UniALT for Regular Language Constrained Shortest Paths on a Multi-Modal Transportation Network

Dominik Kirchler[1,2,3], Leo Liberti[1], Thomas Pajor[4], and 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, Ivry sur Seine, France
4 Karlsruhe Institute of Technology
pajor@kit.edu

Abstract

Shortest paths on road networks can be efficiently calculated using Dijkstra's algorithm (D). In addition to roads, multi-modal transportation networks include public transportation, bicycle lanes, etc. For paths on this type of network, further constraints, e.g., preferences in using certain modes of transportation, may arise. The regular language constrained shortest path problem deals with this kind of problem. It uses a regular language to model the constraints. The problem can be solved efficiently by using a generalization of Dijkstra's algorithm (D_{RegLC}). In this paper we propose an adaption of the speed-up technique uniALT, in order to accelerate D_{RegLC}. We call our algorithm SDALT. We provide experimental results on a realistic multi-modal public transportation network including time-dependent cost functions on arcs. The experiments show that our algorithm performs well, with speed-ups of a factor 2 to 20.

1998 ACM Subject Classification G.2.2 Graph Theory

Keywords and phrases Time-Dependency, ALT, Regular Language, Shortest Path, Multi-Modal

Digital Object Identifier 10.4230/OASIcs.ATMOS.2011.64

1 Introduction

Shortest paths on road networks can be efficiently calculated using Dijkstra's algorithm [8]. In addition to roads, multi-modal transportation networks include public transportation, walking paths, bicycle lanes, etc. Paths on this type of network may require a number of restrictions and/or preferences in using certain modes of transportation. Passengers may be willing to take trains, but not buses. Whereas distances can be covered by walking at almost any point during an itinerary, some transportation modes such as private cars and bikes, once discarded, might not be available again at a later point in the itinerary. More general constraints, such as passing by any pharmacy or post office on the way to the target destination, may also arise.

In order to deal with this problem, appropriate labels are assigned to the arcs of the network and the additional constraints are modeled as a regular language. A valid shortest path minimizes some cost function (distance, time, etc.) and, in addition, the word produced by concatenating the labels on the arcs of the shortest path must form an element of the

11th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems.
Editors: Alberto Caprara & Spyros Kontogiannis; pp. 64–75
OpenAccess Series in Informatics
OASICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

regular language. The problem is called regular language constrained shortest path problem (RegLCSP). An in-depth theoretical study of a more general problem, the formal language constrained shortest path problem, as well as a generalization of Dijkstra's algorithm (D_{RegLC}) to solve RegLCSP can be found in [3].

In recent years much effort has been spent to produce speed-up techniques for Dijkstra's algorithm (D) and shortest paths on continental sized road networks can now be found in a few milliseconds [6]. D_{RegLC} has received less attention. First attempts to adapt speed-up techniques of D to D_{RegLC} have been described in [1].

Our Contribution In this paper, we propose an adaption of the speed-up technique uniALT [9], in order to accelerate D_{RegLC}. UniALT uses preprocessed data to guide D faster toward the target. The idea is to adapt uniALT to D_{RegLC} by transferring information of the regular language of the RegLCSP instance into the preprocessing phase of uniALT. For each instance of RegLCSP, we produce specific preprocessed data which guides D_{RegLC}. We call this algorithm SDALT (State Dependent uniALT). We provide experimental results on a realistic multi-modal public transportation network. It is composed of the road and public transportation network of the French region Ile-de-France which includes the city of Paris and consists of five layers: private bike, rental bike, walking, car (including changing traffic conditions over the day), and public transportation. To our knowledge, this is the first work to consider a multi-modal network in this configuration and on this scale. The experiments show that our algorithm performs well, with speed-ups of a factor 2 to 20, in respect to plain D_{RegLC}, 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.

2 Related work

Early works on the use of regular languages as a model for constrained shortest path problems include [21, 15, 23], with applications to database queries. A finite state automaton is used in [14] to model path constraints (called path viability) on a multi-modal transportation network for the bi-objective multi-modal shortest path problem. Algorithmic and complexity-theoretical results on the use of various types of languages for the label 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}). Experimental data on networks including time-dependent edge cost can be found in [2, 22].

In recent years, much focus has been given on accelerating the mono-modal shortest path problem on large road graphs. There are three basic ingredients to most modern speed-up techniques: bi-directional search, goal-directed search, and contraction. See [6] for a comprehensive overview.

ALT is a bi-directional, goal directed search technique based on the A^* 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 ALT. Efficient implementations of uniALT and ALT as well as experimental data on continental size road networks with time-dependent edge cost are given in [16]. A^* and ALT can be easily adapted to dynamic networks. Efficient algorithms including contractions can be found in [17, 4].

In [1], bi-directional and goal-directed speed-up techniques have been applied to D_{RegLC} on a multi-modal network. Results vary in function of the regular language used. The authors of [19, 5] observe that ALT in combination with contraction yields only mild speed-ups in a multi-modal context. They propose a method called Access Node Routing to isolate the public transportation network from road networks so that they can be treated individually.

