Efficient routing on multi-modal transportation networks

Thèse présentée pour obtenir le grade de

DOCTEUR DE L'ECOLE POLYTECHNIQUE

par

Dominik Kirchler

Soutenue le 3 octobre 2013 devant le jury composé de :

Leo Liberti	Ecole Polytechnique, Palaiseau	Directeur de thèse
Roberto Wolfler Calvo	Université Paris 13, Villetaneuse	Co-directeur de thèse
Dorothea Wagner	Karlsruhe Institute of Technology	Rapporteur
Emmanuel Neron	Ecole Polytechnique de l'Université de Tours	Rapporteur
Dominique Feillet	Ecole des Mines de Saint-Etienne	Rapporteur
Philippe Goudal	Mediamobile, Ivry-sur-Seine	Membre du jury
Olivier Bournez	Ecole Polytechnique, Palaiseau	Président du jury

Abstract

Mobility is an important aspect of modern society. Consequently, there is a growing demand for services offering efficient route planning. In this thesis, we study multi-modal routing and the Dial-a-Ride system. Both respond to the need of a more efficient utilization of the available transportation infrastructure, which is an important component of sustainable development.

Multi-modal route planning is complex because of the various modes of transportation which have to be combined. A generalization of Dijkstra's algorithm may be used to find shortest paths on multi-modal networks. However, its performance is not sufficient for real world applications. For this reason, this thesis introduces a new algorithm called `SDALT`. It is an adaption of the speed-up technique `ALT`. To evaluate the performance of `SDALT`, we produced a graph of a real-world multi-modal network based on transportation data of the French region Ile-de-France. It includes walking, public transportation, car, and bicycle, as well as timetable information and traffic data. Experiments show that `SDALT` performs well, with speed-ups of a factor 1.5 to 60 with respect to the basic algorithm.

Other than finding shortest multi-modal paths that optimally combine the use of several modes of transportation, yet another problem arises: finding an optimal multi-modal return path or 2-way path. When using a private vehicle for parts of the outgoing path, it has to be picked up during the incoming path so that it can be taken to the starting location. For this reason, the parking must be chosen in such a way as to optimize the combined travel times of the outgoing and incoming path. We propose an efficient algorithm that solves this problem faster than previous techniques.

The Dial-a-Ride system offers passengers the comfort and flexibility of private cars and taxis at a lower cost and higher eco-efficiency by combining similar transportation demands. It works as follows: passengers request the service by calling a central unit. They specify their pick-up point, their delivery point, the number of passengers, and some limitations on their service time (e.g., the earliest departure time). An algorithm then calculates the routes and schedules of the vehicles. We propose a new efficient and fast heuristic, a Granular Tabu Search, to produce good solutions in a short amount of time (up to 3 minutes). Our algorithm produces better results for more than half of the test instances after 60 seconds of optimization time in comparison with other methods.

Résumé

La mobilité est un aspect important des sociétés modernes. Par conséquent, il y a une demande croissante pour des solutions informatiques de calcul d'itinéraire. Cette thèse analyse le routage multimodal et le système Dial-a-Ride. Ils contribuent à une utilisation plus efficace de l'infrastructure de transport disponible, élément déterminant dans la perspective d'un développement durable.

La planification d'itinéraires multimodaux est rendus complexe en raison des différents modes de transport qui doivent être combinés. Une généralisation de l'algorithme de Dijkstra peut être utilisée pour trouver les chemins les plus courts sur un réseau multimodal. Cependant, sa performance n'est pas suffisante pour les applications industrielles. De ce fait, cette thèse introduit un nouvel algorithme appelé `SDALT`. Il s'agit d'une adaptation de la technique d'accélération `ALT`. Pour évaluer la performance de `SDALT`, un graphe a été construit à partir d'un réseau multimodal réel basé sur les données de transport de la région française Ile-de-France. Ce graph inclut la marche, les transports en commun, la voiture, la bicyclette ainsi que des informations relatives aux horaires et aux conditions de circulation. Les tests de performance montrent que `SDALT` fonctionne bien, avec un temps de calcul réduit d'un facteur compris entre 1.5 et 60 par rapport à l'algorithme de base.

Dans un contexte multimodal autre la question de la détermination du chemin le plus court, se pose celle de trouver un chemin aller-retour multimodal optimal entre un point de départ et un point d'arrivée. Un véhicule privé (voiture ou bicyclette) utilisé pour une première partie du trajet aller doit être récupéré au cours du trajet retour pour être ramené au point de départ. Pour cette raison, le parking doit être choisi de manière à optimiser les temps de déplacement du trajet aller et du trajet retour combinés. L'algorithme qui est proposé dans cette thèse résout ce problème plus rapidement que les techniques actuelles.

Le système Dial-a-Ride offre aux passagers le confort et la flexibilité des voitures privées et des taxis à un moindre coût et avec plus d'éco-efficacité car il regroupe les demandes de transport similaires. Il fonctionne de la manière suivante : les passagers demandent le service en appelant un opérateur et communiquent leur point de départ, leur point de destination, le nombre de passagers, ainsi que quelques précisions sur les horaires de service. Un algorithme calcule ensuite les itinéraires et les horaires des véhicules. Cette thèse propose une nouvelle heuristique efficace et rapide de type Granular Tabu Search, capable de produire de bonnes solutions dans des délais courts (jusqu'à 3 minutes). Comparativement aux autres méthodes, et au regard des instances de test de la littérature, cet algorithme donne de bons résultats.

Acknowledgements

Trois ans entre les langues.

Innanzitutto grazie a Leo Liberti e al (dai, diciamolo) amico-professore Roberto Wolfler Calvo.

Vielen Dank an Dorothea Wagner und an ihr Forschungsgruppe.

Merci à Philippe Goudal et à mes collègues chez Mediamobile.

But, of course, the biggest thanks go to Sheena.

Contents

Chapter 1

Introduction

1.1 Introduction

Mobility is an important aspect of modern society. Consequently, there is a growing demand for services offering route planning. Several websites provide such services with easy-to-use interfaces. A user can select a starting and destination location, often directly on a map, and can specify some characteristics, such as departure or arrival time, the preferred mode of transportation, and whether he wants to pass by some other location. The service will then propose a routing plan. Most services are domain specific and offer route planning only on specific networks and consider only a limited set of transportation modes. Many websites can calculate paths for walking, cars (often including traffic information), or bicycles on the road network but public transportation is seldom included, or is included in a limited form. Route planning on public transportation networks is offered on websites of transportation agencies which in turn do not incorporate route planning on roads.

In response to a growing demand for integrated solutions and the need for a more efficient utilization of the available transportation infrastructure, which is an important component of sustainable development, in recent years prototypes of route planning services which attempt to consider and to combine all available modes of transportation have been developed. Their goal is to provide *multi-modal route planning* (see Figures 1.1 and 1.2). Modes of transportation on a multi-modal network include private and rental bicycle, private and rental car, walking, and public transportation.

