A label correcting algorithm for the shortest path problem on a multi-modal route network

Dominik Kirchler[1,2,3], Leo Liberti[1], Roberto Wolfler Calvo[2]

[1] LIX; Ecole Polytechnique
{kirchler,liberti}@lix.polytechnique.fr
[2] LIPN; Université Paris 13
roberto.wolfler@lipn.univ-paris13.fr
[3] Mediamobile

Abstract. We consider shortest paths on a multi-modal transportation network where restrictions or preferences on the use of certain modes of transportation may arise. The regular language constraint shortest path problem deals with this kind of problem. It models constraints by using regular languages. The problem can be solved efficiently by using a generalization of Dijkstra's algorithm (D_{RegLC}). Recently a speed-up technique called SDALT has been proposed to speed up D_{RegLC}. In this paper we present a new label correcting algorithm (lcSDALT) which overcomes some limitations of SDALT. We evaluated lcSDALT on a realistic multi-modal transportation network with various modes of transport and road types, including time-dependent cost functions on arcs. We observed a speed-up of a factor 3 to 30, with respect to D_{RegLC} and of up to 50% with respect to SDALT, while at the same time reducing memory requirements.

1 Introduction

Multi-modal transportation networks include, other than 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. Furthermore, they may wish to pass by some grocery shop on the way or to limit the number of changes between public transportation vehicles. Feasibility has also to be assured: whereas walking is permitted at any point of the itinerary, 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 [3], 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 (D_{RegLC}) to solve RegLCSP.

In recent years many scholars worked on speed-up techniques for Dijkstra's algorithm and shortest paths on continental sized road networks can now be found in a few milliseconds [5]. D_{RegLC} has received less attention. A recent work [14] proposes

the algorithm SDALT (State Dependent uniALT) which succeeds in accelerating D_{RegLC} by anticipating information of the additional constraints, modeled by the regular language, during a pre-processing phase. It uses the preprocessed data to calculate lower bounds of the distance of the shortest path to guide D_{RegLC} faster toward the target.

Our Contribution SDALT is based on the uniALT algorithm [9], which works correctly only if the node potential function is *feasible*. The potential function is used to determine lower bounds to guide the algorithm faster to the target. We propose the algorithm lcSDALT which overcomes this limitation. It includes all the characteristics of SDALT but introduces more flexibility in the choice of the potential function. It therefore runs faster than SDALT by up to 50% in our setting. Furthermore, it reduces memory requirements. We provide experimental results on a realistic multi-modal public transportation network including time-dependent cost functions on arcs. It is composed of the road and public transportation network of the French region Ile-de-France : private and rental bicycles, walking, car (including changing traffic conditions over the day), and public transportation. The experiments show that our algorithm performs well in networks where some transportation modes tend to be faster than others or the constraints cause a major detour on the non-constrained shortest path. We observed speed-ups of a factor 3 to 30, in respect to plain 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 [20, 15]. 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 [3]. 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}).

In recent years, much effort has been dedicated to accelerating algorithms to solve the uni-modal shortest path problem on large road networks, see [5] for a comprehensive overview. It identifies three basic ingredients to most modern speed-up techniques: bi-directional search, goal-directed search, and contraction.

ALT is a bi-directional, goal directed search technique based on the A^* search algorithm [11] and has been first discussed in [9]. 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 edges cost are given in [16].

In [1], various speed-up techniques and its combinations including bi-directional and goal-directed search have been applied to D_{RegLC} on rail and road networks. The performance of the algorithm strongly depends on the network properties and on how restrictive the regular language is. The authors of [4] propose Access Node Routing to isolate the public transportation network from road networks so that they can be treated individually. A similar approach has been followed in [7] where

contraction has been applied only to arcs belonging to the road network of a multi-modal transportation network including roads, public transport and flight data. No update of preprocessing data is needed for different regular languages

The authors of [19] 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 effective in such a scenario. The authors report on speed-ups of over 3 orders of magnitude compared to D_{RegLC}.

Overview This paper is organized as follows. Section 3 will first define the graph we are using to model the transportation network and give more details about RegLCSP and SDALT. Section 4 presents our new algorithm lcSDALT and its implementation. Its application to a realistic multi-modal transportation network and computational results, as well as some conclusive remarks are presented in section 4.3.

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 labeled arcs $(i, j, l) \in A$ which are triplets in $V \times V \times \Sigma, i, j \in V, l \in \Sigma$. (i, j, l) represents an arc from node i to node j having label l. The labels are used to mark arcs as, e.g., foot paths (label f), bicycle lanes (label b), etc. Arc costs represent travel times. They are positive and time-dependent: $c : A \to (\mathbb{R}_+ \to \mathbb{R}_+)$, i.e. $c_{ijl}(\tau)$ gives the traveling times from node i to node j using transportation mode l 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 [13], whereas it is NP-hard in non-FIFO networks [17]. FIFO means that $c_{ijl}(x) + x \leq c_{ijl}(y) + y$ for all $x, y \in \mathbb{R}_+, x \leq y, (i, j, l) \in A$ or, in other words, that for any arc (i, j, l), leaving node i earlier guarantees that one will not arrive later at node j (also called the non-overtaking property).

