes in our scenario. Landmarks are calculated and placed exclusively on the walking layer as all paths of the scenarios start and end by walking. The calculation of the constrained landmark distances involves the execution of one backward and one forward $\mathtt{D}_{\mathsf{RegLC}}$ search from each landmark to all other nodes (one-to-all) for each regular language determined by the different methods (bas), (adv), (spe), etc. (For (bas) only one regular language, for (adv) up to $|S|$ regular languages etc. See Sections 5 and 6.) Preprocessing on network IDF takes less than 90s for a single regular language and up to 8m for all the regular languages determined by the chosen method (20s and 1m40s for the network NY, which is of a smaller size). See Tables VIII and IX for preprocessing times and sizes of preprocessed data for all scenarios.

For each scenario, we compare average runtimes of the different variations of SDALT (see Table VI) with $\mathtt{D}_{\mathsf{RegLC}}$ [Barrett et al. 2000] and std (which is based on the goal directed search algorithm go presented in [Barrett et al. 2008]). To the best of our knowledge, no other comparable methods on finding constrained shortest paths on multi-modal networks exist in the literature. A direct comparison to the methods pre-

Table V: Regular expressions of test scenarios for experimental evaluation.

NFA	regular expression
Ia	$f^* \mid (f^* t_a (c_t \mid c_f \mid c_p \mid c_u)^* t_a f^*$
Ib	$f^* \mid (f^* t_c (c_t \mid c_f \mid c_p \mid c_u)^* t_c f^*$
IIa	$(f \mid t_a \mid c_t \mid c_f \mid c_p \mid c_u)^* z (f \mid t_a \mid z \mid c_t \mid c_f \mid c_p \mid c_u)^*$
IIb	$(f \mid t_c \mid c_t \mid c_f \mid c_p \mid c_u)^* z (f \mid t_c \mid z \mid c_t \mid c_f \mid c_p \mid c_u)^*$
IIIa	$(t_a \mid c_t \mid c_f \mid c_p \mid c_u)^* z_{f1} (b \mid f \mid t_b)^* z_{f2} f^*$
IIIb	$(t_a \mid c_p \mid c_u)^* z_{f1} (b \mid f \mid t_b)^* z_{f2} f^*$
IIIc	$(t_p \mid p_b \mid p_m \mid p_r \mid p_t)^* z_{f1} (b \mid f \mid t_b)^* z_{f2} f^*$
IIId	$(t_p \mid p_m \mid p_t)^* z_{f1} (b \mid f \mid t_b)^* z_{f2} f^*$
IVa	$(t_b b^* t_b \mid f)(f^* \mid f^* t_p p t_p (b \mid f \mid t_v)^*$
IVb	$(t_b b^* t_b \mid f)(f^* \mid f^* t_p (p_c \mid p)^* t_p (b \mid f \mid t_v)^*$
IVc	$(t_b b^* t_b \mid f)(f^* \mid f^* t_p (p_m \mid p_t)^* t_p (b \mid f \mid t_v)^*$
Va	$(b \mid f \mid t_b)^* \mid (b \mid f \mid t_b)^* ((t_a c^* t_a) \mid (t_p p^* t_p) \mid (t_p p^* p_c p^* t_p))(b \mid f \mid t_v)^*$
Vb	$(b \mid f \mid t_b)^* \mid (b \mid f \mid t_b)^* ((t_a c^* t_a) \mid (t_p (p_m \mid p_t)^* t_p) \mid (t_p (p_m \mid p_t)^* p_c (p_m \mid p_t)^* t_p))(b \mid f \mid t_v)^*$
VIa	$(b \mid f \mid p_m \mid p_t \mid t_p \mid t_b)^* (z_f \mid (t_a c^* z_c (c \mid z_c)^* t_a)(f \mid p_m \mid p_t \mid t_p \mid z_f)^*$
VIb	$(b \mid f \mid t_b)^* (z_f \mid (t_a c^* z_c (c \mid z_c)^* t_a)(f \mid z_f)^*$

Table VI: List of the different variants of the SDALT algorithm.

lsSDALT	lcSDALT	biSDALT		
bas_ls	-	bas_bi$_{v0}$	bas_bi$_{v1}$	bas_bi$_{v2}$
adv_ls	adv_lc	adv_bi$_{v0}$	adv_bi$_{v1}$	adv_bi$_{v2}$
spe_ls	spe_lc	spe_bi$_{v0}$	spe_bi$_{v1}$	spe_bi$_{v2}$

sented in [Rice and Tsotras 2010] and [Dibbelt et al. 2012] is not possible as they do not consider time-dependent arc costs on the road network and are only applicable to specific scenarios (further discussed in Section 9).

8.1. Discussion

Simple constraints. For a preliminary evaluation of the impact of the use of various modes of transportation, we first run tests for scenarios with simple regular expressions which just exclude modes of transportation but do not impose any other constraints. We solely applied bas_ls as the automaton has only one state. Average runtimes are listed in Table VII. Speed-ups in respect to D_{RegLC} range from a speed-up of a factor of 1.5 to a factor of 40 (up to a factor of 55 with approximation). We observed that bas_ls is always faster than D_{RegLC} and std, and that the faster the modes of transportation which are excluded, the higher the speed-up. Furthermore, time-dependency has a negative impact on runtime, especially on bi-directional search. This is probably due to the fact that bounds are calculated by using the minimum weight cost function.