Users are typically interested in the fastest path to reach their destination, i.e., the shortest path in terms of travel time. On a public transportation network, travel time depends both on the timetable of the transportation vehicles and on traffic perturbations which may cause delays and service interruptions. Travel time on roads depends on their congestion level. Knowledge of real-time and forecast traffic information is required to compute the traversal time of roads for each time instant in the future. For this purpose, statistical models have been developed which are able to predict the evolution of traffic to a certain degree of accuracy. They are based on traffic data provided by traffic sensors which constantly monitor the traffic at strategic locations on the road network. Using a large database of historical traffic information and by applying statistical analysis, speed profiles can be computed for road segments. They provide a forecast of the most probable travel speed on the road segment or travel time to pass the road segment for each time instant of a

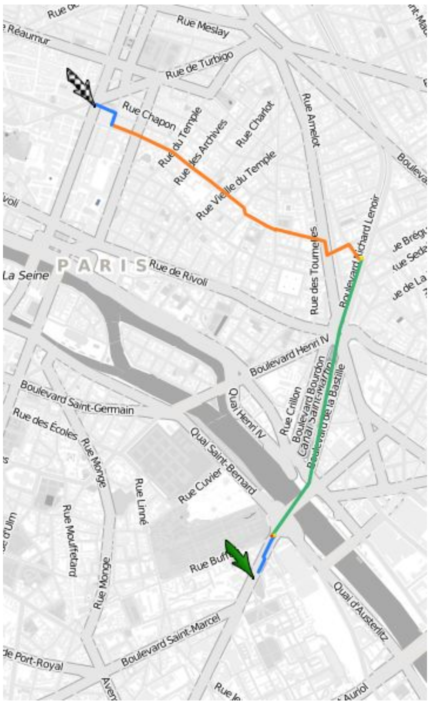


Figure 1.1: Example of a multi-modal path. The path consists of walking (blue line), public transportation (green line), and rental bicycle (orange line). (Map: OpenStreetMap)

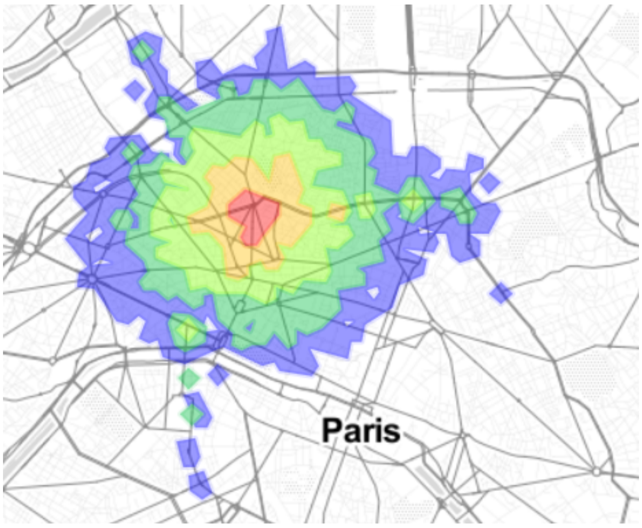


Figure 1.2: Isochrones: destinations reachable by using walking, public transportation, and rental bicycle after 3min (red), 6min (orange), 9min (light green), 12min (green), and 15min (violet) of travel time. Note that isocrone areas can be disconnected because stations of public transportation can often be reached faster then locations between stations. (Map: OpenStreetMap)

day. Not all roads are monitored, thus typically only a part of the road network is provided with real time and forecast traffic information, while the remaining part is associated with *static* travel times. Note that temporary closures of road segments due to construction works or accidents also strongly influence travel times and therefore must also be taken into account.

When travel time over a road segment depends on the time instant at which the segment is traversed, then the road network has the characteristic that it is *time-dependent*. In the same sense, public transportation networks are time-dependent. Travel times from one train station to another depend on the arrival time of the user at the departure station. If the user arrives prior to the departure of the train, he has to wait; the total travel time of the user is equal to the sum of the waiting time and the travel time of the train.

Travel time when walking or biking is not time-dependent, but it may vary slightly from user to user and it might not always be proportional to the length of the road (e.g., because of steepness). In recent years, major cities have adopted bicycle sharing systems, e.g., Paris (France), London (UK), Washington DC (US), Hangzhou (China), New York (US). Besides providing affordable access to bicycles for short-distance trips in an urban area as an alternative to motorized vehicles, these systems are also supposed to cover the *Last Kilometer*, i.e., the distance between public transportation stops and the departure point or final destination. Thus they constitute an important component of a multi-modal transportation network.

Another important component of multi-modal paths are recently introduced flexible rental car systems, e.g., the Autolib' system of the city of Paris. They are specifically designed for urban or short interurban journeys. Traditional car rental services mainly offer rental cars for full days, weekends, or longer. These new flexible systems offer rental cars to be picked up and returned at strategically located car rental stations equipped with automated identification and payment systems.

Besides the minimization of travel time, multi-modal paths have to respect additional constraints such as restrictions and/or preferences of the users in employing certain modes of transportation. Users may be willing to take trains, but not buses, or may want to exclude bicycles when it is raining. Also, the minimization of the number of transfers between modes of transportation is sometimes important. Furthermore, whereas distances can be covered by walking at almost any point during an itinerary, some modes of transportation such as private cars and bicycles, once discarded, are not available again at a later point in the itinerary. Rental cars or bicycles can be accessed at rental stations and have to be returned to rental stations before reaching the destination.

Another important characteristic of a route planning service is that the user may want to pass by an intermediate stop before reaching the destination. However, a user might not specify the precise geographical location of this stop and just wish to pass by any pharmacy, post-office, supermarket, etc. In this case, the route planning service has to autonomously determine the exact location of the intermediate stop and adapt the multi-modal path between start and destination location accordingly.

Yet another problem arises when traveling along a multi-modal path. Sometimes it is advisable to depart from the starting location by using a private car or bicycle and then to transfer at a later point to public transportation or to walk in order to reach the destination. Consequently, on some intermediate location the user has to park his car or bicycle which he will want to pick up on the incoming path in order to take it home. It is clear that the travel time of the incoming path will depend on the location where the user parked his car on the outgoing path and may not be optimal and be heavily influenced by this, as traffic conditions or timetables are not the same at different times of the day. The parking location must thus be chosen wisely in order to optimize the combined travel times of the outgoing

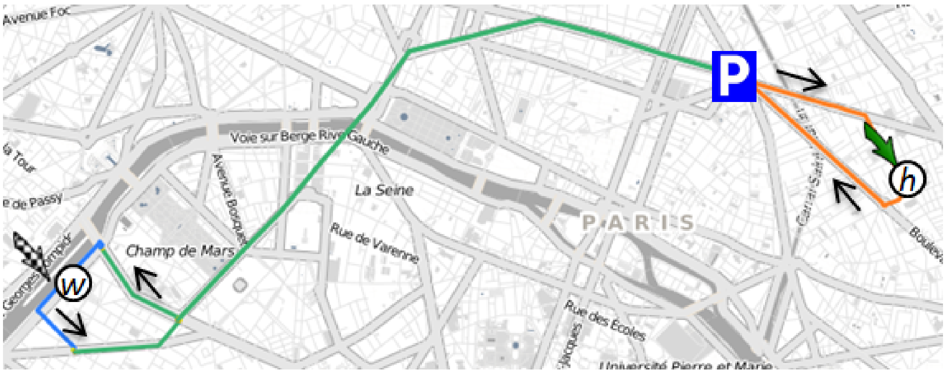


Figure 1.3: Example of a return path or 2-way path. The path consists of walking (blue line), public transportation (green line), and private bicycle (orange line). The outgoing path starts at h (home) by bicycle. The bicycle is discarded at the parking location p. The destination w (work) is reached by public transportation and walking. The incoming path passes by the parking place so that the bicycle can be picked up. (Map: OpenStreetMap)

and incoming paths, which form the multi-modal return path or *2-way path*. See Figure 1.3 for an example.