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 regular language L_0 can be described by a non-deterministic finite state automaton $\mathcal{A}_\prime = (S_0, \Sigma_0, \delta_0, s_0, F_0)$, consisting of a set of states S_0, a set of labels $\Sigma_0 \subseteq \Sigma$, a transition function $\delta_0 : \Sigma_0 \times S_0 \to 2^{S_0}$, an initial state s_0, and a set of final states F_0 (see for an example Figure 4a). Note that $\overleftrightarrow{S}(s, \mathcal{A})$ and $\overleftrightarrow{\Sigma}(s, \mathcal{A})$ return all states and labels reachable on an atomaton $\mathcal{A}$ by starting at state s, backward and forward, respectively. E.g., in Figure 4a, $\overleftarrow{S}(s_1, \mathcal{A}_0) = \{s_0, s_1, s_3\}$ $\overrightarrow{\Sigma}(s_4, \mathcal{A}_0) = \{c_{\mathsf{fast}}, c_{\mathsf{pav}}, f, t, v\}$.

To efficiently solve RegLCSP, a generalization of Dijkstra's algorithm (which we denote D_{RegLC} throughout this paper) has first been proposed in [3]. The D_{RegLC} algorithm can be seen as the application of Dijkstra's algorithm [8] to the product graph $P = G \times S_0$ with tuples (v, s) as nodes for each $v \in V$ and $s \in S_0$ such that there is an arc $((v, s)(w, s'))$ between (v, s) and (w, s') if there is an arc $(i, j, l) \in A$ and a transition such that $s' \in \delta_0(l, s)$. To reduce storage space D_{RegLC} works on the *implicit* product graph P by generating all the neighbors which have to be explored only when necessary.

3.2 SDALT

The SDALT algorithm [14] is a speed-up technique for D_{RegLC} based on A^* [11] and *landmarks* [9]. It is also based on D_{RegLC} and works on the implicit product graph P. Different from D_{RegLC}, SDALT uses an estimated lower bound of the distance to the target to guide the search more directly toward the target. More precisely, it employs a key $k(v, s) = d_r(v, s) + \pi(v, s)$ where $d_r(v, s)$ is the tentative distance label from (r, s_0) to (v, s) and the *potential function* $\pi : (V \times S) \to \mathbb{R}$ gives an under-estimation of the distance from (v, s) to (t, s), $s \in F_0$. At every iteration, the algorithm selects the node (v, s) with the smallest key $k(v, s)$. Intuitively, nodes which are close to the shortest estimated path between the source and the target node are explored first. So the closer $\pi(v, s)$ is to the actual remaining distance, the faster the algorithm will find the target.

It can be shown (see [12]) that SDALT is equivalent to D_{RegLC} on a product graph with *reduced arc costs* $c^{\pi}_{(v,s_v)(w,s_w)l} = c_{vwl} - \pi(v, s_v) + \pi(w, s_w)$. D_{RegLC} is based on Dijkstra's algorithm which works only for non-negative arc costs, so not all potential functions can be used. A potential function π is *feasible*, if $c^{\pi}_{(v,s_v)(w,s_w)l}$ is non-negative for all $(v, w, l) \in A$. Note that, if π' and π'' are feasible potential functions, then $\max(\pi', \pi'')$ is a feasible potential function [9].

Tight bounds can be produced by using landmarks and the triangle inequality [9]. The main idea is to select a small set of nodes $\ell \in \mathcal{L} \subset V$ (also called *landmarks*), appropriately spread over the network, and pre-compute all distances of shortest paths between the landmarks and any other node. By using these *landmark distances* and the triangle inequality, lower bounds on the distances between any two nodes can be derived. Finding good landmarks is difficult and several heuristics exist [9, 10]. SDALT applies these concepts to speed-up D_{RegLC} on P by using the following potential function (see also Figure 1):

$$\pi(v, s) = \max \bigcup_{\ell \in \mathcal{L}} \{d'(\ell, t, s) - d'(\ell, v, s), d'(v, \ell, s) - d'(t, \ell, s), 0\} \tag{1}$$

$d'(i, j, s)$ denotes the *constrained landmark distance*, which is the travel time on the shortest path from (i, s) to (j, s') for some $s' \in F_0$ *constrained* by the regular language $L_s^{i \to j}$. Here lies the major conceptual difference between SDALT and uniALT. Differently from uniALT, SDALT does not use Dijkstra's algorithm to determine landmark distances, but it uses the D_{RegLC} algorithm instead. This way, it is possible to constrain the landmark distance calculation by some regular languages which is derived from L_0. Thus SDALT is able to already consider the constraints given by L_0 during the preprocessing phase. Note that $d'(v, t, s)$ is constrained by

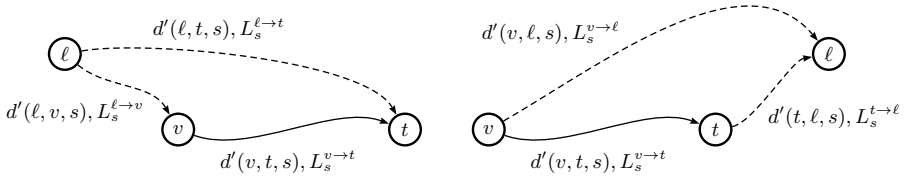


Fig. 1: Landmark distances for SDALT

$L_s^{v\to t} = L_0^s$. L_0^s is equal to L_0 except that the initial state of $\mathcal{A}_0$ is replaced by s. Intuitively L_0^s represents the constraints of L_0 to be considered for the remaining portion of the shortest path from an arbitrary node (v, s) to the target.

In [14] three methods, (bas), (adv), and (spe) have been presented on how to choose $L_s^{\ell\to t}$, $L_s^{\ell\to v}$, $L_s^{v\to \ell}$, $L_s^{t\to \ell}$ used to constrain the calculation of $d'(\ell, t, s)$, $d'(\ell, v, s)$, $d'(v, \ell, s)$, $d'(t, \ell, s)$ in such a way that the resulting potential function (Equation 1) results feasible and provides correct lower bounds. They are based on Proposition 1. Note that 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 $L_1 \circ L_2 = L_3 = \{ac, ad, bc, bd\}$.