Table VII: Experimental results for scenarios with simple regular languages: no constraints other than exclusion of modes of transportation (average runtimes in milliseconds, preprocessing time (pre) in seconds). Size of preprocessed data for scenarios on IDF and NY is 306 MB and 86 MB, respectively.

regular expression	allowed modes of transportations	net[b]	pre[c] [s]	D_{RegLC} [ms]	`std` [ms]	`bas_ls` [ms]	`bas_bi`$_{v0}$ [ms]	10%[a] [ms]
$(f)^*$	only foot	IDF	19s	88	117	5	***4**	[illegible]
		NY	6s	27	38	***1.6**	2.4	1.
$(b\|f\|t_b)^*$	bike	IDF	32s	199	248	13	9	*
		NY	12s	75	96	5.4	3.2	***2.**
$(c\|f\|t_c)^*$	car	IDF	57s	356	130	124	261	17
		NY	11s	68	96	3.8	2.6	***2.**
$(f\|p_c\|p_m\|p_t\|p_r\|p_b\|t_p)^*$	public trans	IDF	34s	182	186	***116**	291	26
		NY	9s	63	76	***37**	89	6
$(f\|p_c\|p_m\|p_t\|t_p)^*$	metro/tram	IDF	24s	135	175	23	44	2
		NY	9s	48	64	***14**	30	2
$(f\|p_c\|p_r\|t_p)^*$	trains	IDF	29s	166	172	***73**	177	16
		NY	8s	42	57	***17**	35	2
$(f\|p_b\|p_c\|t_p)^*$	bus	IDF	28s	174	216	***157**	431	41
		NY	9s	61	79	***35**	90	8
$(b\|f\|t_v)^*$	rental bike	IDF	30s	223	300	10	5	*
$(c\|f\|t_a)^*$	rental car	IDF	51s	509	623	90	96	1
$(c_f\|c_p\|c_u\|f\|t_c)^*$	private car, no toll roads	IDF	57s	347	126	108	219	13
$(c_p\|c_u\|f\|t_c)^*$	private car, no toll/fast roads	IDF	55s	340	209	***134**	349	25

[a] `bas_bi`$_{v0}$ with approximation factors 10% and 20%, [b] network, [c] preprocessing time for `bas_ls` and `bas_bi`$_{v0}$ (in seconds). Preprocessing time for `std`: 50s

Bounds are especially bad for public transportation at night time, as connections are not served as frequently as during the day.

Complex constraints. Let us now look at the scenarios with more complex constraints. In Figures 10, 12, 14, 16, 18, and 20, we report average runtimes of the different versions of `SDALT` by using methods (bas), (adv), and (spe) applied to 15 scenarios on the IDF network. Of those 15 scenarios, we run 5 on the NY network (Figures 21 and 21). See Figure 10 for information on how to read these graphs. Note that the conclusions which follow apply to both networks, IDF and NY, which proves the applicability of our algorithm to different multi-modal transportation networks.

Let us examine the uni-directional versions of `SDALT` first. Runtimes of `std` are always the worst, and sometimes even lower than plain D_{RegLC}. This can be explained intuitively by the observation that it is likely to guide the search toward arcs with the lowest cost on the shortest *un-constrained* path to the target. The uni-directional versions of SDALT, on the other hand, are able to anticipate the constraints of L_0 during the pre-processing phase and thus will tend to explore nodes toward low cost arcs which are likely to not violate the constraints of L_0. Version bas_ls works well in situations where L_0 excludes a priori fast modes of transportation. See Table VII and scenarios Ia and IIa, here the fastest mode of transportation, private car, is excluded. Version adv_ls gives a supplementary speed-up in cases where initially allowed fast modes of transportation are excluded from a later state on $\mathcal{A}_0$ onward. This can be observed in scenarios IV where the use of public transportation is excluded in state s_4, and also in scenarios V, where, when moving from s_0, either public transportation or the use of a rental car is excluded. Version spe_ls has a positive impact on runtimes for scenarios where the constrained shortest path is very different from the un-constrained shortest path. We simulate this by imposing the visit of some infrequent labels, which would generally not be part of the un-constrained shortest path. In scenarios II, III, and VI an arc with labels z_{f_1}, z_{f_2}, or z_{c_1} has to be visited which is likely to impose a detour from the un-constrained shortest path. Other cases where spe_ls is likely to improve runtimes are scenarios in which the use of fast modes of transportation is somehow limited (e.g., in scenario IVa public transportation can be used only once and no changes are allowed, in scenarios V exactly one change is allowed). Finally, versions adv_lc and spe_lc prove to be quite efficient. Especially adv_lc runs faster than adv_ls in most scenarios as it substantially reduces the number of calculated potentials, the negative effect on the runtime caused by the re-insertion of nodes turns out to be outbalanced by the lower number of visited nodes.

Let us now look at the results of the bi-directional versions. We conclude that time-dependent arcs, in general, have a negative impact on runtimes of the bi-directional versions of SDALT (scenarios Ib, II, V, and IV). In some cases, bi-directional searches which employ approximation run very fast when the number of time-dependent arcs is limited (as is the case in Ia, rental cars are available only in a small part of the graph, namely Paris and its surroundings, and in IVc where no buses and trains may be used). Bi-directional search performs very well in cases where spe_ls also works well. These are cases where the constrained shortest path is very different from the unconstrained shortest path, e.g., scenarios III and VI. As forward and backward search *communicate* with each other by using the concept of the tight potential function, the bi-directional search is able to predict these difficult constraints. Finally, version $\texttt{bi}_{v2}$ seems to dominate the other two bi-directional versions in most cases. By looking at the number of settled nodes for each version, we found that versions $\texttt{bi}_{v1}$ and $\texttt{bi}_{v2}$ settled constantly fewer nodes than $\texttt{bi}_{v0}$, but runtimes are not always lower as the algorithmic overhead is higher.

9. CONCLUSIONS

We presented different versions of uni- and bi-directional SDALT which solves the Regular Language Constraint Shortest Path Problem. Constrained shortest paths minimize costs (e.g., travel time) and in addition must respect constraints like preferences or exclusions of modes of transportation. In our scenario, a realistic multi-modal transportation network, SDALT finds constrained shortest paths 1.5 to 40 (60 with approximation) times faster than the standard algorithm, a generalized Dijkstra's algorithm ($\mathsf{D}_{\mathsf{RegLC}}$).