Overview This paper is organized as follows. Section 3 will give more details about the graph model, uniALT, and the generalisation of Dijkstra's algorithm which is used to solve the RegLCSP. Section 4 presents SDALT and its implementation. Its application to a multi-modal transportation network and computational results are presented in section 5. Section 6 concludes our work along with directions for future research.

3 Preliminaries

Consider a directed graph $G = (V, A)$ consisting of a set of nodes $v \in V$ and a set of arcs $(i, j) \in A$ with $i, j \in V$. Arc costs are positive and represent travel times. They may be time-independent or time-dependent. Time-independent costs for arc (i, j) are given by c_{ij}. To model time-dependent arc costs, we use a positive function $c_{ij} : \mathbb{R}_+ \to \mathbb{R}_+$. We only use functions which satisfy the FIFO property as the time-dependent shortest path problem in FIFO networks are polynomially solvable [13], whereas it is NP-hard in non-FIFO networks [18]. 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 $(v_i, v_{i+1}) \in A$ for all $1 \leq i < k$. The cost of the path in a time-independent scenario is given by $c(p) = \sum_{i=1}^{k-1} c_{v_i v_{i+1}}$. We denote as $d(r, t)$ the cost of the *shortest path* between nodes r and t. 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 $\gamma((v_1, \ldots, v_j), \tau) = \gamma(v_1, \ldots, v_{j-1}, \tau)) + c_{v_{j-1}, v_j}(\gamma(v_1, \ldots, v_{j-1}, \tau))$.

3.1 A^* and uniALT algorithm

The A^* algorithm [11] 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. A^* is similar to Dijkstra's algorithm [8], which we shall denote as D throughout our paper. The difference lies in the order of selection of the next node v to be settled. A^* employs a key $k(v) = d_r(v) + \pi(v)$ where the *potential function* $\pi : V \to \mathbb{R}$ gives an under-estimation of the distance from v to t. $d_r(v)$ gives the *tentative* distance from r to v. 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. In [12], it is shown that A^* is equivalent to D on a graph with *reduced arc costs* $c^{\pi}_{vw} = c_{vw} - \pi(v) + \pi(w)$. D 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 $v, w \in V$. $\pi(v)$ 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 [9].

Good bounds can be produced by using landmarks and the triangle inequality [9]. The main idea is to select a small set of nodes $\ell \in \mathcal{L} \subset V$, spread appropriately over the network, and precompute all distances of shortest paths $d(\ell, v)$ and $d(v, \ell)$ between these nodes (*landmarks*) and any other node $v \in V$, by using D. 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. $\pi(v) =$

$\max_{\ell \in \mathcal{L}}(d(v, \ell) - d(t, \ell), d(\ell, t) - d(\ell, v))$ gives a lower bound for the distance $d(v, t)$ and is a feasible potential function. The A^* algorithm based on this potential function is called uniALT [9]. As observed in [7], potentials stay feasible as long as arc weights only increase and do not drop below a minimal value. Based on this, uniALT 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 [9, 10]. UniALT provides a speed-up of about factor 10 on road graphs with time-dependent arc costs [7].

3.2 Solving the RegLCSP

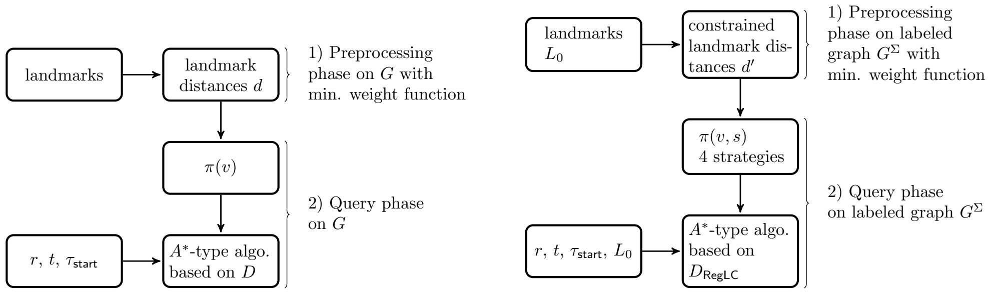
Consider a *labeled* graph $G^\Sigma = (V, A)$. It is produced by associating a label l of a set of labels Σ to each arc (e.g., f to mark foot-paths or b to mark bicycle lanes). A is a set of triplets in $V \times V \times \Sigma$. (i, j, l) represents an arc from node i to node j having label l. The RegLCSP consists in finding a shortest path from a source node r to a target node t with starting time τ_{start} on G^Σ 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. This language can be described by a non-deterministic finite state automaton $\mathcal{A}_0 = (S, \Sigma_0, \delta, s_0, F)$, consisting of a set of states S, a set of labels $\Sigma_0 \subseteq \Sigma$, a transition function $\delta : \Sigma_0 \times S \to 2^S$, an initial state s_0, and a set of final states F. E.g., consider a labeled graph which consists of arcs with labels $\Sigma = \{b, c, f, p, v, t\}$ representing each a different transportation mode. The automaton in Figure 3 describes a regular language with five states $S = \{s_0, s_1, s_2, s_3, s_4\}$, an initial state s_0, a set of final states $F = \{s_2, s_4\}$, and an alphabet $\Sigma_0 = \{b, f, p, v, t\}$.