Proposition 1 ([14]). *For all $s \in S$, if $L_s^{\ell\to v} \circ L_s^{v\to t} \subseteq L_s^{\ell\to t}$, then $d'(\ell, t, s) - d'(\ell, v, s)$ is a lower bound for the distance $d'(v, t, s)$. Equally, if $L_s^{v\to t} \circ L_s^{t\to \ell} \subseteq L_s^{v\to \ell}$ then $d'(v, \ell, s) - d'(t, \ell, s)$ is a lower bound for $d'(v, t, s)$.*

4 Label Correcting State Dependent uniALT: lcSDALT

SDALT 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. This compensates, at least in our scenario, the additional computational effort required to remedy the disturbing effects of the use of negative reduced costs on the underlying Dijkstra's algorithm and in addition results in shorter query times and lower memory requirements.

This is why we propose a variation 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. We name the new algorithm Label Correcting State Dependent uniALT (lcSDALT). Note that here *label* refers to the distance label of the algorithm and not to the labels on arcs.

4.1 Query

The algorithm lcSDALT is similar to D_{RegLC} and uniALT. As priority queue Q we use a binary heap. The pseudo-code in figure 2 works as follows: the algorithm maintains, for every visited node (v, s) in the product graph P, a tentative distance label $d_r(v, s)$ and a parent node $p(v, s)$. It starts by computing the potential for the start node (r, s_0) and by inserting into Q (line 3). At every iteration, the algorithm

```
1 function SDALT_Lcor(G,r,t,τ_start,L_0)
2    d_r(v,s):= ∞, p(v,s):=-1, π_{v,s} := 0, ∀(v,s) ∈ V × S
3    path_found:= false, d_r(r,s_0):= 0, k(r,s_0):= π(r,s_0), p(r,s_0):= -1
4    insert (r,s_0) in priority queue Q
5    while Q is not empty:
6       extract (v,s) with smallest key k from Q
7       if v == t and s ∈ F_0:
8          path_found:= true, break
9       for each (w,s') s.t. (v,w,l) ∈ A ∧ s' ∈ δ(l,s):
10         d_tmp:= d_r(v,s) + c_vwl(τ_start + d_r(v,s))   //time-dependency
11         if d_tmp < d_r(w,s'):
12            p(w,s'):= (v,s), d_r(w,s'):= d_tmp
14            if (w,s') not in Q and never visited:   //insert
15               π_{w,s'}:= π(w,s')
16               k(w,s'):= d_r(w,s') + π_{w,s'}
17               insert (w,s') in Q
18            elif (w,s') not in Q:                    //re-insert
19               k(w,s'):= d_r(w,s') + π_{w,s'}
20               insert (w,s') in Q
21            else:                                    //decrease
22               k(w,s'):= d_r(w,s') + π_{w,s'}
23               decreaseKey w,s' in Q
24      end for
25   end while
```

Fig. 2: Pseudo-code lcSDALT

extracts the node (v, s) in Q with the smallest key (the node is *settled*) and *relaxes* all outgoing arcs (line 9), i.e. checking and possibly updating the key and distance label for every node (w, s') where $s' \in \delta(l, s)$. More precisely, a new temporary distance label $d_{\mathsf{tmp}} = d_r(v, s) + c_{vwl}(\tau_{start} + d_r(v, s))$ is compared to the currently assigned distance label (lines 10). If it is smaller, it either inserts or re-inserts node (w, s') into the priority queue or decreases its key (line 14, 18, 21). Note that it is necessary to calculate the potential of a node (w, s') only the first time it is visited. The cost of arc (v, w, l) might be time-dependent and thus has to be evaluated for time $\tau_{start} + d_r(v, s)$. The algorithm terminates when a node (t, s) with $s \in F_0$ is settled. The resulting shortest path can be produced by following the parent nodes backward starting from node (t, s).

Correctness The algorihtm lcSDALT is based on D_{RegLC} and uniALT. It suffices to prove that when the target node (t, s'), $s' \in F_0$ is extracted from the priority queue the algorithm can stop (see Lemma 1 and Proposition 2). Note that $\pi(t, s') = 0$, that $d_r^*(v, s)$ is the distance of the shortest path from (r, s_0) to (v, s), and that there are no negative cycles as arc costs and potentials are always non-negative.

Lemma 1. *The priority queue always contains a node (i, s') with key $k(i, s') = d_r^*(i, s') + \pi(i, s')$ and which belongs to the shortest path from (r, s_0) to (t, s'') where $s'' \in F_0, s' \in S$.*

Proposition 2. *If a solution exists, lcSDALT finds the shortest path.*

Proof. Let us suppose that a node (t, s), where $s \in F_0$, is extracted from the priority queue but its distance label is not optimal, so $d_r(t, s) \neq d_r^*(t, s)$. Node (t, s) has key $k(t, s_f) = d_r(t, s_f) + \pi(t, s) \neq d_r^*(t, s)$. By Lemma 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 $k(i, s') = d_r^*(i, s') + \pi(i, s') > d_r(t, s) + \pi(t, s) = k(t, s)$, which is a contradiction. □

Complexity Complexity of D_{RegLC}, SDALT as well as lcSDALT when a feasible potential function is used is equal to the complexity of the Dijkstra's algorithm on the product graph P which is $O(m \log n)$ where $m = |A||S|^2$ and $n = |V||S|$ are the number of arcs and nodes of P. 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 (line 18-20) 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): $O(mn \log n)$.