Recent works on finding constrained shortest paths on multi-modal networks report speed-ups of different orders of magnitude. They achieve this by using contraction hierarchies. The authors of [Rice and Tsotras 2010] apply contraction to a graph consisting of different road types and limit the regular languages which can be used to constrain the shortest paths to Kleene languages (road types may only be excluded, for example toll roads). We use Kleene languages for the scenarios reported in Table VII. Here, SDALT provides maximum speed-ups of about factor 20. However, besides limiting the range of applicable regular languages, [Rice and Tsotras 2010] do not consider public transportation nor traffic information (time-dependent arc cost functions) which are important components of multi-modal route planning. The authors of [Dibbelt et al. 2012] apply contraction only to the road network of a multi-modal transportation network consisting of foot, car, and public transportation. Their scenario is comparable to scenarios IV. Here, SDALT provides maximum speed-ups of about factor 3 to 10. However, the authors do not consider traffic information nor different road classes. SDALT considers and incorporates both.

SDALT is a general method to speed-up D_{RegLC} for all regular languages and for all types of labeled graphs and which can be applied to networks including time-dependent arc costs. We discussed under which conditions SDALT should provide good speed-ups. Another advantage of SDALT, although not explicitly discussed in this work, is that the original graph is not modified by the preprocessing process, as it is based on ALT. Because of that, real time information can be incorporated easily (changing traffic information, closures of roads, etc.), without recalculating preprocessed data (under mild conditions).

The objective of future research on constrained shortest path calculation is to further increase speed-ups. The combination of SDALT and contraction is a viable option, although handling time-dependency and considering the labels on arcs during the contraction process is not straightforward. A further area of future research is to study the multi-criteria scenario, where not only travel time but also, e.g., travel cost or the number of changes are minimized.

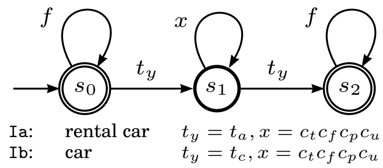


Fig. 9: Scenarios `I`: a path starts and ends by walking. A car (scenario `Ia`) or rental car (scenario `Ib`) may be used once.

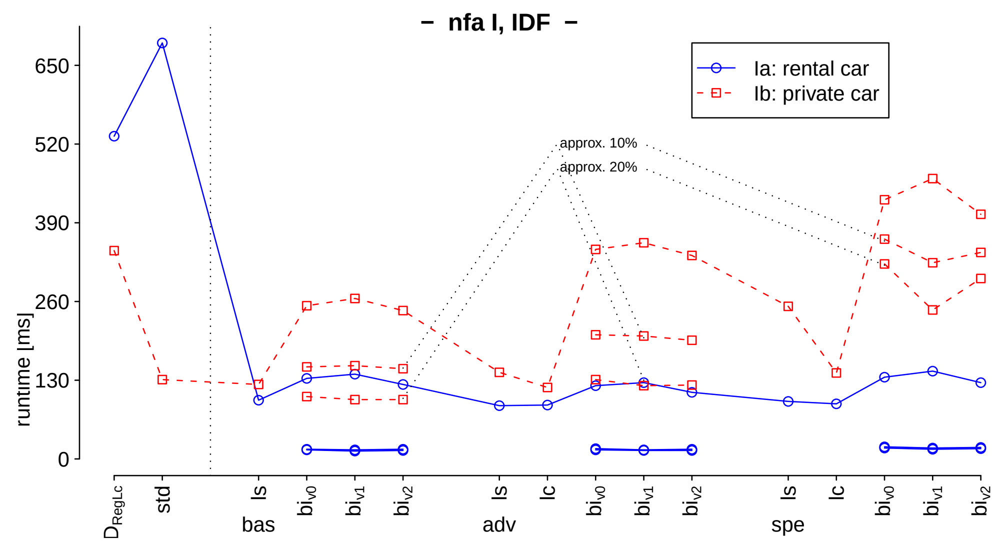


Fig. 10: Experimental results for scenarios `I`. The different line-types indicate average runtimes (in milliseconds [ms]) of the different `SDALT` variants when varying the allowed modes of transportation. In this example, the continuous blue and dashed red lines indicate average runtimes for the different `SDALT` variants for scenarios `Ia` and `Ib`. We provide average runtimes for D_{RegLC}, `std`, `bas_ls`, `bas_bi`$_{vx}$, `adv_ls`, `adv_lc`, `adv_bi`$_{vx}$, `spe_ls`, `spe_lc`, and `spe_bi`$_{vx}$ (abbreviated in this order on the graph). For all bi-directional versions of the algorithms we also report average runtimes for an approximation factor of 10% and of 20% (in the graph indicated for scenario `Ib`). For scenario `Ia` average runtimes for D_{RegLC} are about 530ms. Applying `std` results in a speed-down (680ms). Instead, `bas_ls` works very well (100ms) and applying bi-directional search with approximation even more so (10ms). Note that results for an approximation of 10% and 20% for this scenario coincide. For scenario `Ib`, average runtimes for D_{RegLC} are about 360ms. `std` and `bas_ls` provide a speed-up of about factor 3. The other algorithms do not provide better results.

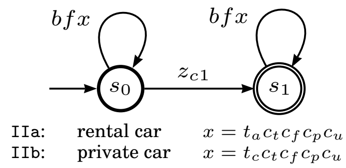


Fig. 11: Scenarios II: Walking, rental car (scenario IIa), or private car (scenario IIb) may be used to reach the target. One arc with label z_{c1} has to be visited.