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 D to the product graph $P = G^\Sigma \times S$ with nodes (v, s) 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 $(i, j, l) \in A$ and a transition such that $s' \in \delta(l, s)$. 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. Similarly to D, D_{RegLC} can easily be adapted to the time-dependent scenario as shown in [2].

4 State Dependent uniALT: SDALT

To speed up D_{RegLC}, the authors of [1] employ A^* and bidirectional search. In this work, we extend uniALT to speed-up D_{RegLC} on a graph G^Σ with time-dependent arc costs and call the resulting algorithm SDALT. It consists of a preprocessing phase and a query phase (see Figure 1). The key of the performance of the algorithm lies in the proposed *constrained landmark distances*, which are used to calculate the potential function.

Preprocessing phase A set of landmarks $\ell \in \mathcal{L} \subset V$ is selected by using the *avoid* heuristic [9]. Then the costs of the shortest paths between all $v \in V$ and each landmark ℓ on G^Σ where arcs are weighted by the minimum weight cost function are determined. Here lies one of the major differences between SDALT and uniALT. Differently from uniALT, SDALT does not use D to determine landmark distances but uses instead the D_{RegLC} algorithm. In this way, it is possible to constrain the cost calculation by some regular languages which we will derive from L_0. We refer to these costs as *constrained landmark distances* $d'(i, j, s)$, which is the travel time of the shortest path from (i, s) to (j, s_j) for some $s_j \in F$ *constrained* by the



(a) UniALT **(b)** SDALT

Figure 1 Comparison uniALT and SDALT

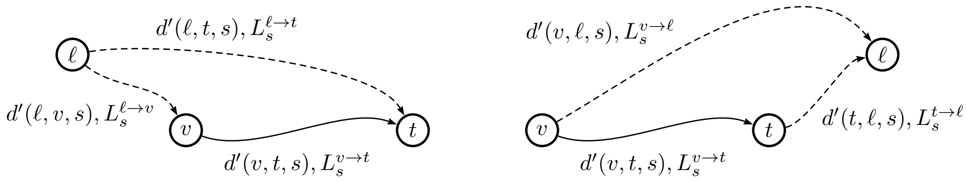


Figure 2 Landmark distances for SDALT

regular language $L_s^{i\to j}$. In the next section, we will provide four different methods 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 $d'(\ell,t,s)$, $d'(\ell,v,s)$, $d'(v,\ell,s)$, $d'(t,\ell,s)$ (see Figure 2).

Potential function $\pi(v,s)$ The constrained landmark distances determined during the preprocessing phase are used to calculate the potential function $\pi(v,s)$ given in Equation (1) and to provide a lower bound on the distance $d'(v,t,s)$ of the shortest path from (v,s) to (t,s_t) for some $s_t \in F$. Note that $d'(v,t,s)$ is constrained by $L_s^{v\to t} = L_0^s$. L_0^s is equal to L_0 except that the initial state s_0 of L_0 is replaced by s. Intuitively, it represents the *remaining* constraints of L_0 to be considered for the shortest path from an arbitrary pair (v,s) to the target.

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

Query phase The query phase deploys a D_{RegLC} algorithm enhanced by the characteristics of the A^* algorithm. For each pair (v,s), the query maintains a tentative distance label $d_r(v,s)$ and a parent pair $p(v,s)$. At every iteration, it selects the pair (v,s) with the smallest key $k(v,s) = d_r(v,s) + \pi(v,s)$ and *relaxes* all outgoing arcs of (v,s). D_{RegLC}, in contrast, uses key $k(v,s) = d_r(v,s)$. Relaxing an arc (v,w,l) means calculating $tmp = d_r(v,s) + c_{vwl}(\tau_{start} + d_r(v,s))$, checking cost labels $d_r(w,s') > tmp$, and if that is the case, to set $d_r(w,s') = tmp$ and $p(w,s') = (v,s)$ *for all* states $s' \in \delta(l,s)$. Note that 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 query terminates when a pair (t,s) with $s \in F$ is settled. See Listing 1.

Note that if $\pi(v,s)$ is feasible, all characteristics that we discussed before for uniALT also hold for SDALT. SDALT can be seen as an A^* search on the product graph P using potential function $\pi(v,s)$. Hence, SDALT is correct and terminates always with the correct constrained shortest path.