4.2 Constrained landmark distances

For calculating the distance bounds for a generic node (v, s) of P, we 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 are used to constrain the landmark distance calculation (see Table 1). The languages produced by procedure 1 allows every combination of labels in Σ_0. The language produced by procedure 2 is similar but depends on the state s of the node (v, s). It allows every combination of labels in Σ_0 except those labels for which there is no longer any transition between states which are reachable from s. Procedure 3 is the trickiest and produces four distinct languages for a node (v, s) of P. $L_s^{\ell \to t}$ is determined in such a way that all constraints given by $\mathcal{A}_0$ on shortest paths which pass by a node (v, s) on $\mathcal{A}_0$ will be considered. $L_s^{v \to \ell}$ considers only those constraints that apply to paths starting from (v, s) to a final node. $L_s^{\ell \to v}$ and $L_s^{t \to \ell}$ are derived in such a way that they put as few constraints as possible on the distance calculation but assure that Proposition 1 is valid. Consider, e.g., a transportation network. The procedures are based on the intuition that fast transportation modes which are excluded by L_0 should also not be used to calculate the bounds. Also, if there are constraints which infer a major detour from the unconstrained shortest path, this detour should also be considered by the landmark distance calculation.

In [14] these procedures have been used with SDALT to produce three methods which assure that reduced costs are always non-negative; a basic method (bas) which applies procedure 1 to all nodes (v, s) of P; an advanced method (adv) which applies procedure 2 to all nodes and uses a slightly modified potential function; a specific method (spe) which applies procedure 3 to all nodes. These methods can also be used with lcSDALT. In the following we will present two new methods which can only be used with lcSDALT, as reduced costs may be negative: an adapted version of (adv) which we will call (adv_{lc}) and a new method (mix_{lc}).

(adv_{lc}) Equally to (adv), this method applies procedure 2 to all nodes (v, s) of P. Differently to (adv) it uses Equation 1 as potential function and thereby considerably reduces the number of potentials to be calculated.

(mix_{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

proc. regular language and/or NFA	
1	$L_s^{v\to\ell} = L_s^{t\to\ell} = L_s^{\ell\to v} = L_s^{\ell\to t} = L_{\mathsf{proc1}} = \{\Sigma_0^*\}$
	$L_{\mathsf{proc1}} : \mathcal{A}_{\mathsf{proc1}} = (\{s\}, \Sigma_0, \delta : \{s\} \times \Sigma_0 \to \{s\}, s, \{s\})$
2	$L_s^{v\to\ell} = L_s^{t\to\ell} = L_s^{\ell\to v} = L_s^{\ell\to t} = L_{\mathsf{proc2,s}} = \{\overrightarrow{\Sigma}(s, \mathcal{A}_0)^*\}$
	$L_{\mathsf{proc2,s}} : \mathcal{A}_{\mathsf{proc2,s}} = (\{s\}, \overrightarrow{\Sigma}(s, \mathcal{A}_0), \delta : \{s\} \times \Sigma(s, \mathcal{A}_0) \to \{s\}, s, \{s\})$
3	$L_s^{\ell\to v} : \mathcal{A}_s^{\ell\to v} = (S = \overleftarrow{S}(s, \mathcal{A}_0), \Sigma = \overleftarrow{\Sigma}(s, \mathcal{A}_0), \delta_0\vert_{S\times\Sigma}), s, \{s\})$
	$L_s^{\ell\to t} : \mathcal{A}_s^{\ell\to t} = (S = \overleftarrow{S}(s, \mathcal{A}_0) \cup \overrightarrow{S}(s, \mathcal{A}_0),$
	$\Sigma = \overleftarrow{\Sigma}(s, \mathcal{A}_0) \cup \overrightarrow{\Sigma}(s, \mathcal{A}_0), \delta_0\vert_{S\times\Sigma}, s, F_0 \cap S)$
	$L_s^{v\to\ell} : \mathcal{A}_s^{v\to\ell} = (S = \overrightarrow{S}(s, \mathcal{A}_0), \Sigma = \overrightarrow{\Sigma}(s, \mathcal{A}_0), \delta_0\vert_{S\times\Sigma}), s, F_0 \cap S)$
	$L_s^{t\to\ell} : \mathcal{A}_s^{t\to\ell} = (\{s\}, \Sigma = \bigcap s \in F_0 \cap \overrightarrow{S}(s, \mathcal{A}_0)\{l \in \Sigma_0 : \exists\delta_0(s, l) = \{s\}\},$
	$\delta : s \times \Sigma = \{s\}, s, \{s\})$

Table 1: With reference to a generic RegLCSP where the SP is constrained by regular language L_0 ($\mathcal{A}_0 = (S_0, \Sigma_0, \delta_0, s_0, F_0)$) the table shows three procedures to determine the regular language to constrain the distance calculation for a generic node (n, s) of the product graph P.

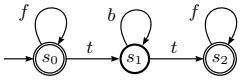
with certain states may be worse than those produced by procedure 2. This is why we introduce a more flexible new method (mix_{lc}) which provides the possibility to freely choose for each state between the application of procedure 2 or procedure 3. This also often gives a trade-off between memory requirements and performance improvement. To find the desired calibration for a given L_0 and to determine when to use procedure 2 or procedure 3 some testing is necessary.

Memory requirements and time-dependency Memory requirements to hold preprocessing data for (adv_{lc}) grow linearly in respect to $|\mathcal{L}| \times |V| \times |S|$. ($\text{mix}_{\mathsf{lc}}$) may require in the worst case up to four times more space than (adv_{lc}). Similar to D, D_{RegLC} and also lcSDALT can easily be adapted to the time-dependent scenario by selecting landmarks and calculating landmark distances by using the *minimum weight cost function* $\overline{c}_{ijl} = \min_\tau c_{ijl}(\tau)$. Indeed, potentials stay valid as long as arc weights only increase and do not drop below a minimal value [2, 6].