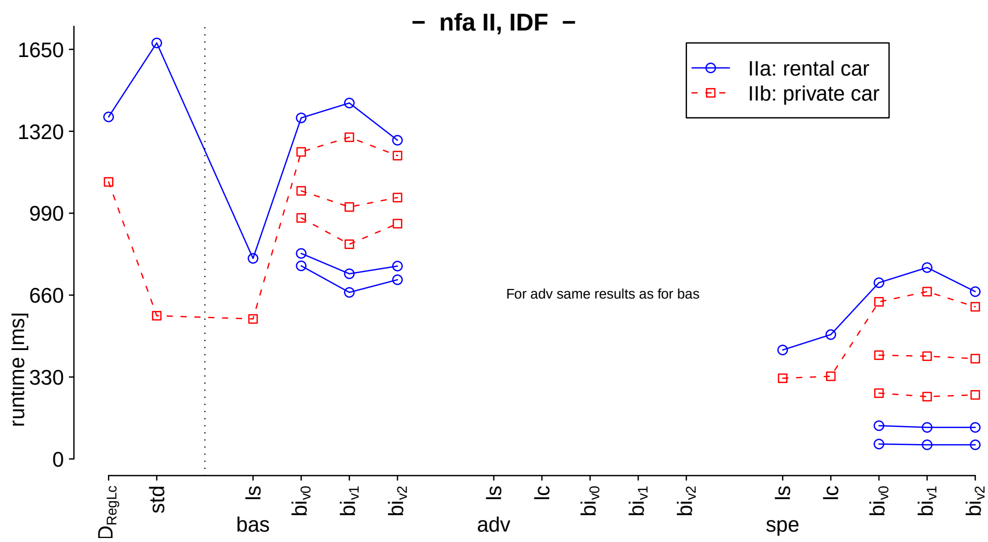
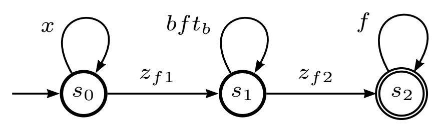


Fig. 12: Experimental results for scenarios II. For scenario IIa std is slower than D_{RegLC}. bas_ls and bas_bi$_{vx}$ provide a speed-up of about factor 2. spe_ls runs slightly faster. The bi-directional algorithms spe_bi$_{vx}$ work very well and provide average runtime of about 60ms (speed-up factor of about 20). For scenario IIa, std and bas_ls perform equally, the different versions of spe provide slightly better results.



`IIIa:`	private car	$x = t_a f c_t c_f c_p c_u$
`IIIb:`	private car, no fast roads	$x = t_a f c_p c_u$
`IIIc:`	public trans	$x = t_p f p_b p_m p_r p_t$
`IIId:`	metro/tram	$x = t_p f p_m p_t$

Fig. 13: Scenarios `III`: the path begins with private car (scenarios `IIIa` and `IIIb`) or public transportation (scenarios `IIIc` and `IIId`). After visiting an arc with label z_{f1}, the path may be continued by rental bicycle and/or by walking. Before reaching the target by walking, an arc with label z_{f2} has to be visited.