Under reasonable assumptions, the problem of finding multi-modal paths as discussed above is theoretically solved in polynomial time by a generalization of Dijkstra's algorithm. However, an application of this algorithm over medium-sized multi-modal networks may require several seconds of calculation time. For real-world applications, this is too slow. For this reason, techniques to speed up the algorithm are required.

In addition to combining existing modes of transportation in a better way, it is also important to study innovative transportation services to provide better mobility to passengers. The Dial-a-Ride (DAR) system is such a service. It offers passengers the comfort and flexibility of private cars and taxis at a lower cost and higher eco-efficiency by combining similar transportation demands. It is thus in line with the demands of the sustainable development requirement. Dial-a-Ride systems are already employed in several cities. It works as follows: passengers request the service by calling a central unit. They specify their pick-up point, their delivery point, the number of passengers, and some limitations on the service time (e.g., the earliest departure time). An algorithm then calculates the routes and schedules of the vehicles depending on the received requests, thus permitting a global optimization of the transportation system. The routing problem cannot be solved by Dijkstra's algorithm and other techniques have to be applied.

1.2 Contribution

The major contributions of this thesis are:

Speed-up technique. A generalization of Dijkstra's algorithms may be used to solve routing problems on a multi-modal network. However, its performance may not be sufficient for real world applications. For this reason, this thesis introduces a new speed-up technique called `SDALT`. It is an adaption of the speed-up technique `ALT`. The experiments show that `SDALT` performs well, with speed-ups of a factor 1.5 to 40 (up to a factor of 60 with approximation), with respect to the basic algorithm, in networks where some modes of transportation tend to be faster than others.

2-Way Multi-Modal Shortest Path Problem. We will introduce the 2-way multi-modal shortest path problem (`2WMMSP`). When using a private vehicle for parts of the outgoing path, it has to be picked up during the incoming path so that it can be taken to the starting location. For this reason, the parking must be chosen in order to optimize the combined travel times of the outgoing and incoming path. We will propose an algorithm which solves this problem and various ameliorations to reduce runtime including the application of `SDALT`.

Granular Tabu Search to solve the Dial-A-Ride problem. We will address the Dial-A-Ride problem. The objective is to maximize the number of passengers served and the quality of service, as well as to minimize overall system cost. The main contribution here is the development of an efficient and fast heuristic to produce good solutions in a short amount of time (up to 3 minutes). We propose a new Granular Tabu Search which uses information provided by the solution of a simple and useful sub-problem to guide the local search process. This sub-problem provides distance information and clusters of close requests. The idea is that passengers who are close both spatially (in terms of the distance between pick-up and delivery points) and temporally (with respect to time windows) are probably best served by the same vehicle in order to produce good solutions.

Network modeling. A minor but original contribution of this thesis is the network modeling of a complete transportation network of an inter-urban region, including all major modes of transportation, i.e., rental and private cars and bicycles, walking and public transportation. It includes all relevant information about the modes of transportation, such as traffic conditions for cars and timetable information for public transportation vehicles. To our knowledge, this is the first work to consider a multi-modal network in this configuration and on this scale.

1.3 Overview

We start by giving some basic notations which are used throughout this work (Chapter 2). We introduce the concept of formal languages and more specifically regular languages. Regular languages will be used to model the constraints on multi-modal paths, such as maximal number of transfers, viability, choice of modes of transportation. Furthermore, basic notations of graph theory are given.

In Chapter 3, we discuss how to model different modes of transportation and how to combine the models to produce a graph of a multi-modal transportation network. We show a practical application: our graphs include foot, bicycle, car, public transportation, as well as rental car and rental bicycle. They are based on transportation data of the French region Ile-de-France and New York City.

The shortest path problem is introduced in Chapter 4. We will discuss Dijkstra's algorithm and uni-modal routing on road networks as well as public transportation networks.

We will also introduce the regular language constrained shortest path problem (`RegLCSP`) which can be applied for routing problems on multi-modal transportation networks.

In Chapter 5, we will discuss a new speed-up technique for a generalization of Dijkstra's algorithm which solves `RegLCSP`. It allows to calculate multi-modal paths faster than previous methods. We will first introduce the general concepts of the algorithm and then discuss several different versions of it, including uni-directional and bi-directional search.

Chapter 6 presents an efficient algorithm to solve the 2-way multi-modal shortest path problem (`2WMMSP`). Its goal is to find an optimal return path on a multi-modal network. The shortest incoming path is often not equal to the shortest outgoing path as traffic conditions and timetables of public transportation vary throughout the day. The main difficulty lies in finding an optimal parking location of a private bicycle or private car, since they may be used for parts of the outgoing path and will need to be picked up during the incoming path. We present extensive experimental results. Our algorithm outperforms a previous algorithm.

In Chapter 7, we discuss a new Granular Tabu Search algorithm for the static Dial-a-Ride Problem with the objective of producing good solutions in a short amount of time (up to 3 minutes). We evaluate the algorithm on test instances from the literature. For most instances, our results are close to the results of another approach and we report new best solutions for some instances.

Chapter 8 concludes this thesis.

Bonne lecture!

Chapter 2

Definitions and Notations

In this chapter, important notations from formal language theory and graph theory, which are used throughout this work, will be introduced. We will also give a formal definition of the time-dependent shortest path problem.

2.1 Languages and Automata

We will use regular expressions and automata to describe the different constraints which may arise when solving multi-modal routing problems. In this section, we first introduce formal languages and regular languages. Then we will describe more in detail regular expressions and automata. A rigorous exposition of formal languages and the theory of computation can be found in the books [71, 123].

Formal Languages. Formal language theory concerns the study of various types of formalisms to describe languages. Different to a natural language, a formal language is an abstract language with the primary focus not being communication but its mathematical use. Formal languages are particularly useful for a precise mathematical description of chains of symbols. Examples are programming languages. Formal languages are primarily used in the fields of linguistics, logics, and theoretical computer science. Important characteristics are their expressive power, their recognizability and their comparability. *Regular languages* are a type of formal languages which provide a good compromise between expressivity and ease of recognizability, and are widely used in practical applications.