4.3 Experimental Results

The algorithm lcSDALT is implemented in C++ and compiled with GCC 4.1. A binary heap is used as priority queue. Similar to the basic version of uniALT [16], periodical additions of landmarks (max 6 landmark) take place. Experiments are run on a Intel Xeon, clocked at 2.6 Ghz, with 16 GB of RAM.

The multi-modal transportation network is based on road and public transportation data of the French region Ile-de-France. It consists of five layers: private bike, rental bike, walking, car and public transportation. Each layer is connected to the walking layer through transfer arcs which model the time needed to transfer from one layer to another (e.g., the time needed to unchain and mount a bike). Each arc has exactly one associated label, e.g. f for arcs representing foot paths, pt_{r} for train



(a) $\mathcal{A}_0$: Automaton allows walking (f) and biking (b). Once the bike is discarded (state s_2) it may not be used again. $S_0 = \{s_0, s_1, s_2\}$, initial state s_0, $F = \{s_0, s_2\}$, $\Sigma_0 = \{f, b, t\}$.

$$L_0 : f^*|(f^*tb^*tf^*)$$

(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.

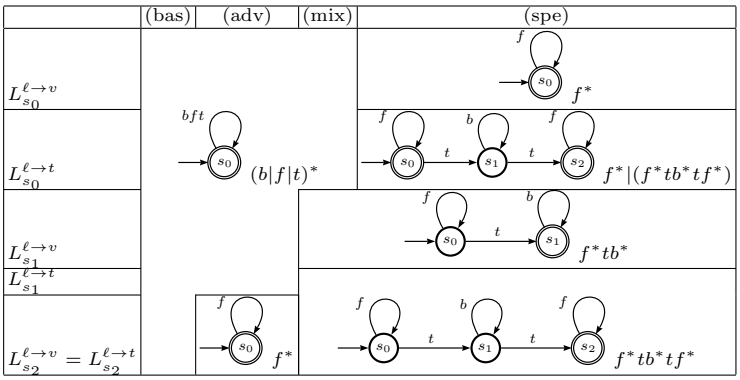


Fig. 3: Example of a regular language L_0 (scenario A) and its representation as an automaton (3a) and regular expression (3b). The table lists the languages used to constrain the landmark distance calculation by applying (bas), (adv), (mix), and (spe). E.g., for (adv): $L_{s_0}^{\ell\to v} = L_{s_0}^{\ell\to t} = L_{s_1}^{\ell\to v} = L_{s_1}^{\ell\to t} : (b|f|t)^*$, $L_{s_2}^{\ell\to v} = L_{s_2}^{\ell\to t} : f^*$.

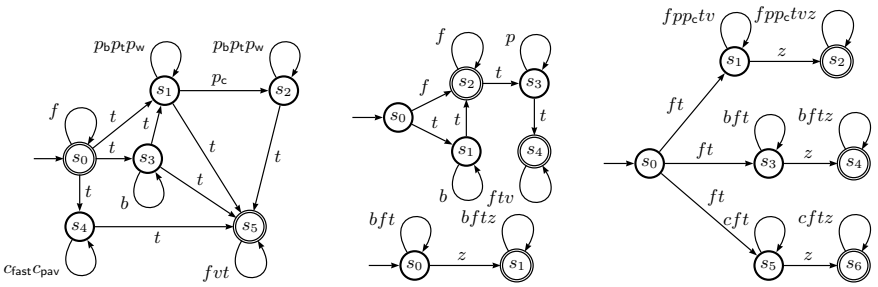
tracks, c_{toll} for toll roads. The graph consists off circa 4.1mil arcs and 1.2mil nodes. Dimensions of the graph and a list off all used labels are given in Table 2.

Data of the public transportation network has been provided by STIF[4]. It includes geographical information, as well as timetable data on bus lines, tramways, subways and regional trains. Our model is similar to the one presented in [18]. Data for the car layer is based on road and traffic information provided by Mediamobile[5]. Arc labels and travel time are set according to the road type. Circa 15% of the road arcs have a time-dependent cost function to represent changing traffic conditions throughout the day. Transfers from the car layer to the walking layer are possible at uniformly distributed transfers arcs. The walking as well as the private and rental bike layers are based on road data (walking paths, cycle paths, etc.) extracted from geographical data freely available from OpenStreetMap. Arc cost equals walking or biking time. Arcs are replicated and inserted in each of the layers if both walking and biking are possible. The rental bike layer includes only arcs in the area around the city of Paris, where the rental bike service[6] is available. Bike rental stations serve as connection points between the walking layer and the rental bike layer as rental bikes have to be picked up and returned at bike rental stations. The private bike layer is connected to the walking layer at common street intersections. In addition,

[4] Syndicat des Transports IdF, www.stif.info, data for scientific use (01/12/2010)
[5] www.v-trafic.fr, www.mediamobile.fr
[6] Vélib', www.velib.paris.fr

layer	nodes	arcs	labels
walking	275 606	751 144	f (all arcs except 20 arcs with label z)
public transportation	109 922	292 113	p_{b} (bus, 72 512 arcs), p_{m} (metro, 1 746), p_{r} (tram, 1 746), p_{t} (train, 8 309), pt_{c} (connection between stations, 32 490), pt_{w} (walking station intern, 176 790), time-dependent 82 833
private bike	250 206	583 186	b
rental bike	38 097	83 928	v
car	613 972	1 273 170	c_{toll} (toll roads, 3 784), c_{fast} (fast roads, 16 502), c_{pav} (paved roads except toll and fast roads, 1 212 957), c_{unpav} (unpaved roads, 27 979), time-dependent 188 197
transfers	-	1 107 457	t (walking↔private bike 493 601, walking↔rental bike 2 404, walking↔car 572 604, walking↔public transportation 38 848)
Tot	1 287 803	4 095 971	time-dependent arcs 271 030 (7 687 204 Time Points)