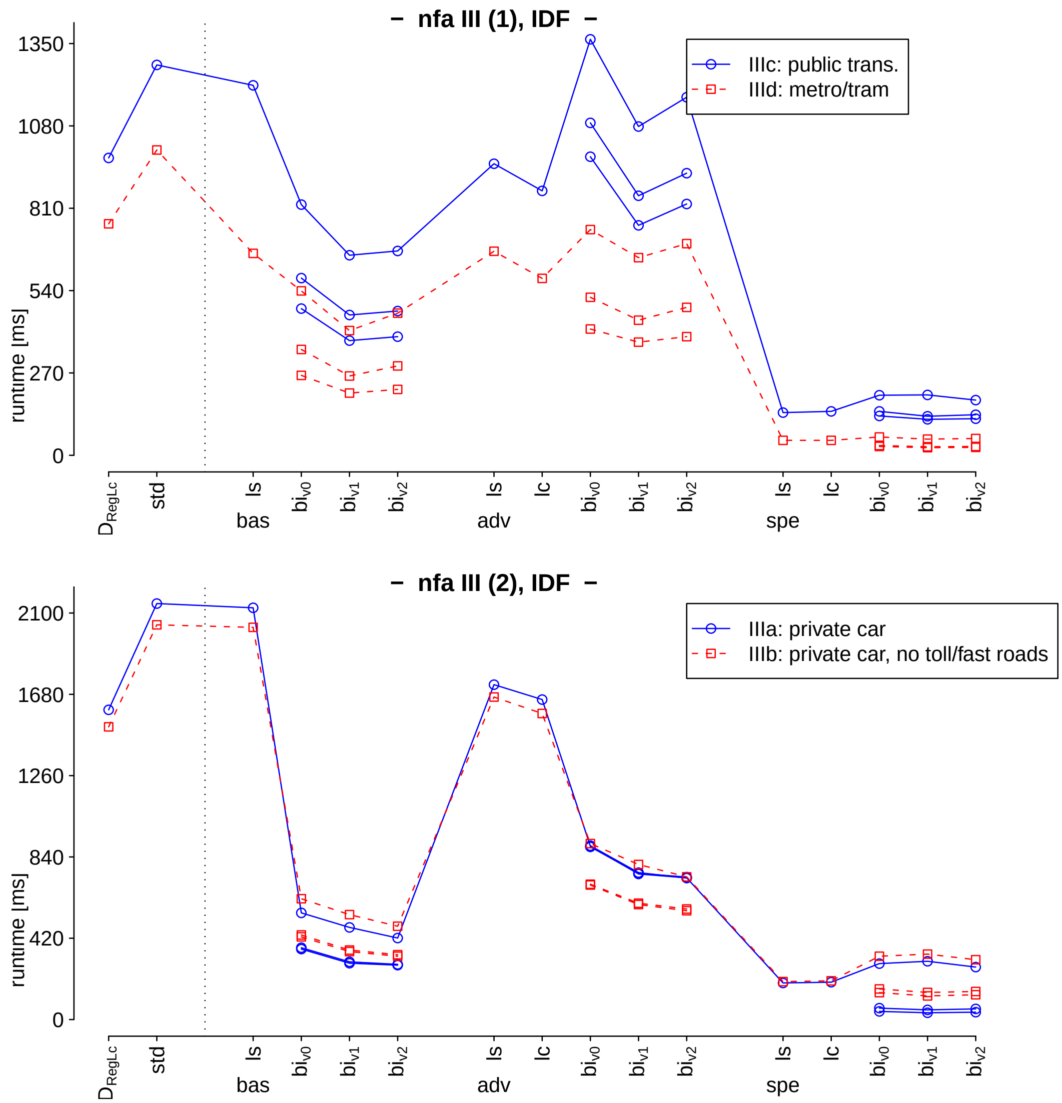


Fig. 14: Experimental results for scenarios III. For all scenarios the algorithms std, bas_ls, adv_ls, and adv_ls are not very efficient. Instead, spe_ls and spe_lc and the bi-directional versions work very well. They provide a speed-up of a factor of 10 to 15.

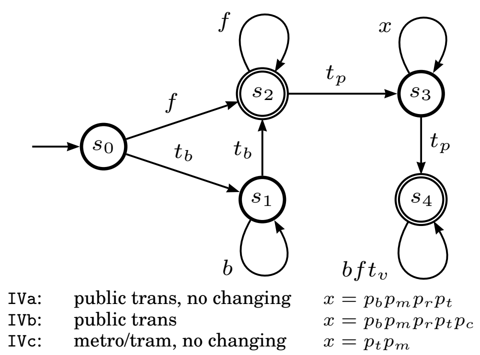


Fig. 15: Scenarios IV: the path begins either by walking or private bicycle. Once the private bicycle is discarded, the path may be continued by walking. Public transportation may be used (all public transportation without changing (scenario IVa), with changing (scenario IVb), or only metro/tram without changing (scenario IVc)). Finally, the target may be reached by walking or by using a rental bicycle.

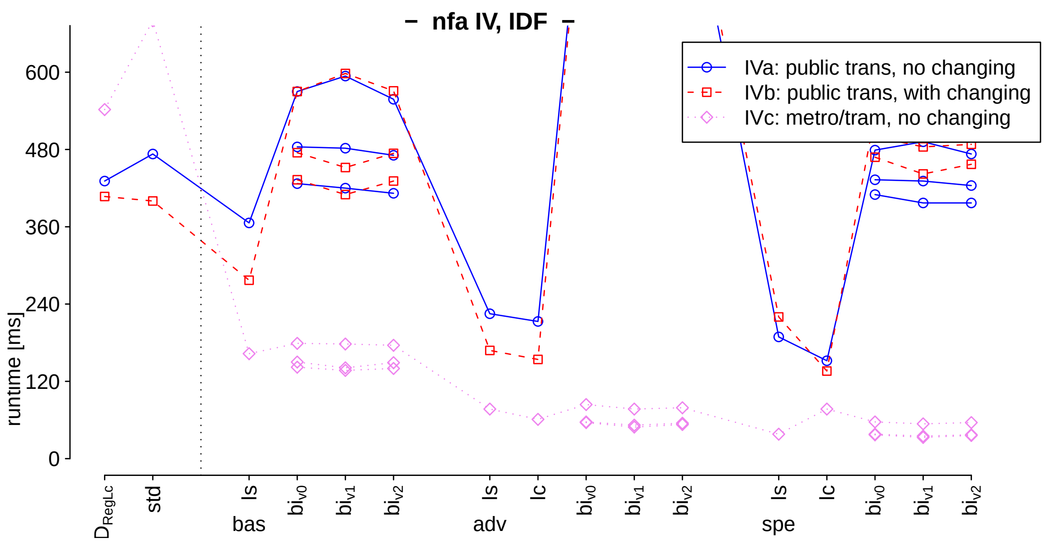


Fig. 16: Experimental results for scenarios IV. The bi-directional versions of the algorithm and std are not efficient. Instead, bas_ls, adv_ls, and spe_ls provide speed-ups of a factor between 2 and 10.

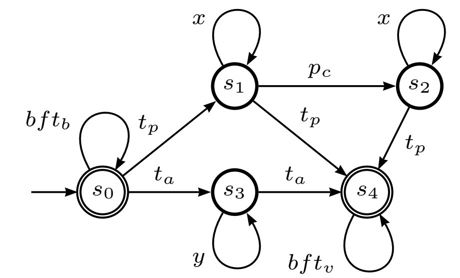


Fig. 17: Scenarios V: a path begins by walking or by using a private bicycle. Then either a rental car or public transportation may be used (one or two changes). At the end a rental bicycle or walking may be used to reach the target. In scenario Va all public transportation may be used, in scenario Vb only metro and tram.