Listing 1 Pseudo-code SDALT

```
function SDALT(G^Σ, r, t, τ_start, L_0)
   d_r(v,s):= ∞, p(v,s):= −1, path_found:= false
   d_r(r,s_0):= 0, k(r,s_0):= d_r(r,s_0) + π(r,s_0)
   insert (r,s_0) in priority queue Q
   while Q is not empty:
      extract (v,s) with smallest key k from Q
      if v = t and s ∈ F_0:
         path_found:= true, break
      for each (w,s') of (v,s) where (v,w,l) ∈ A, s' ∈ δ(l,s):
         tmp:= d_r(v,s) + c_vwl(τ_start + d_r(v,s)) //time-dependency
         if tmp < d_r(w,s'):
            d_r(w,s'):= tmp
            k(w,s'):= d_r(w,s') + π(w,s')
            p(w,s'):= (v,s)
            if (w,s') not in Q: insert (w,s') in Q
            else: reorder Q
      end for
   end while
```

4.1 Constrained landmark distances

The only open question now is how to produce good bounds which are capable to guide SDALT efficiently toward the target while considering the constraints given by L_0. More formally, how to choose the regular languages $L_s^{\ell\to t}$, $L_s^{\ell\to v}$, $L_s^{v\to\ell}$, $L_s^{t\to\ell}$ used to constrain the calculation of $d'(\ell,t,s)$, $d'(\ell,v,s)$, $d'(v,\ell,s)$, $d'(t,\ell,s)$ in order that $d'(\ell,t,s)-d'(\ell,v,s)$, $d'(v,\ell,s)-d'(t,\ell,s)$ are valid lower bounds for $d'(v,t,s)$ (see Figure 2) and that $\pi(v,s)$ is feasible. Proposition 1 partially answers this question. 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.** For all $s \in S$, if the concatenation of $L_s^{\ell\to v}$ and $L_s^{v\to t}$ is included in $L_s^{\ell\to t}$ ($L_s^{\ell\to v} \circ L_s^{v\to t} \subseteq L_s^{\ell\to t}$), then $d'(\ell,t,s)-d'(\ell,v,s)$ is a lower bound for the distance $d'(v,t,s)$. Similarly, 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 $d'(v,t,s)$.

This is derived from 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). Now we proceed to present four methods on how to set $L_s^{\ell\to t}$, $L_s^{\ell\to v}$, $L_s^{v\to\ell}$, $L_s^{t\to\ell}$. We name these four methods standard (std), basic (bas), advanced (adv), and specific (spe).

(std) In the standard method, the landmark distance calculation is not constrained by any regular language. (std) represents the application of plain uniALT to D_{RegLC}.

(bas) The motivation for the basic method comes from the observation that if L_0 totally excludes the use of some *fast* transportation modes, these modes should not be considered when calculating the landmark distances. This means that (bas) uses $L_s^{\ell\to v} = L_s^{\ell\to t} = L_s^{v\to\ell} = L_s^{t\to\ell} = L_{\mathsf{bas}} = \{\Sigma_0^*\}$, which is the language consisting of all words over Σ_0. E.g., for the RegLCSP with L_0 represented by automaton in Figure 3a the landmark distances calculation would be constrained by using automaton in Figure 3b. In an ideal scenario where one transportation mode, which is excluded by L_0, dominates any other (e.g., bike over foot), it can be proven that (bas) produces better bounds than (std).

▶ **Proposition 2.** Given a labeled graph $G^{\Sigma}_{\mathsf{bas}} = (V, A_1 \cup A_2)$ with $\Sigma = \{\ell_1, \ell_2\}$, where for any two shortest paths $p_1 \subseteq A_1$, $p_2 \subseteq A_2$ between two arbitrary nodes, there exists an $\alpha > 1$ such that $c(p_1) > \alpha c(p_2)$. Arcs in A_1 are labeled ℓ_1 and arcs in A_2 are labeled ℓ_2. For a RegLCSP on $G^{\Sigma}_{\mathsf{bas}}$ exclusively allowing arcs with label ℓ_1, $L_0 = \{\ell_1^*\}$, bounds calculated by using (bas) are at least a factor α greater than bounds calculated using (std).

(adv) The advanced method consists in calculating separate constrained landmark distances for each pair (v, s) by using the regular language $L_s^{\ell\to v} = L_s^{\ell\to t} = L_s^{v\to\ell} = L_s^{t\to\ell} = L_{\mathsf{adv},s} = \{\Sigma(s, \mathcal{A}_0)^*\}$. $\Sigma(s, \mathcal{A}_0)$ returns all labels of Σ_0 *except* those of fast transportation modes which use is no longer allowed from state s onward. This means that for s_0 it includes all transportation modes present in Σ_0, equally to (bas). For the calculation of the constrained landmark distances for the other states $s \in S$ it *excludes* fast transportation modes of Σ_0, *if* from s onward on $\mathcal{A}_0$ these transportation modes may not be used anymore for the remaining path to reach the target. E.g., consider the RegLCSP with L_0 represented by automaton in Figure 3a. By applying (adv) the landmark distances calculation would be constrained be using automata in Figures 3b, 3c, and 3d. From state s_2 onward, private bike cannot be used any more (dominates walking, and sometimes even public transport), from state s_4 also private transport is excluded. Note that by using (adv), $\pi(v, s)$ may be infeasible, so we change it to: $\pi_{\mathsf{adv}}(v, s) = \max\{\pi(v, s_x) | s_x \in \Omega(s, \mathcal{A}_0)\}$, where $\Omega(s, \mathcal{A}_0)$ returns the set containing all states $s_x \in S$ from which s is reachable by some sequence of transitions on $\mathcal{A}_0$, including s. E.g., in reference to the automaton in Figure 3a, $\Omega(s_0, \mathcal{A}_0) = \{s_0\}$ whereas $\Omega(s_2, \mathcal{A}_0) = \{s_0, s_1, s_2\}$. In an ideal scenario where transportation modes hierarchically dominate each other (car over taxi over trains over biking over walking) and in which they are excluded in decreasing order of speed by advancing on $\mathcal{A}_0$ it can be proven, by generalizing Proposition 2, that (adv) produces better bounds than (bas).