Table 2: Dimensions of the graph



(a) Scenario B (b) Scenarios C and E (c) Scenario D

Fig. 4: Instances used for experimental evaluation. Note that labels $p_{\mathsf{b}}p_{\mathsf{m}}p_{\mathsf{r}}p_{\mathsf{t}}p_{\mathsf{w}}$ have been substituted by p and $c_{\mathsf{toll}}c_{\mathsf{fast}}c_{\mathsf{pav}}$ by c.

we introduced ten arcs with label z between nodes of the foot layer. They represent foot paths 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.

We ran 500 test instances for five RegLCSP scenarios, see Figures 3a and 4. They have been chosen with the intention to represent real-world queries, which may arise when looking for constraint shortest paths on a multi-modal transportation network. Source node r, target node t, and start time τ_{start} are picked at random. r and t always belong to the walking layer. We use 32 landmarks which are placed exclusively on the walking layer. Preprocessing for the different scenarios takes less than a minute. For each instance we compare running times of the different methods with D_{RegLC}. See Table 3 for experimental results. *time* is the average running time in msec of the algorithm over 500 test instances. *SettNo*, *reInsNo* and *calcPot* give the average of the number of settled nodes, re-inserted nodes and calculated potentials. *MaxSett* gives the maximum number of settled nodes. For $(\text{mix})_{\mathsf{lc}}$ all states for which procedure 2 has been applied is given. For all other states procedure 3 has been used.

sce.	method	time	speed-up	settNo	maxSett	reInsNo	nCalPot	size	proc 2
A	D_{RegLC}	238	1.0	296 656	986 292	-	-	0	
	(bas)	19	12.4	21 476	950 992	0	76 144	311	
	(adv)	18	13.6	10 930	684 296	0	129 020	622	
	$(\text{adv})_{\mathsf{lc}}$	13	18.1	10 927	682 981	11	68 820	622	
	(spe)	62	3.9	45 548	202 382	0	272 174	1 244	
	$(\text{mix})_{\mathsf{lc}}$	13	*18.9	11 034	254 259	19	74 544	933	s_1
B	D_{RegLC}	624	1.0	610 375	1 487 766	-	-	0	
	(bas)	219	2.9	156 828	973 345	0	676 969	311	
	(adv)	190	3.3	51 258	346 549	0	2 295 030	1 866	
	$(\text{adv})_{\mathsf{lc}}$	174	3.6	145 535	857 619	30	623 198	1 866	
	(spe)	237	2.6	176 766	667 125	0	543 388	2 177	
	$(\text{mix})_{\mathsf{lc}}$	89	*7.0	60 053	350 237	87	343 585	1 244	s_0, s_3, s_4
C	D_{RegLC}	630	1.0	658 738	1 785 747	-	-	0	
	(bas)	515	1.2	414 553	1 721 989	0	1 981 520	311	
	(adv)	338	1.9	158 193	977 793	0	2 437 770	933	
	$(\text{adv})_{\mathsf{lc}}$	263	2.4	223 070	2 217 083	64 034	919 554	933	
	(spe)	238	2.7	156 739	929 544	0	633 859	1 866	
	$(\text{mix})_{\mathsf{lc}}$	149	*4.2	91 917	499 646	1 064	674 115	1 244	s_0, s_1, s_2, s_4
	$(\text{mix})_{\mathsf{lc}}$	161	3.9	97 119	620 991	2 564	686 516	933	s_0, s_1, s_2
D	D_{RegLC}	1 722	1.0	1 248 060	3 085 428	-	-	0	
	(bas)	565	3.0	373 695	1 830 094	0	1 731 400	311	
	(adv)	695	2.5	352 969	1 557 331	0	3 297 090	1 244	
	$(\text{adv})_{\mathsf{lc}}$	558	3.1	356 679	1 575 981	40	1 640 950	1 244	
	(spe)	476	3.6	337 849	1 791 504	0	734 068	2 488	
	$(\text{mix})_{\mathsf{lc}}$	364	*4.7	221 631	1 278 208	48	953 802	2 177	s_0
E	D_{RegLC}	764	1.0	795 822	1 458 519	-	-	0	
	(bas)	143	5.3	115 941	598 273	0	407 108	311	
	(adv)	119	6.4	115 941	598 273	0	411 869	311	
	$(\text{adv})_{\mathsf{lc}}$	120	6.4	116 042	598 273	32	406 877	311	
	(spe)	25	30.3	27 532	216 389	0	109 125	622	
	$(\text{mix})_{\mathsf{lc}}$	22	*34.4	23 607	145 308	608	85 776	933	s_0

Table 3: Experimental results

4.4 Discussion of experimental results and conclusive remarks

The algorithm lcSDALT, by adopting the five methods (bas), (adv), (adv_{lc}), (spe) and (mix_{lc}), in comparison to D_{RegLC}, succeeds in directing the constrained shortest path search faster toward the target. (bas) works well in situations where L_0 excludes a priori fast transportation modes (scenario A). (adv) gives a supplementary speed-up in cases where initially allowed fast transportation modes are excluded from a later state on A_0 onward. (spe) has a positive impact on running times for scenarios where the visit of some infrequent labels is imposed by L_0 (E) or the use of fast transportation modes is somehow limited (B). Finally, the new methods (adv_{lc}) and (mix_{lc}) prove to be very efficient. (adv_{lc}) runs up to 20% faster than (adv) as it substantially reduces the number of calculated potentials, especially for larger automata. The extra flexibility provided by (mix_{lc}) often results in a reduction of running time (up to 50%) and a substantial reduction of memory space.