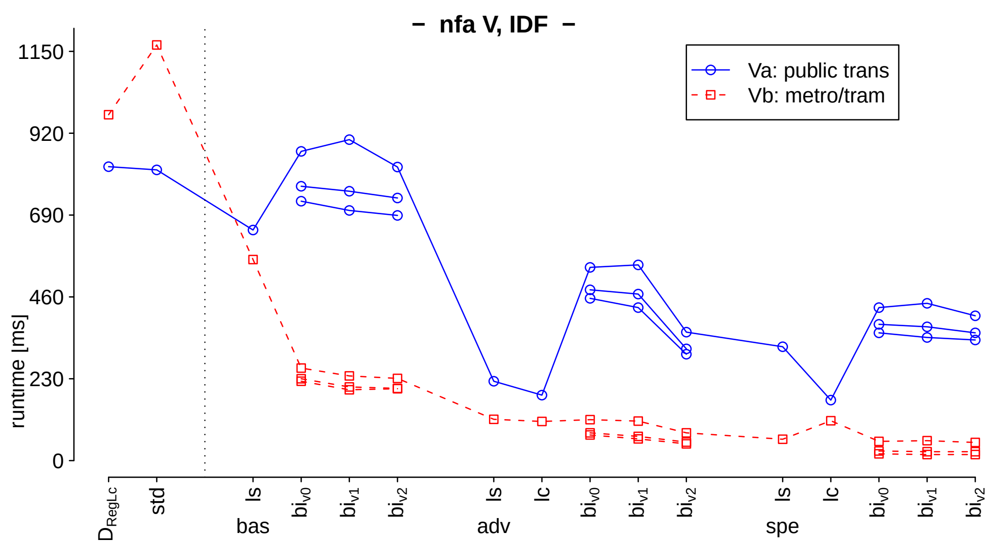


Fig. 18: Experimental results for scenarios V. Bi-directional search does not work well if public transportation can be used (scenario Va). Instead, if public transportation is restricted (scenario Vb) bi-directional search is very fast. For scenario Vb, bi-directional search with approximation of 20% provides a speed-up of about a factor of 60, spe_ls of a factor of 15.

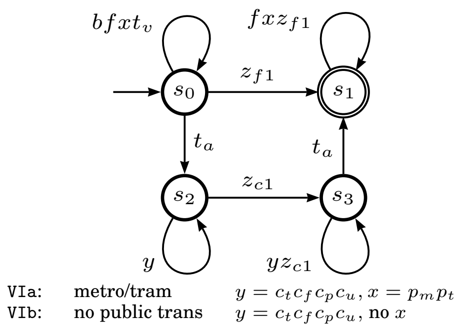


Fig. 19: Scenarios VI: Walking, rental bicycle, and rental car may be used, but either an arc with label z_{f1} or z_{c1} has to be visited (scenario VIb). In scenario VIb also metro and tram may be used.

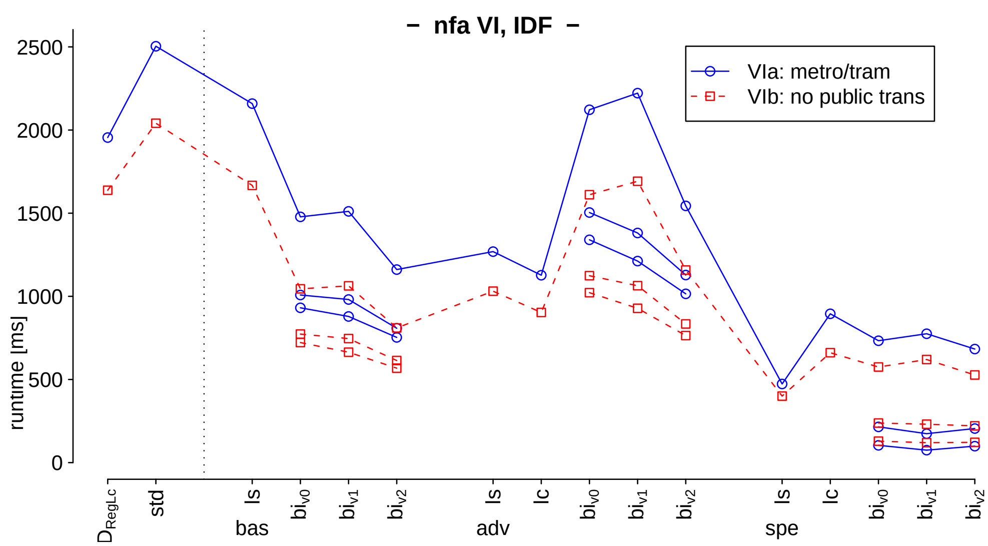


Fig. 20: Experimental results for scenarios VI.

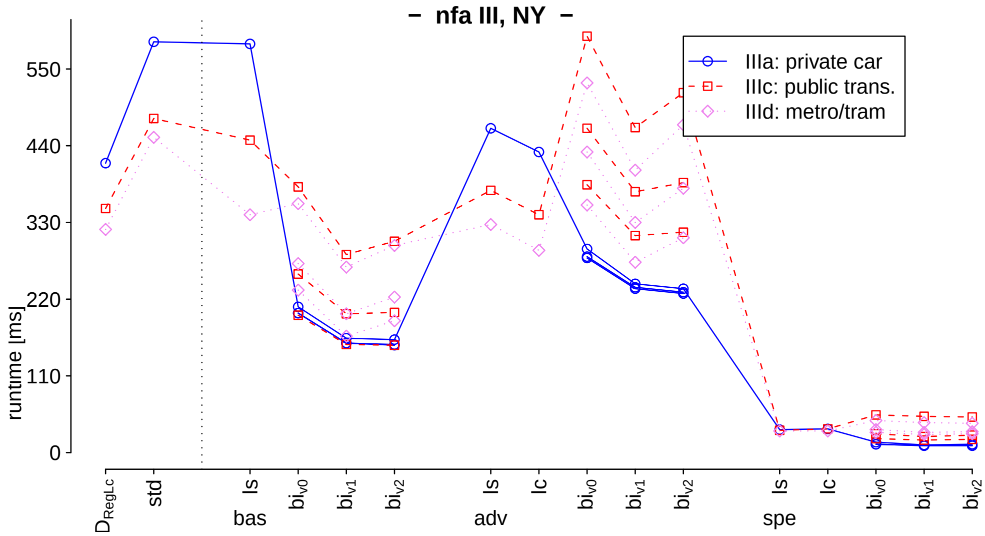


Fig. 21: Experimental results for scenarios `III` on network NY.