An *alphabet* Σ is a set of letters or symbols. A *word* over an alphabet can be any finite sequence $w = [\sigma_1, \sigma_2, ..., \sigma_k]$ of letters. For simplicity, we write $w = \sigma_1\sigma_2 \ldots \sigma_k$. The set of all words over an alphabet Σ is usually denoted by Σ^* (using the *Kleene star*, see below). The *length* of a word is the number of symbols it is composed of. For any alphabet there is only one word of length 0, the empty word, which is denoted by ϵ. By concatenation, two words $w_1 = \sigma_1 \ldots \sigma_k$ and $w_2 = \sigma_{k+1} \ldots \sigma_l$ can be combined to form a new word $w = w_1w_2 = \sigma_1 \ldots \sigma_k\sigma_{k+1} \ldots \sigma_l$, whose length is the sum of the lengths of the original words. Concatenating a word with the empty word gives the original word.

A *formal language* L over an alphabet is a subset of Σ^*, i.e., a set of *words* over that alphabet. Note that this set is not necessarily finite. All operations on sets like union, intersection, and complement also apply to languages. Suppose L_1 and L_2 to be languages over some common alphabet, then the *concatenation* $L_3 = L_1 \circ L_2$ is the language consisting of all words of form vw where v is a word of L_1 and w is a word of L_2, $L_3 = \{v \circ w | v \in L_1 \wedge w \in L_2\}$. E.g., if $L_1 = \{a, b\}$ and $L_1 = \{c, d\}$ then $L_1 \circ L_2 = L_3 = \{ac, ad, bc, bd\}$.

Regular Languages. In this work, we will use regular languages as their expressive power suffices for our purpose and they do not influence much the complexity of the routing algorithms we are using.

Definition 2.1.1 (Regular Languages)**.** *The collection of regular languages over an alphabet Σ is defined as follows.*

- *The empty language $\emptyset$ is a regular language.*
- *For each $\sigma \in \Sigma$, the singleton language $\{\sigma\}$ is a regular language.*
- *If L_1 and L_2 are regular languages, then $L_1 \cup L_2$ (union), $L_1 \circ L_2$ (concatenation), and L_1^* (Kleene star) are regular languages.*
- *No other languages over Σ are regular.*

Any regular language can be described by a *regular expression* and a *non-deterministic finite automaton* or *NFA*. We will use both concepts to represent regular languages.

Non-deterministic finite automaton (NFA). A non-deterministic finite automaton (NFA) is a finite state machine where from each state and a given input symbol the automaton may change into several possible next states. This distinguishes it from the deterministic finite automaton (DFA), where the next possible state is uniquely determined. However, a NFA can always be translated to an equivalent DFA, which recognizes the same formal language. Both types of automata recognize only regular languages. NFAs were introduced in 1959 by the authors of [108], who also showed their equivalence to DFAs.

A NFA is a 5-tuple, $\mathcal{A} = (S, \Sigma, \delta, I, F)$, consisting of a finite set of states S, an alphabet Σ, a transition function $\delta : \Sigma \times S \to 2^S$, a set of initial states $I \subseteq S$, and a set of final states $F \subseteq S$. State diagrams (or transition graphs) are used to illustrate automata: states $s \in S$ are presented as nodes and for each state s we draw an arc from s to s' labeled by σ if and only if $s' \in \delta(s, \sigma)$. Initial states are marked by an incoming arc-tip whereas final states are double framed. The size of a NFA is defined as $|\mathcal{A}| = |S||\Sigma|$.

Let $L \subseteq \Sigma^*$ be an arbitrary language. A word $w \in L$ is *accepted* by $\mathcal{A}$, if there is a path in the state diagram starting at an initial state $s \in I$, leading to a final state $s \in F$, and where the subsequent arcs on the path are labeled by the subsequent symbols of w. If no such path exists, then the word is *rejected*. If every word $w \in L$ is accepted by $\mathcal{A}$, then the language L is accepted by $\mathcal{A}$.

The backward automaton $\texttt{back}(\mathcal{A})$ of $\mathcal{A} = (S, \Sigma, \delta, I, F)$ is the automaton which has the same set of states S, the same alphabet Σ, but with its transition function reversed, i.e., all arcs of the automaton are reversed. The set of the final and initial states of the backward automaton $\texttt{back}(\mathcal{A})$ is equal to the initial and final states of the original automaton $\mathcal{A}$, respectively.

Figure 2.1 shows a simple example of a NFA. We define $\overleftrightarrow{S}(s, \mathcal{A})$ and $\overleftrightarrow{\Sigma}(s, \mathcal{A})$ as the functions which return the sets of all states and labels, respectively, reachable by multiple

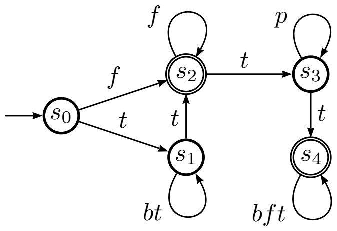


Figure 2.1: A simple non-deterministic finite automaton (NFA) given by its state diagram. It is non-deterministic because from state s_1 and input t two states, s_1 and s_2, can be reached. The NFA consists of five states, one initial state (s_0) and two final states (s_1,s_4). The words f, $tbbt$, and $ffftppt$ are accepted by the automaton whereas tb, bb, and ftp are not accepted.

transitions on an automaton $\mathcal{A}$ by starting at state s, backward and forward. E.g., in Figure 2.1, $\overrightarrow{S}(s_2, \mathcal{A}) = \{s_2, s_3, s_4\}$, $\overleftarrow{\Sigma}(s_2, \mathcal{A}) = \{b, f, t\}$.

A simple example of a language that is not regular is the set of words $\{a^n b^n | n \geq 0\}$. It consists of all words consisting of a number of a's followed by the same number of b's. This language cannot be recognized by a NFA. Intuitively, a NFA has finite memory and cannot remember or count the exact number of a's. If n is bound, i.e., $n < l$, then the language is recognizable by a NFA. The size of the set of states S of the NFA is proportional to l. In general, however, if n has no such bound, there is no fixed size automaton that can recognize this language.

Regular expressions. Given a finite alphabet Σ, the empty set $\emptyset$, the empty string ϵ, and the literal character $\sigma \in \Sigma$ are regular expressions. Given regular expressions R_1 and R_2, the following operations are defined to produce regular expressions:

- (concatenation) $R_1 \circ R_2$ denotes the set $\{\alpha\beta | \alpha \in R_1 \wedge \beta \in R_2\}$. For example, $\{ab, c\} \circ \{d, ef\} = \{abd, abef, cd, cef\}$.
- (alternation) $R_1 | R_2$ denoting the set union of sets described by R_1 and R_2. For example, $\{ab, c\} | \{ab, d, ef\} = \{ab, c, d, ef\}$.
- (Kleene star) R^* is the set of all words that can be made by concatenating any finite number (including zero) of strings from the set described by R. For example, $\{0, 1\}^*$ is the set of all finite binary strings (including the empty string), and $\{ab, c\}^* = \{\epsilon, ab, c, abab, abc, cab, cc, ababab, abcab, \ldots\}$.