(spe) Besides using L_0 for gradually excluding transportation modes, it can also be used to impose further restrictions, for example to not allow transfers from one vehicle of public transportation to another. L_0 can also be used to force the shortest path to pass by any arc marked with a certain label. Suppose we are looking for the shortest foot path to a target which also passes by the nearest pharmacy. To handle this problem, we can label all arcs of the foot layer which represent streets on which a pharmacy is located not with f but with z. E.g., L_0, represented by automaton in Figure 4a, imposes the use of the foot layer *and* that an arc with label z has to be obligatorily visited. (spe) is capable of anticipating such constraints in the preprocessing phase by inserting these constraints in the languages used during the landmark distance calculations. We define *four* different regular languages $L_s^{\ell\to v}$, $L_s^{\ell\to t}$, $L_s^{t\to\ell}$, $L_s^{v\to\ell}$ to calculate the constrained landmark distances for each pair (v, s). Consider the following rules to determine $L_s^{\ell\to v}$, $L_s^{\ell\to t}$, $L_s^{v\to\ell}$, $L_s^{t\to\ell}$, which are here represented as automata, and Proposition 3.

Rule 1 $\mathcal{A}_{s_x}^{\ell\to v}$ is the sub-automaton of $\mathcal{A}_0$ consisting of s_x, all the states from which s_x is reachable, and the transitions between these states. Any s which is an initial state in $\mathcal{A}_0$, is also an initial state in $\mathcal{A}_{s_x}^{\ell\to v}$, s_x is a final state.

Rule 2 $\mathcal{A}_{s_x}^{\ell\to t}$ is the sub-automaton of $\mathcal{A}_0$ consisting of all states reachable from s_x and all states from which these states are reachable, including all transitions between these states. Any s which is an initial state in $\mathcal{A}_0$ is also an initial state in $\mathcal{A}_{s_x}^{\ell\to t}$. Any s which is reachable from s_x and is final in $\mathcal{A}_0$ is also final in $\mathcal{A}_{s_x}^{\ell\to t}$.

Rule 3 $\mathcal{A}_{s_x}^{v\to\ell}$ is the sub-automaton of $\mathcal{A}_0$ consisting of s_x, all the states which are reachable from s_x, and the transitions between these states. Any s which is a final state in $\mathcal{A}_0$, is also a final state in $\mathcal{A}_{s_x}^{\ell \to t}$. Mark s_x as initial state.

Rule 4 $\mathcal{A}_{s_x}^{t \to \ell}$ consists of one final/initial state whose set of self-loops is equal to the intersections of self-loops of all final states of $\mathcal{A}_{s_x}^{v \to \ell}$.

Rule 5 If $\mathcal{A}_{s_x}^{\ell \to v}$ ($\mathcal{A}_{s_x}^{t \to \ell}$) consists of one state with no self-loops, then add an auto-loop to s_x in $\mathcal{A}_0$ to be used in rules 1 and 2 (rules 3 and 4) with arbitrary transitions so that node (v, s_x) is reachable from landmark ℓ (so that landmark ℓ is reachable from node (t, s_x)).

▶ **Proposition 3.** By using the regular languages, described by the automata constructed by applying rules 1 to 5, for the constrained landmark distance calculation for all pairs (v, s), the potential function $\pi(v, s)$ in Equation (1) is feasible.

An example of the application of (spe) can be found in Table 4b where rules 1 to 5 have been applied to the automaton in Figure 4a. Under weak conditions it can be proven that (spe) succeeds in providing better bounds in comparison to (bas) and (adv), for RegLCSP similar to the one discussed in the example.

Performance and memory consumption Finally note that the number of bounds to be calculated grows linearly to the number of relaxed arcs in (std), (bas), and (spe). For (adv), the number of calculated bounds in worst case scenario is an additional factor $|S|$ higher. Memory requirement for (bas) is equal to (std). It grows linearly in respect to $|S|$ and may be up to $|S|$ times higher in (adv). Memory requirement for (spe) may grow by a constant factor of 4 in the worst case with respect to (adv).