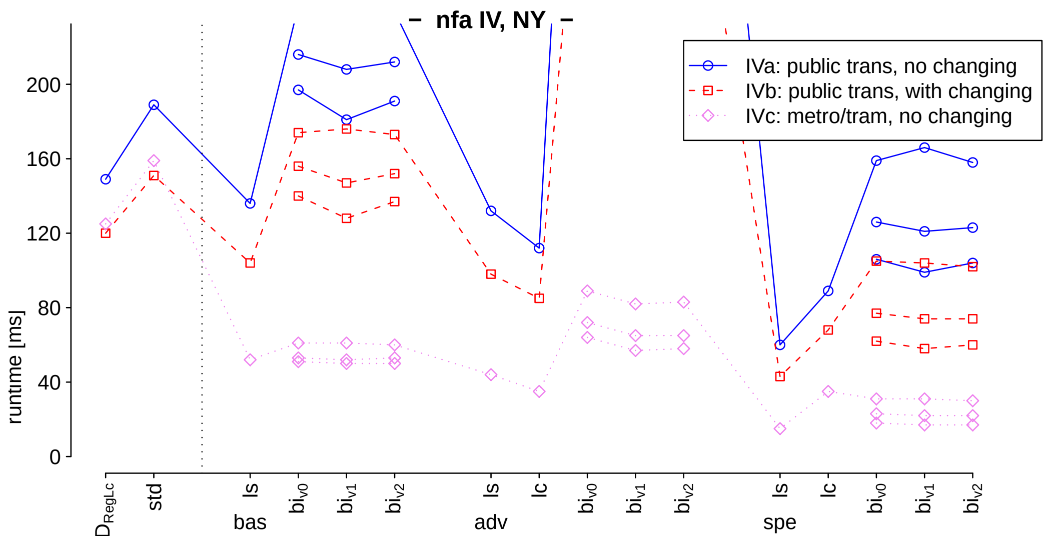


Fig. 22: Experimental results for scenarios `IV` on network NY.

Table VIII: Preprocessing times (in minutes and seconds). (For `std`: 50s.)

Scenarios	bas_ls bi-bas$_{vx}$	adv_ls adv_lc	bi-adv$_{vx}$	spe_lc	spe_ls bi-spe$_{vx}$
Ile-de-France, IDF					
Ia	51s	1m11s	1m11s	2m54s	2m54s
Ib	58s	1m16s	1m11s	3m6s	3m6s
IIa	52s	-	-	3m23s	2m32s
IIb	57s	-	-	3m56s	2m58s
IIIa	1m19s	2m17s	4m37s	5m02s	4m39s
IIIb	1m11s	2m2s	4m8s	4m58s	4m20s
IIIc	50s	1m48s	3m10s	4m01s	3m33s
IIId	37s	1m31s	2m32s	3m35	2m59s
IVa	48s	2m10s	3m31s	2m49s	5m41s
IVb	48s	2m0s	3m18s	2m43s	5m32s
IVc	37s	1m42s	2m52s	2m30s	5m6s
Va	1m28s	4m41s	8m08s	6m01	6m12s
Vb	1m14s	4m0s	6m54s	5m29	5m39s
VIa	1m15s	2m35s	5m41s	5m26s	5m27s
VIb	1m8s	2m19s	5m07s	4m52s	4m52s
New York, NY					
IIIb	17s	34s	1m01s	1m28s	1m10s
IIIc	16s	33s	58s	1m23s	1m8s
IIId	14s	31s	53s	1m20s	1m6s
IVb	15s	32s	59s	45s	1m38s
IVc	13s	29s	54s	44s	1m34s

Table IX: Size of preprocessed data (in MB).

Scenarios	std_ls bas_ls, bi-bas$_{vx}$	adv_ls adv_lc	bi-adv$_{vx}$	spe_lc	spe_ls bi-spe$_{vx}$
Ile-de-France, IDF					
Ia, Ib	306	612	612	1224	1224
IIa, IIb	306	-	-	918	612
IIIa, IIIb, IIIc, IIId	306	918	1530	1530	1224
IVa, IVb, IVc	306	918	1530	1224	1836
Va, Vb	306	1530	2754	1836	1836
VIa, VIb	306	918	1836	1224	1224
New York, NY					
IIIa, IIIb, IIIc, IIId	86	258	430	430	344
IVa, IVb, IVc	86	258	430	344	516

REFERENCES

Barrett, C. L., Bisset, K. R., Holzer, M., Konjevod, G., Marathe, M., and Wagner, D. 2008. Engineering label-constrained shortest-path algorithms. In *Algorithmic Aspects in Information and Management*, R. Fleischer and Jinhui Xu, Eds. LNCS Series, vol. 5034. Springer, Berlin, 27–37.

Barrett, C. L., Bisset, K. R., Jacob, R., Konjevod, G., and Marath, M. V. 2002. Classical and contemporary shortest path problems in road networks: Implementation and experimental analysis of the TRANSIMS router. In *European Symposium on Algorithms (ESA)*, R. H. Mohring and R. Raman, Eds. LNCS Series, vol. 2461. Springer, Berlin, 126–138.

Barrett, C. L., Jacob, R., and Marathe, M. 2000. Formal-Language-Constrained Path Problems. *SIAM Journal on Computing 30,* 3, 809–837.

Delling, D., Goldberg, A. V., Pajor, T., and Werneck, R. F. F. 2011. Customizable Route Planning. In *Symposium on Experimental Algorithms (SEA)*, P. M. Pardalos and S. Rebennack, Eds. LNCS Series, vol. 6630. Springer, Berlin, 376–387.