The Kleene star has the highest priority, followed by concatenation and then alternation. Kleene's Theorem [83, 108] states that each regular language R can be described by a NFA $\mathcal{A}$, i.e., for every word $w \in \Sigma^*$ it holds that $\mathcal{A}$ accepts w if and only if $w \in R$. On the other hand, for every finite automaton $\mathcal{A}$ the set of words accepted by $\mathcal{A}$ has the property of being a regular language. E.g., the regular expression of the automaton represented in Figure 2.1 is $(f|(t(t|b)^*t)f^*)|(f|(t(t|b)^*t)f^*t(p^*)t(b|f|t)^*)$.

2.2 Graph Theory

We model transportation networks using graphs. In this section, we introduce basic concepts of graph theory.

Graph. A graph $G = (V, A)$ consists of a finite set of nodes $v \in V$, and a set of arcs $(i, j) \in A$, $i, j \in V$. We only use directed graphs so an arc (i, j) is considered to be directed from i to j; i is called the head and j is called the tail of the arc. An arc (i, j) where $i = j$ and so connects a node to itself, is called a *self-loop*. We define the union between two graphs $G_k = (V_k, A_k)$ and $G_l = (V_l, A_l)$ as $G_k \cup G_l = (V_k \cup V_l, A_k \cup A_l)$. In our scenario nodes typically represent road intersection or public transportation stations and arcs the connections or roads between intersections and stations.

Arc cost. Every arc has an associated arc cost which in our case represents travel times. Whereas for non time-dependent route planning it is sufficient to have constant costs (c_{ij} gives the cost for arc (i, j)), we generalize this concept and use periodic cost functions to be able to assign to an arc different costs for different times of the day. We call a graph which uses such cost functions a time-dependent graph.

All cost functions are elements of a function space $\mathbb{F}$ of positive functions $f : \mathbb{R}^+ \to \mathbb{R}^+$. The cost functions associated with an arc $(i, j) \in A$ is given by $c : A \to \mathbb{F}$ and are denoted by $c_{ij}(\tau)$. In this work, we only use *periodic* cost functions with period P. We have that $\forall \tau \geq P$, $c_{ij}(\tau) = c_{ij}(\tau - kP)$, where $k = \max\{k \in \mathbb{N} | \tau - kP \in \mathcal{T}\}$ and $\mathcal{T} = [0, P] \subset \mathbb{R}$. This implies $c_{ij}(\tau + P) = c_{ij}(\tau)$, $\forall \tau \in \mathcal{T}$. Cost functions are interpreted as travel time. We additionally require that

$$c(x) + x \leq c(y) + y, \forall c \in \mathbb{F}, x, y \in \mathbb{R}^+, x \leq y;$$

this ensures the FIFO property (see below). The lower and upper bound of $c_{ij}(\tau)$ is defined as $\underline{c}_{ij} = \min_{\tau \in \mathcal{T}} c_{ij}(\tau)$ and $\bar{c}_{ij} = \max_{\tau \in \mathcal{T}} c_{ij}(\tau)$, respectively. A graph which uses the lower or upper bound of $c_{ij}(\tau)$ as cost function is marked by $\underline{G}$ and $\bar{G}$, respectively

Choice of the cost function. We will use the graph to search for shortest paths. The efficiency of algorithms for shortest paths computations on time-dependent graphs depend on the choice of the cost function. See [32] for a study of the use of different cost functions. In this work, we will use *piecewise linear functions* to model cost functions. They can easily approximate traffic data, represent time tabled data, and they allow for some flexibility (data accuracy versus memory requirements) while being simple to treat computationally. Furthermore, the FIFO property can be checked efficiently: the condition $f(x) + x \leq f(y) + y, \forall x \leq y$ can be written as $\frac{df(x)}{dx} \geq -1$.

A piecewise-defined function is a function which is defined by multiple subfunctions, each subfunction applying to a certain interval of the main function's domain. A piecewise linear function f is a function composed of straight-line sections. It is a piecewise-defined function whose pieces or subfunctions are affine functions. It can be be described by a finite set $\mathcal{B}$ of interpolation points where each interpolation point $p_i \in \mathcal{B}$ consists of a departure time τ_i and an associated function value $f(\tau_i)$. The value of f for an arbitrary time τ is computed by interpolation. Note that this is done differently for time-dependent road networks and public transportation networks (see Section 3.1).

Labeled graph. A *labeled* graph is a triplet $G = (V, A, \Sigma)$. Other then nodes and arcs, it includes a set of labels $l \in \Sigma$. In our case the labels are used to mark arcs as, e.g., foot paths (label f), bicycle lanes (label b), highways (label c), etc. An arc in a labeled graph is a triple $(i, j, l) \in A \subseteq V \times V \times \Sigma$. It has constant cost c_{ijl} or time-dependent cost $c_{ijl}(\tau)$.

Path. A path p in G is a sequence of nodes $p = (v_1, \ldots, v_k)$ such that $(v_i, v_{i+1}) \in A$ for all $1 \leq i < k$. The cost of the path in a non time-dependent 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

$$\gamma((v_1, \ldots, v_j), \tau) = \gamma((v_1, \ldots, v_{j-1}), \tau) + c_{v_{j-1}, v_j}(\gamma(v_1, \ldots, v_{j-1}, \tau) + \tau).$$

For paths on labeled graphs, the function `Word(p)` returns the sequence of labels along the path p. The *concatenation* of two paths $p' = (v_1, \ldots, v_k)$ and $p'' = (v_{k+1}, \ldots, v_{k+n})$, with $1 \leq k < n$, is the path $p = p' \circ p'' = (v_1, \ldots, v_k, v_{k+1}, \ldots, v_{k+n})$.

2.3 Shortest Path Problem

The shortest path problem (`SP`) is the problem of finding a path p between two nodes in a graph such that its cost $c(p)$ is minimized. Relevant for us is the time dependent shortest path problem (`TDSP`):

Definition 2.3.1 (Time dependent shortest path problem (`TDSP`))**.** *Given a directed graph $G = (V, A)$ with a cost function $c : A \to \mathbb{F}$, a source node $r \in V$, a target node $t \in V$, a departure time τ_0, find a path $p = (s = v_1, v_2, \ldots, v_k = t)$ in G such that its time-dependent cost $\gamma(p, \tau_0)$ is minimum.*

The `TDSP` can be solved by adapted versions of Dijkstra's algorithm. See Chapter 4.

Definition 2.3.2 (Time dependent regular language constrained shortest path problem (`TDRegLCSP`))**.** *Given a directed and labeled graph $G = (V, A, \Sigma)$ with a cost function $c : A \to \mathbb{F}$, a source node $r \in V$, a target node $t \in V$, a departure time τ_0, and a regular language L_0 over Σ, find a path $p = (s = v_1, v_2, \ldots, v_k = t)$ in G such that its time-dependent cost $\gamma(p, \tau_0)$ is minimum and `Word(p)` is an element of L_0.*

The `TDRegLCSP` is a generalisation of the `TDSP`. We will treat it more in detail in Section 4.3.