5 Experimental evaluation

We consider a multi-modal graph composed of the road and public transportation network of the French region Ile-de-France, which includes the city of Paris. It consists of five layers: private bike (b), rental bike (v), walking (f), car (c), and public transportation (p). Layers are connected by transfer arcs (t) which model the time needed to transfer from one transportation mode to another. The cost of transfer arcs is set uniformly to 20sec. Each arc has exactly one associated label $l \in \Sigma = \{b, v, f, c, p, t\}$. The graph consists of circa 3.7mil arcs and 1.2mil nodes. Dimensions of the single layers are summarized in Table 1. See [20, 19] for more information about graph models of a multi-modal network and time-dependency.

The private bike, walking, and rental bike layers are based on OpenStreetMap[1] data. Arc cost equals travel-time. Bikes have been considered to move at 12km/h, pedestrians at a speed of 4km/h. The private bike layer is connected to the walking layer at common street intersections. The bike rental layer is connected to the walking layer at the locations of bike rental stations[2]. In addition, 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, supermarket, etc.

Data for the public transportation layer has been provided by STIF[3]. It includes geographical and timetable data on buses, tramways, subways and regional trains. Our model is similar to the one presented in [20]: A *trip* of a public transportation vehicle is defined as a sequence of *route* nodes. Route nodes can be pictured as station platforms and are connected to *station* nodes, which model public transportation stations, such as those pictured on

[1] See www.openstreetmap.org
[2] Vélib', www.velib.paris.fr
[3] Syndicat des Transports d'Ile de France, www.stif.info, data for scientific use from 01/12/2010

subway network maps. Trips consisting of the same sequence of route nodes are grouped into *routes*. Travel times are modeled according to timetable information by time-dependent cost functions. They include waiting times at stations.

The car layer is based on geographical road data and traffic information provided by Mediamobile[4]. It is connected to the walking layer by transfer arcs at station nodes. Arc cost equals travel time which depends on the type of road. Circa 10% of the arcs have a time-dependent cost function to represent changing traffic conditions throughout the day.

SDALT is implemented in C++ and compiled with GCC 4.1. We merged and adapted the implementations of uniALT described in [16, 9] and D_{RegLC} described in [19]. As priority queue, we use a binary heap. As in the case of uniALT, periodical additions of landmarks (max. 6 landmark) and refresh cycles of the priority queue take place. We use an Intel Xeon, 2.6 Ghz, with 16 GB of RAM. 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 takes less than a minute. We compare SDALT employing the different methods (bas), (adv), and (spe), with D_{RegLC} and (sta). SDALT has been evaluated by running 500 test instances for five RegLCSP scenarios, see Figures 3a, 4a and 5. They have been chosen with the intention to represent real-world queries, which may often arise when looking for constrained shortest paths on a multi-modal transportation network. See Table 2 for experimental results. *Runtime* is the average running time of the algorithm over 500 test instances. *SettNo*, *touchNo* and *reInsNo* give the average of the number of settled, touched and reinserted nodes. *MaxSett* gives the maximum number of settled nodes. *TouchEd* and *calcPot* give the average number of touched edges and calculated potentials.

layer	arcs	nodes	time-dependent	PT-transfer	stations	transfer
Walking (f)	601 280	220 091	-	-	-	-
Private Bike (b)	600 952	220 091	-	-	-	440 182
Rental Bike (v)	600 952	220 091	-	-	1 198	2 396
Car (c)	1 112 511	514 331	111 641	-	-	37 906
Public Transportation (p)	259 623	109 922	82 833	176 790	21 527	37 944
Special Arcs (z)	10	-	-	-	-	-
Tot	3 731 700	1 284 526	194 474	(9 803 812 Time Points)		556 372

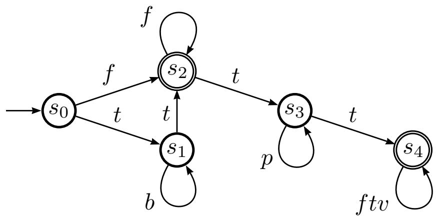
Table 1 Dimensions of the graph

5.1 Discussion of experimental results

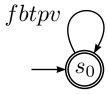
SDALT, in comparison to D_{RegLC}, succeeds in directing the constrained shortest path search faster toward the target in situations where L_0 is likely to introduce a detour from the unconstrained shortest path. This is the case when the use of fast transportation modes is excluded or limited, or if arcs with infrequent labels have to be obligatorily visited.

(bas) works well in situations where L_0 excludes a priori fast transportation modes. This can be observed in scenario C and in scenario D, where shortest paths are limited to the walking and rental bike layer, both being much slower than the car or public transportation layer, which are excluded. (adv) gives a supplementary speed-up in cases where initially allowed fast transportation modes are excluded from a later state on A_0 onward. This can be observed in scenario B, where by transition from the initial state s_0 toward s_1 or s_2, either

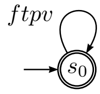
[4] www.v-trafic.fr, www.mediamobile.fr



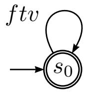
(a) $\mathcal{A}_0$ scenario A



(b) $\mathcal{A}_{\mathsf{bas}}$, $\mathcal{A}_{\mathsf{adv},s_0}$, $\mathcal{A}_{\mathsf{adv},s_1}$