Recent works on finding constrained shortest paths on multi-modal networks reported speed-ups of different orders of magnitude. They succeed in doing this by using contraction hierarchies and by either limiting the constraints which can be imposed on the shortest paths [19] or by identifying homogenous regions (arcs with the same label) of the network and by applying contractions only to those regions [7]. lcSDALT does not provide such speed-ups but proves to work consistently good even when considering more difficult constraints then the one considered in [19] and on a highly dis-homogenous graph. Also, time-dependent cost functions on arcs can

be easily incorporated. Therefore, lcSDALT seems a good ingredient to future more advances algorithms on multi-modal networks.

References

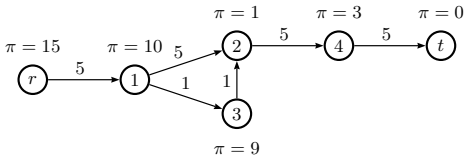
1. C. Barrett, K. Bisset, M. Holzer, G. Konjevod, M. Marathe, and D. Wagner. Engineering label-constrained shortest-path algorithms. *Algorithmic Aspects in Information and Management*, pages 27–37, 2008.
2. C. Barrett, K. Bisset, R. Jacob, G. Konjevod, and M. V. Marath. Classical and contemporary shortest path problems in road networks: Implementation and experimental analysis of the TRANSIMS router. *Proc. ESA*, pages 126–138, 2002.
3. C. Barrett, R. Jacob, and M. Marathe. Formal-Language-Constrained Path Problems. *SIAM Journal on Computing*, 30(3):809, 2000.
4. D. Delling, T. Pajor, and D. Wagner. Accelerating Multi-Modal Route Planning by Access-Nodes. *Algorithms-ESA 2009*, 2, 2009.
5. D. Delling, P. Sanders, D. Schultes, and D. Wagner. Engineering route planning algorithms. *Algorithmics of large and complex networks*, 2:117–139, 2009.
6. D. Delling and D. Wagner. Time-Dependent Route Planning. *Online*, 2:207–230, 2009.
7. J. Dibbelt, T. Pajor, and D. Wagner. User-Constrained Multi-Modal Route Planning. In *Proceedings of the 14th Workshop on Algorithm Engineering and Experiments (ALENEX'12)*. SIAM, 2012.
8. E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1(1):269–271, Dec. 1959.
9. A. V. Goldberg and C. Harrelson. Computing the shortest path: A search meets graph theory. In *Proceedings of the 16th annual ACM-SIAM Symposium on Discrete algorithms*, pages 156–165. SIAM, 2005.
10. A. V. Goldberg and R. Werneck. Computing point-to-point shortest paths from external memory. *US Patent App. 11/115,558*, 2005.
11. P. Hart, N. Nilsson, and B. Raphael. A Formal Basis for the Heuristic Determination of Minimum Cost Paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968.
12. T. Ikeda, H. Imai, S. Nishimura, H. Shimoura, T. Hashimoto, K. Tenmoku, and K. Mitoh. *A fast algorithm for finding better routes by AI search techniques*. IEEE, 1994.
13. D. Kaufman and R. Smith. Fastest paths in time-dependent networks for intelligent vehicle-highway systems applications. *Journal of Intelligent Transportation Systems*, 1(1):1–11, 1993.
14. D. Kirchler, L. Liberti, T. Pajor, and R. Wolfler Calvo. UniALT for Regular Language Constrained Shortest Paths on a Multi-Modal Transportation Network. In *ATMOS 2011*, pages 64–75. Schloss Dagstuhl, 2011.
15. A. O. Mendelzon and P. T. Wood. Finding Regular Simple Paths in Graph Databases. *SIAM Journal on Computing*, 24(6):1235, 1995.
16. G. Nannicini, D. Delling, L. Liberti, and D. Schultes. Bidirectional A search for time-dependent fast paths. *Experimental Algorithms*, 2(2):334–346, 2008.
17. A. Orda and R. Rom. Shortest-path and minimum-delay algorithms in networks with time-dependent edge-length. *Journal of the ACM*, 37(3):607–625, July 1990.
18. E. Pyrga, F. Schulz, D. Wagner, and C. Zaroliagis. Efficient models for timetable information in public transportation systems. *Journal of Experimental Algorithmics*, 12(2):1–39, June 2007.
19. M. Rice. Graph indexing of road networks for shortest path queries with label restrictions. *Proceedings of the VLDB Endowment*, pages 69–80, 2010.
20. J. Romeuf. Shortest path under rational constraint. *Information processing letters*, 28(5):245–248, 1988.