The FIFO property. The FIFO (First-In-First-Out) property states that for each pair of time instance $\tau, \tau' \in \mathcal{T}$ where $\tau' > \tau$

$$c_{ij}(\tau) + \tau \leq c_{ij}(\tau') + \tau', \forall (i, j) \in A.$$

This means, in other words, that for any arc (i, j), if a car c_1 leaves node i earlier than another car c_2, FIFO guarantees that c_1 will not arrive later at node j than c_2. FIFO is also called then non-overtaking property. The `TDSP` in FIFO-networks is polynomially solvable [80], even when considering traffic lights [4]. It is NP-hard in non-FIFO networks [100]. The same considerations hold for `TDRegLCSP`.

Note that in some transportation networks overtaking is rare (such as in train networks). On the other hand, modeling of car transportation may yield networks where the FIFO property does not apply. See [96] for a mathematical programming formulation for the shortest path problem which takes into account time-dependency in non-FIFO networks and non-linear time-dependent cost functions. The author proposes some algorithms for the resulting MILP and MINLP problem formulations.

2.4 Summary

In this chapter, we introduced relevant notations from formal language theory and graph theory. We will use a labeled graph with time-dependent arc-costs to model a multi-modal transportation network (see Chapter 3). On this network, we will use a generalization of Dijkstra's algorithm to solve the `TDRegLCSP`. We use regular languages to impose constraints on the shortest path, e.g., to exclude certain modes of transportation or to limit the number of transfers (more details on this in Chapter 4). Finally, in Chapter 5, we will present a new algorithm which is able to solve `TDRegLCSP` faster than previous algorithms.

Chapter 3

Network Modeling

We use a labeled graph with time-dependent arc-costs to model a multi-modal transportation network. Efficient algorithms to compute shortest paths on such graphs exist (see Section 4.3). In this chapter, we first discuss graphs for single network types: foot, private bicycle, rental bicycle, private car, rental car, and public transportation. Then we describe how to combine the different networks to build a model of a multi-modal transportation network. We also discuss how we produced the graphs of two real world multi-modal transportation networks: of the French region Ile-De-France, which includes the city of Paris, and of New York City. These graphs will be used in later chapters to evaluate our shortest path algorithms.

3.1 Single Networks

In this section, we introduce approaches to model foot, bicycle, road, and public transportation networks.

3.1.1 Foot network

The construction of the model of the foot network is straightforward. Junctions are represented by nodes, and arcs between two nodes exist if there is a footpath between two junctions (see Figure 3.1 for an example). Arc costs depend on geographical length and on average pedestrian walking speed. Labels on arcs mark the type of path, such as side-way, stairs, trail, etc.

3.1.2 Bicycle network

The bicycle network is constructed in a similar way to the foot network. Nodes represent road junctions and arcs are inserted into the graph whenever biking is allowed between two junctions. Different road types for cycling exist, such as roads with a separate cycling path, roads with a separate lane for cyclists which are shared with buses, roads shared with general traffic, etc. Arcs may be labeled accordingly. Arc costs represent cycling time and depend on geographical length and on average cycling speed. See [114] for information about multi-

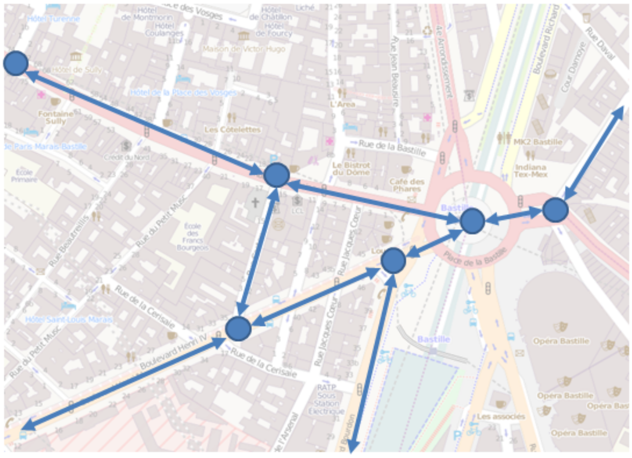


Figure 3.1: Model of the foot path. Road junctions are represented by nodes and arcs between two nodes represent footpaths between two junctions.

critera routing on cycling networks and for advanced graph modeling of cycling networks which also consider security aspects[1].

3.1.3 Road network

For the road network, nodes represent road junctions and arcs represent roads connecting the junctions. Different road types exist, e.g., local roads, urban roads, inter-urban roads, motorways, toll roads, etc. Arc costs may be time-dependent and represent travel time which depends on the geographical length, the speed limits, and the traffic conditions (see Figure 3.2 for an example of a cost function and Section 3.3.1 for more information about traffic data).

3.1.4 Public transportation network

A public transportation network consists of buses, subways, tramways, ferries, local trains, etc. Several approaches for modeling a public transportation network exist. We will shortly introduce the *condensed*, the *time-expanded*, and the *time-dependent* model. We refer to [107, 117] for an in-depth discussion. All these models are based on a *timetable*. We will introduce this concept first.

Timetable

A *timetable* is a 4-tuple $(\mathcal{C}, \mathcal{B}, \mathcal{Z}, P)$: $\mathcal{B}$ is a set of stations where vehicles stop (e.g., bus stops, metro or train stations), $\mathcal{Z}$ is a set of vehicles, P is the periodicity of the timetable, and $\mathcal{C}$ is a set of elementary connections. An *elementary connection* is defined by a 5-tuple $c = (z, s_1, s_2, \tau_1, \tau_2)$. A vehicle $z \in \mathcal{Z}$ travels from station $s_1 \in \mathcal{B}$ to station $s_2 \in \mathcal{B}$ departing from s_1 at time τ_1 and arriving at s_2 at time τ_2. Note that $\tau_1 < P$, $\tau_2 < P$, and that the vehicle does not stop at any other stop while going from s_1 to s_2. Note also that it is possible to depart in the evening and to arrive the next day. The travel time δ of an elementary connection is calculated as follows:

[1] See vgps.paris.fr for an online routing service for cyclists which optimizes distance and security of the path.