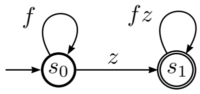


(c) $\mathcal{A}_{\mathsf{adv},s_2}$, $\mathcal{A}_{\mathsf{adv},s_3}$

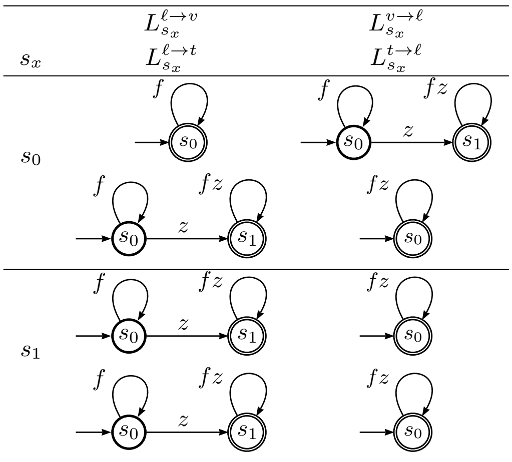


(d) $\mathcal{A}_{\mathsf{adv},s_4}$

Figure 3 Automata for scenario A. Shortest path must start either by walking (f) or by private bike (b). Once the private bike is discarded, the path can be continued by walking or by taking public transportation (p). The trip may then be continued by using bike rental (v) or by walking. Transfer arcs (t) are used to change between transportation modes. The automata in Figure 3b and Figures 3b, 3c, and 3d are used during the pre-processing phase for (bas) and (adv), respectively.

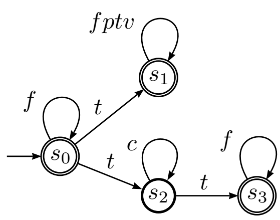


(a) $\mathcal{A}_0$ scenario D

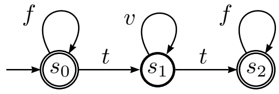


(b)

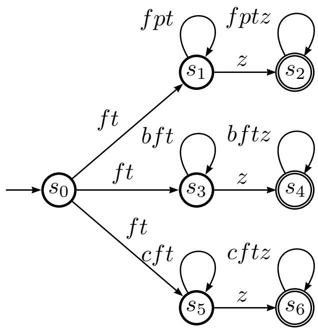
Figure 4 Automata for scenario D. Automata used for (spe) on the right. Landmark distance calculation of $d'(\ell, v, s_0)$ is constrained by language $L_{s_0}^{\ell\to v}$ described by the top-left automaton in row s_0, $d'(\ell, t, s_1)$ is constrained by $L_{s_1}^{\ell\to t}$ described by the bottom-left automaton in row s_1, etc.



(a) $\mathcal{A}_0$ scenario B



(b) $\mathcal{A}_0$ scenario C



(c) $\mathcal{A}_0$ scenario E

Figure 5 Automata of scenarios

scenario	algo	space [MB]	runtime [ms]	settNo	maxSett	touchNo	touchEd	calcPot
scenario A	D_{RegLC}	0	529	542 914	1 397 414	547 643	1 998 610	-
	sta	310	486	376 527	1 376 485	381 081	1 405 720	1 750 580
	bas	310	427	333 121	1 350 973	337 528	1 244 450	1 591 770
	adv	930	361	139 635	688 616	183 104	516 746	2 133 710
	spe	1 660	262	162 982	861 574	224 389	617 503	598 707
scenario B	D_{RegLC}	0	509	446 279	1 576 407	453 835	1 303 320	-
	std	310	243	176 971	1 387 476	182 469	511 462	861 436
	bas	310	138	117 549	894 842	121 489	337 650	510 982
	adv	1 240	114	66 100	409 027	71 174	149 285	619 147
	spe	1 550	198	165 105	612 247	177 596	387 361	368 139
scenario C	D_{RegLC}	0	355	456 674	865 722	457 957	1 649 190	-
	std	310	431	406 837	865 395	408 279	1 491 630	1 608 090
	bas	310	17	20 252	220 571	22 217	76 099	68 880
	adv	620	18	14 536	159 146	18 020	51 665	93 915
	spe	1 240	16	13 195	210 208	17 510	48 228	69 609
scenario D	D_{RegLC}	0	160	235 943	417 117	236 854	944 596	-
	std	310	209	224 408	415 861	225 384	899 874	940 876
	bas	310	38	45 151	223 726	46 192	185 620	147 207
	spe	930	8	8 389	59 073	9 181	35 210	32 180
scenario E	D_{RegLC}	0	1 995	1 430 230	4 231 958	1 447 310	4 570 860	-
	sta	310	1 174	723 364	3 169 152	737 249	2 342 480	3 578 470
	adv	1 240	902	487 880	1 600 241	497 815	1 564 900	4 593 790
	spe	2 480	511	395 947	1 565 563	406 472	1 256 090	960 304

Table 2 Experimental results

the public transportation network or the car network is excluded. However, speed-ups are mild as the number of potentials which have to be calculated for (adv) is much higher as it is for (bas). Finally, (spe) has a positive impact on running times for scenarios where the visit of some infrequent labels, which would generally not be part of the unconstrained shortest path, is imposed by L_0, see scenario D and scenario E.