Delling, D. and Nannicini, G. 2008. Bidirectional core-based routing in dynamic time-dependent road networks. In *International Symposium on Algorithms and Computation (ISAAC)*, S.-H. Hong, H. Nagamochi, and T. Fukunaga, Eds. LNCS Series, vol. 5369. Springer, Berlin, 812–823.

Delling, D., Pajor, T., and Wagner, D. 2009a. Accelerating Multi-Modal Route Planning by Access-Nodes. In *European Symposium on Algorithms (ESA)*, A. Fiat and P. Sanders, Eds. LNCS Series, vol. 5757. Springer, Berlin, 587–598.

Delling, D., Sanders, P., Schultes, D., and Wagner, D. 2009b. Engineering route planning algorithms. In *Algorithmics of Large and Complex Networks*, J. Lerner, D. Wagner, and K. A. Zweig, Eds. LNCS Series, vol. 5515. Springer, Berlin, 117–139.

Delling, D. and Wagner, D. 2009. Time-Dependent Route Planning. In *Robust and Online Large-Scale Optimization*, R. K. Ahuja, R. H. Mohring, and C. D. Zaroliagis, Eds. LNCS Series, vol. 5868. Springer, Berlin, 207–230.

Dibbelt, J., Pajor, T., and Wagner, D. 2012. User-Constrained Multi-Modal Route Planning. In *Algorithm Engineering and Experiments (ALENEX)*, D. A. Bader and P. Mutzel, Eds. SIAM, 118–129.

Dijkstra, E. W. 1959. A note on two problems in connexion with graphs. *Numerische Mathematik 1,* 1, 269–271.

Goldberg, A. V. and Harrelson, C. 2005. Computing the shortest path: A* search meets graph theory. In *Proceedings of the Symposium on Discrete Algorithms (SODA)*. SIAM, Philadelphia, 156–165.

Goldberg, A. V. and Werneck, R. F. F. 2005. Computing point-to-point shortest paths from external memory. In *ALENEX/ANALCO*, C. Demetrescu, R. Sedgewick, and R. Tamassia, Eds. SIAM, 26–40.

HART, P. E., NILSSON, N. J., AND RAPHAEL, B. 1968. A Formal Basis for the Heuristic Determination of Minimum Cost Paths. *IEEE Transactions on Systems Science and Cybernetics 4,* 2, 100–107.

IKEDA, T., HSU, M.-Y., IMAI, H., NISHIMURA, S., SHIMOURA, H., HASHIMOTO, T., TENMOKU, K., AND MITOH, K. 1994. A fast algorithm for finding better routes by AI search techniques. In *Proceedings of Vehicle Navigation and Information Systems Conference*. IEEE, 291–296.

KAUFMAN, E. AND SMITH, R. L. 1993. Fastest paths in time-dependent networks for intelligent vehicle-highway systems applications. *IVHS Journal 1,* 1, 1–11.

KIRCHLER, D., LIBERTI, L., PAJOR, T., AND WOLFLER CALVO, R. 2011. UniALT for Regular Language Constrained Shortest Paths on a Multi-Modal Transportation Network. In *Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS)*, A. Caprara and S. C. Kontogiannis, Eds. OASICS Series, vol. 20. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Germany, 64–75.

KIRCHLER, D., LIBERTI, L., AND WOLFLER CALVO, R. 2012. A label correcting algorithm for the shortest path problem on a multi-modal route network. In *Symposium on Experimental Algorithms (SEA)*, R. Klasing, Ed. LNCS Series, vol. 7276. Springer, Berlin, 236–247.

LOZANO, A. AND STORCHI, G. 2001. Shortest viable path algorithm in multimodal networks. *Transportation Research Part A 35*, 225–241.

MENDELZON, A. O. AND WOOD, P. T. 1995. Finding Regular Simple Paths in Graph Databases. *SIAM Journal on Computing 24,* 6, 1235–1258.

NANNICINI, G., DELLING, D., LIBERTI, L., AND SCHULTES, D. 2008. Bidirectional A* search for time-dependent fast paths. In *Conference on Experimental Algorithms (WEA)*, C. C. McGeoch, Ed. LNCS Series, vol. 5038. Springer, Berlin, 334–346.

ORDA, A. AND ROM, R. 1990. Shortest-path and minimum-delay algorithms in networks with time-dependent edge-length. *Journal of the ACM 37,* 3, 607–625.

POHL, I. 1971. Bi-directional Search. In *Machine Intelligence 6*, B. Meltzer and D. Michie, Eds. Vol. 6. Edinburgh University Press, Edinburgh, 127–140.

PYRGA, E., SCHULZ, F., WAGNER, D., AND ZAROLIAGIS, C. D. 2007. Efficient models for timetable information in public transportation systems. *ACM Journal of Experimental Algorithmics 12*.

RICE, M. AND TSOTRAS, V. J. 2010. Graph indexing of road networks for shortest path queries with label restrictions. *Proceedings of the VLDB endowment 4,* 2, 69–80.

ROMEUF, J. 1988. Shortest path under rational constraint. *Information processing letters 28,* 5, 245–248.

SHERALI, H. D., HOBEIKA, A. G., AND KANGWALKLAI, S. 2003. Time-Dependent, Label-Constrained Shortest Path Problems with Applications. *Transportation Science 37,* 3, 278–293.

Sherali, H. D., Jeenanunta, C., and Hobeika, A. G. 2006. The approach-dependent, time-dependent, label-constrained shortest path problem. *Networks 48,* 2, 57–67.

Yannakakis, M. 1990. Graph-theoretic methods in database theory. In *Proceedings of Symposium on Principles of Database Systems*. ACM, New York, USA, 230–242.