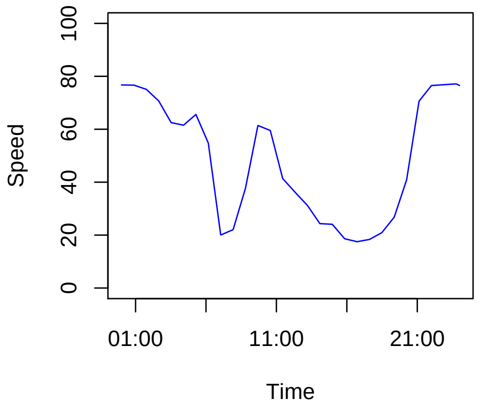


Figure 3.2: Typical cost function of a road segment. Average speeds (in km/h) in the evening and during the night are close to the speed limit (80km/h). During rush hour in the morning and late afternoon, congestion increases and the average speed on the road segment is much lower (20km/h). Over midday, traffic conditions are slightly better and the average speed is about 60km/h.

$$\delta(\tau_1, \tau_2) = \begin{cases} \tau_2 - \tau_1 & \text{if } \tau_2 \geq \tau_1 \\ P - \tau_1 + \tau_2 & \text{otherwise} \end{cases} \tag{3.1}$$

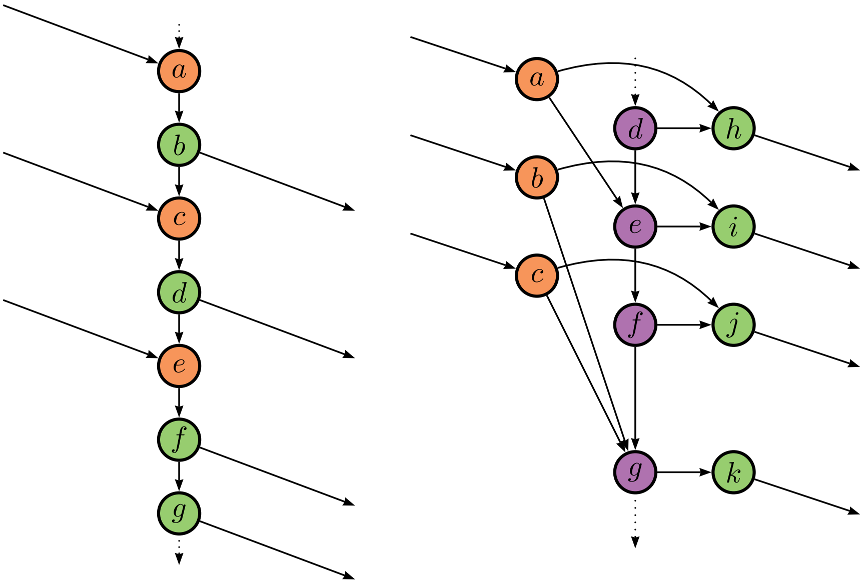
Condensed model

A simple approach to model a public transportation system is the use of the condensed model. It represents the network structure but not the scheduling, thus it is non time-dependent. Examples of condensed models are subway or bus maps which only indicate stations, connections, and lines, but no departure or travel times. Every station $s \in \mathcal{B}$ is represented by exactly one node $v \in V$ in the graph. An arc (s_i, s_j) is introduced if and only if at least one elementary connection exists in the timetable that goes from station s_i to station s_j. Arc costs are omitted or are defined as the minimum travel time over all elementary connections from s_i to s_j. The condensed model provides an overview of the structure of the public transportation system but is not useful for exact shortest path calculations.

Time-expanded model

A more complete representation of a public transportation system provides the time-expanded model. Two versions exist: a simple version and a realistic version. The latter incorporates realistic transfer times between two vehicles at stations.

Simple version. Nodes represent *departure events* and *arrival events*. For each elementary connection $c = (z, s_1, s_2, \tau_1, \tau_2)$ two nodes are created: a departure node v, which represents the departure event of vehicle z at station s_1 at time τ_1, and an arrival node u which represents the arrival event of vehicle z at station s_2 at time τ_2. Each node is assigned its



(a) Simple time-expanded model. (b) Realistic time-expanded model.

Figure 3.3: Representation of a station in the simple and the realistic version of the time-expanded model. Arrival, departure, and transfer nodes are colored orange, green, and violet, respectively. Note that the third train leaving can be reached from the second train arriving in the simple version but not in the realistic version.

station s and its timestamp τ when the event occurs. Two types of arcs exist, *travel arcs* and *internal station arcs*. A travel arc connects the node representing the departure event with the node representing the arrival event of an elementary connection c. The arc cost represents travel time and is set to $\delta(\tau_1, \tau_2)$. Internal station arcs are only inserted between nodes which are associated with the same station. First, all nodes associated with the same station are sorted in ascending order with respect to their timestamp; we obtain an ordered series of nodes $(v_1, \ldots, v_k)$. Then for two subsequent nodes v_i, v_j having timestamps τ_i and τ_j, a transfer arc $e := (v_i, v_j)$ is inserted in the graph. It has arc cost $\delta(\tau_i, \tau_j)$ which represents the waiting or transfer time between arrival and departure event. To allow transfers over the end of the period (midnight), an arc connecting the node v_k with the latest timestamp and the node v_1 having the lowest timestamp is inserted into the graph. See Figure 3.3 for an example.

Realistic version. The simple version is enhanced by inserting *transfer nodes* and *transfer arcs* to correctly model and include transfer times at stations. See Figure 3.3 for an example.

Time-dependent model

In the time-dependent model, travel times are represented by functions. Again two versions exist [107, 117]: a simple and a realistic version. The realistic version incorporates transfer times.

Simple version. The simple version is an expansion of the condensed model. There is a node for every station in $\mathcal{B}$ and an arc if there exists at least one elementary connection between the two stations represented by the node. Unlike the condensed model, arc costs

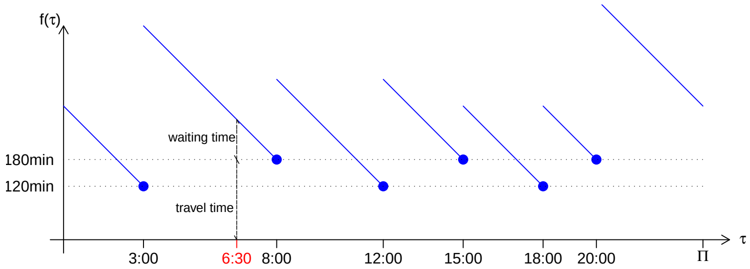


Figure 3.4: A piecewise linear function with 6 interpolation points. There are 3 fast trains departing at 3:00, 12:00, and 18:00, and 3 slow trains at 8:00, 15:00, and 20:00. The fast trains take 120min and the slow trains take 180min to reach the target station. If a passenger arrives at the station at 6:30, then the journey to the arrival target takes 270min: 90min waiting time and 180min travel time.