Speed-ups for scenarios including labels of arcs with time-dependent arcs costs (public transportation, car) are lower then speed-ups for scenarios considering only arcs with time-independent arcs costs. This is due to the fact that bounds are calculated by using the minimum weight cost function. Bounds are especially bad for public transportation at night time, as connections are not served as frequently as during the day.

6 Conclusions

We presented a method on how to apply the speed-up technique uniALT to the generalized Dijkstra's algorithm (D_{RegLC}) which is used to solve the RegLCSP. SDALT uses preprocessed data to anticipate the impact of the given regular language on the shortest path. We proposed four different methods on how to produce this preprocessed data and explained in which situations they are likely to work best. We implemented our algorithm and produced different versions which differ only slightly in terms of coding but differ in terms of memory requirements and performance. We ran experiments on a real-world public transportation network. The results showed that SDALT succeeds in providing speed-ups of a factor 2 to 20 in respect to D_{RegLC}. Among the possible improvements, we believe that there is space to reduce memory consumption. A logical direction for future research would be the investigation of the impact of a bi-directional search on SDALT and the applicability and effects of contraction. Another question is how to adapt SDALT efficiently to multi-objective versions of D_{RegLC}. It would also be interesting to test its performance on dynamic networks.

References

1 Chris Barrett, Keith Bisset, Martin Holzer, Goran Konjevod, Madhav Marathe, and Dorothea Wagner. Engineering label-constrained shortest-path algorithms. *Algorithmic Aspects in Information and Management*, pages 27–37, 2008.

2 Chris Barrett, Keith Bisset, Riko Jacob, Goran 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 Chris Barrett, Riko Jacob, and Madhav Marathe. Formal-Language-Constrained Path Problems. *SIAM Journal on Computing*, 30(3):809, 2000.

4 Daniel Delling and Giacomo Nannicini. Bidirectional core-based routing in dynamic time-dependent road networks. *Algorithms and Computation*, 0(2):812–823, 2008.

5 Daniel Delling, Thomas Pajor, and Dorothea Wagner. Accelerating Multi-Modal Route Planning by Access-Nodes. *Algorithms-ESA 2009*, 2, 2009.

6 Daniel Delling, Peter Sanders, Dominik Schultes, and Dorothea Wagner. Engineering route planning algorithms. *Algorithmics of large and complex networks*, 2:117–139, 2009.

7 Daniel Delling and Dorothea Wagner. Time-Dependent Route Planning. *Online*, 2, 2009.

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

9 Andrew V Goldberg and C. Harrelson. Computing the shortest path: A search meets graph theory. In *Proceedings of the 16th annual ACM-SIAM Sym. on Discrete algorithms*, 2005.

10 Andrew V Goldberg and R Werneck. Computing point-to-point shortest paths from external memory. *US Patent App. 11/115,558*, 2005.

11 Peter Hart, Nils Nilsson, and Bertram Raphael. A Formal Basis for the Heuristic Determination of Min. Cost Paths. *IEEE Trans. on Sys. Science and Cybern.*, 4(2), 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 David Kaufman and Robert Smith. Fastest paths in time-dependent networks for intelligent vehicle-highway systems applications. *Journal of Intelligent Transportation Systems*, 1(1):1–11, 1993.

14 Angelica Lozano and Giovanni Storchi. Shortest viable path algorithm in multimodal networks. *Transportation Research Part A: Policy and Practice*, 35(3):225–241, 1999.

15 Alberto O. Mendelzon and Peter T. Wood. Finding Regular Simple Paths in Graph Databases. *SIAM Journal on Computing*, 24(6):1235, 1995.

16 Giacomo Nannicini, Daniel Delling, Leo Liberti, and Dominik Schultes. Bidirectional A* search for time-dependent fast paths. *Experimental Algorithms*, 2(2):334–346, 2008.

17 Giacomo Nannicini and Leo Liberti. Shortest paths on dynamic graphs. *International Transactions in Operational Research*, 15(5):551–563, 2008.

18 Ariel Orda and Raphael Rom. Shortest-path and minimum-delay algorithms in networks with time-dependent edge-length. *Journal of the ACM*, 37(3):607–625, 1990.

19 Thomas Pajor. *Multi-Modal Route Planning.* Master thesis, Univ. Karlsruhe (TH), 2009.

20 Evangelia Pyrga, Frank Schulz, Dorothea Wagner, and Christos Zaroliagis. Efficient models for timetable information in public transportation systems. *Journal of Experimental Algorithmics*, 12(2):1–39, 2007.

21 J.F. Romeuf. Shortest path under rational constraint. *Information processing letters*, 28(5):245–248, 1988.

22 Hanif Sherali, Antoine Hobeika, and Sasikul Kangwalklai. Time-Dependent, Label-Constrained Shortest Path Problems with Applications. *Transp. Science*, 37(3), 2003.

23 Mihalis Yannakakis. *Graph-theoretic methods in database theory.* ACM Press, 1990.