A Proofs

Lemma 2. *The priority queue always contains a node (i, s') with key $k(i, s') = d^*_r(i, s') + \pi(i, s')$ and which belongs to the shortest path from (r, s_0) to (t, s'') where $s'' \in F_0, s' \in S$.*

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 P (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 $k(r, s) = d^*_r(r, s) + \pi(r, s) = \pi(r, s)$. When node p_n with $k(i, s) = d^*_r(i, s) + \pi(i, s)$ for some $n \in \{1, \ldots, m\}$ is extracted from the priority queue, at least one new node $p_{n+1} = (j, s')$ with $d_r(j, s') = d^*_r(j, s') = d^*_r(i, s) + c_{(i,s)(j,s')l}(\tau)$ is inserted in the queue by lines 14, 18, 21. □

B Suppressed figures



step 1 insert node r in Q with key 15
step 2 extract node r from Q and insert node 1 in Q with key 15
step 3 extract node 1 from Q and insert node 2 in Q with key 11
and insert node 3 in Q with key 15
step 4 extract node 2 from Q and insert node 4 in Q with key 18
step 5 extract node 3 from Q and insert node 2 in Q with key 8
step 6 extract node 2 from Q and decrease node 3 in Q with key 15
step 7 extract node 4 from Q and insert node t in Q with key 15
step 8 extract node t from Q and terminate

Fig. 5: Application of algorithm lcSDALT without constraints. Arc from node 1 to node 2 has negative reduced cost $5 + 1 - 10 = -4$ and as a result node 2 is inserted twice in the priority queue (P).

method	characteristics
(adv_{lc})	lcSDALT uses Equation 1, SDALT for (adv) must apply a modified potential function: $\pi_{\text{adv}}(v, s) = \max\{\pi(v, s_x) \mid s_x \in \Omega(s, \mathcal{A}_0)\}$
(mix_{lc})	mixes procedures 2 and 3

Table 4: Characteristics of (adv_{lc}) and (mix_{lc})

algo	regular language to constrain distance calculation $L_{s_0}^{t\to\ell}$ | $L_{s_0}^{v\to\ell}$ | $L_{s_1}^{t\to\ell}$ | $L_{s_1}^{v\to\ell}$	settled nodes on P (key in square brackets)
D_{RegLC}	no potential	all
std	not constrained	$(r, s_0)[9], \ldots$ all except $(1, s_0)$, $(2, s_0)$
bas	$(f\|b)^*$	$(r, s_0)[10]$, $(5, s_1)[10]$, $(6, s_1)[10]$,$(7, s_1)[11]$, $(8, s_1)[11]$, $(t, s_1)[11]$
adv	$(f\|b)^*$ | f^*	$(r, s_0)[10]$, $(5, s_1)[10]$, $(7, s_1)[11]$, $(8, s_1)[11]$, $(t, s_1)[11]$
spe	f^*bf^* | f^*	$(r, s_0)[11]$, $(7, s_1)[11]$, $(8, s_1)[11]$,$(t, s_1)[11]$

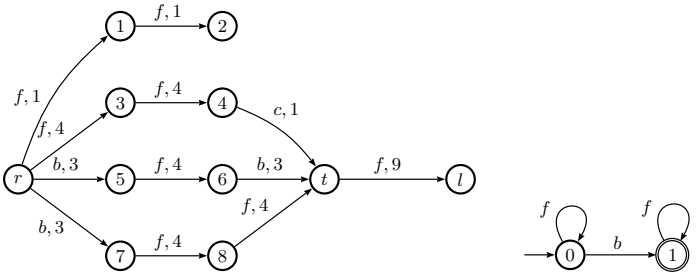


Fig. 6: Example showing the application of lcSDALT on a labeled graph. The shortest paths is constrained by the regular expression f^*bf^* (NFA on the right of the graph). The table shows the regular languages used to constrain the landmark distance calculation, as well as the search space. Optimal path: (r, s_0), $(9, s_1)$, $(10, s_1)$, (t, s_1). Node l is the landmark.

C Further details for instances

Here we provide more information about the instances used for experimental evaluation. Note that labels $p_{\mathsf{b}}p_{\mathsf{m}}p_{\mathsf{r}}p_{\mathsf{t}}p_{\mathsf{w}}$ have been substituted by p and $c_{\mathsf{toll}}c_{\mathsf{fast}}c_{\mathsf{pav}}$ by c.

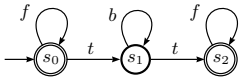


Fig. 7: Scenario A: The target may be reach be either walking or walking and taking a private bike. Once discarded, the bike may not be used again.

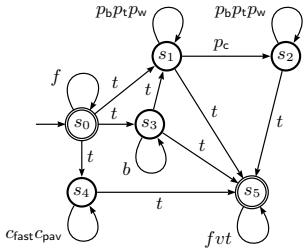


Fig. 8: Scenario B: The target may be reached by walking or by using a private car (only paved and fast roads are allowed, no toll roads). The target may also be reached by private bike or/and by taking public transportation (no metro allowed). Maximal one change of public transportation vehicles is permitted. Once discarded the private car or private bike or left public transportation, the itinerary may be continued by walking or by taking a rental bike.

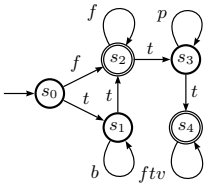


Fig. 9: Scenario C: The target may be reached by walking or by taking a private bike. Public transportation may be used once but without changing vehicles. Once left public transportation, the itinerary may be continued by walking or by taking a rental bike.

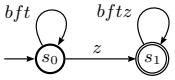


Fig. 10: Scenario E: At any point of the itinerary taking a private bike or walking is possible. An arc with label z has obligatorily to be visited. It may represent a street with a grocery shop, a post office, etc

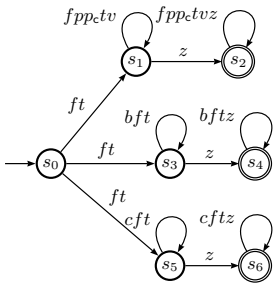


Fig. 11: Scenario D: An arc with label z has obligatorily to be visited. It may represent a street with a grocery shop, a post office, etc. Other than that either public transportation or private bike or private car may be used.