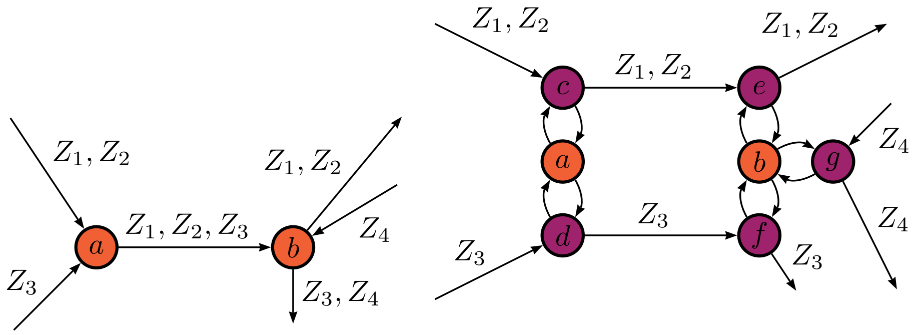
are *time-dependent* and are represented by piecewise linear functions. For each elementary connection $c = (Z, s_1, s_2, \tau_1, \tau_2)$ an interpolation point $p = (\tau_1, \delta(\tau_1, \tau_2))$ is added to the function f of the arc between stations s_1 and s_2. τ_1 is the departure time and $\delta(\tau_1, \tau_2)$ the travel time. To evaluate the function f for a time point τ we apply the equation

$$f(\tau) = \underbrace{(\tau_i - \tau)}_{\text{waiting time}} + \underbrace{f(\tau_i)}_{\text{travel time}} , \tag{3.2}$$

where $\tau \leq \tau_i$ and τ_i is the departure time of the closest interpolation point. $\tau = \tau_i$ means that the passenger arrives at station s_1 exactly at the time of departure. If $\tau < \tau_i$ then the passengers arrives early at the station and his travel time to s_2 equals the sum of the waiting time at the station s_1 and the travel time of the vehicle. See Figure 3.4 for an example.

Realistic version. The realistic version of the time-dependent model considers also transfer times at stations. It consists of *station nodes* which represent public transportation stations, such as those pictured on subway network maps, and *route nodes*. Route nodes can be pictured as station platforms and are connected by *transfer arcs* to station nodes. The construction of this model proceeds as follows. First, for every station a station node is inserted into the graph. Then the route nodes are inserted: A *trip* of a vehicle is defined as the sequence of stations it visits according to its elementary connections defined in the timetable. Trips consisting of the exact same sequence of stations are grouped into *routes*. For every station included in a route, a route node is inserted in the graph and connected by transfer arcs to the corresponding station node. Note that it is possible for a station that more than one route node is inserted in the graph in case more than one route includes the station. Subsequent route nodes belonging to the same route are connected by travel arcs with time-dependent cost functions.

The realistic version can be further generalized. Transfer times between platforms of the same station may vary considerably especially in big stations. These transfer times can be incorporated into the model by inserting additional transfer arcs between route nodes. Furthermore, station nodes can be duplicated to represent entrances of the station and to provide access points to other networks, such as the foot network. However, for such a detailed representation, real world data might not always be available. Furthermore, the model becomes more complex especially regarding the number of transfer arcs [107].



(a) Simple time-dependent model. (b) Realistic time-dependent model.

Figure 3.5: Representation of two stations and four vehicles in the simple and in the realistic version of the time-dependent model. Station and route nodes are colored orange and violet, respectively. Note that vehicles Z_1 and Z_2 belong to the same route.

FIFO-property. The `TDSP` on a time-dependent graph of a public transportation network can be solved efficiently when the arc cost function f is non-negative and FIFO. In our case, non-negativity is ensured by construction. We have to make sure that it is FIFO: $\tau \leq \tau' \Rightarrow \tau + f(\tau) \leq \tau' + f(\tau')$. We will assume that the condition is always fulfilled. This is reasonable as, e.g., trains with different speeds almost never serve the exact same sequence of stations (slower trains stop more frequently), i.e., they do not belong to the same route. Even if that were the case, FIFO can be ensured by duplicating the route and assigning the slower train to one route and the faster train to the other. This should only seldom be the case for public transportation systems, so the increase in graph size is insignificant.

Conclusions

The time-expanded model *rolls out* the time schedule of the public transportation timetable. It allows exact shortest path queries and it is easy to adapt the Dijkstra's algorithm to work on this model [117]. But there are two major disadvantages of this model: First, as the exact arrival node is not known in advance, bi-directional speed-up techniques are difficult to apply. Second, the size of the graph is very large even for small networks. This leads to a large search space and high memory requirements. For these reasons, we will use the time-dependent model in this work. In [38], the authors show techniques on how to reduce the size of the time-expanded model and its impact on query times. Find more information on multi-modal routing on public transportation networks in Section 4.2.

3.1.5 Rental bicycle and rental car networks

Major cities in recent years have adopted bicycle rental systems, e.g., Paris (France), London (UK), Washington DC (US), Hangzhou (China). Besides providing affordable access to bicycles for short-distance trips in an urban area as an alternative to motorized vehicles, these systems are also supposed to cover the *Last Kilometer*, i.e., the distance between public transportation stops and the departure or final location. For this reason, they are an important component of a multi-modal transportation network.

We consider bicycle sharing systems with fixed rental stations, like the bicycle sharing systems in operation in Paris[2] or Washington DC[3]. In such systems, users retrieve and return bicycles at bicycle rental stations distributed evenly over the served area. Each rental station v is characterized by a maximal number of bicycles it can hold $c_v^{\max} \in \mathcal{Z}^+$. Bicycle rental costs may also be an important characteristic of a rental bicycle system. The modeling of a rental bicycle system is straightforward. It uses the bicycle network, and nodes representing locations near bicycle rental stations are labeled accordingly. Shortest path queries are only allowed between these labeled nodes.

Recently, similar rental principles have been adapted to rental car systems. See for example the Autolib' system[4] in Paris which offers electric cars to rent for short trips. The modeling of a rental car system is similar to the modeling of a bicycle rental system, but instead of using the bicycle network, the car network is used. Nodes of the car network next to car rental stations are labeled accordingly.

Availability of rental vehicles. Rental vehicles are subject to availability. Furthermore, they can only be returned at stations with free return slots. This information must be known before a shortest path query is started. It is easy to incorporate this constraint in the `Dijkstra` algorithm. It suffices to assign a flag to each node v labeled as a station, which is true, if rental vehicles are available ($f^{\text{bike\_avail}}(v) = true$), and a second flag which is true if return slots are available ($f^{\text{free\_slots}}(v) = true$). The flags have to be updated periodically.

3.1.6 Locations of interest

The user may want to pass by an intermediate stop before reaching the target location. However, a user might not specify the precise geographical location of this stop and just wish to pass by any pharmacy, post-office, supermarket, etc. In this case, the route planning service has to determine autonomously the exact location of the intermediate stop and adapt the path between source and target location accordingly. We use the following simple approach to include information about such *locations of interest* in a graph. We duplicate arcs modeling roads on which locations of interest are located and we substitute the assigned label with a new label indicating the presence of, e.g., a pharmacy along a footpath. Note that in this way no information on the time spent at the location of interest can be incorporated in the model. However, this can easily be done by appropriately expanding the graph and by modeling a location of interest through the insertion of additional nodes.

3.2 Multi-Modal Network

In this section, we combine the different networks described in the previous section into a single *multi-modal network*. This requires the choice of the locations at which different networks meet (e.g., public transportation stations, parking places). Once the locations are identified the networks have to be linked by the insertion of *transfer arcs* between nodes of different networks. A technical difficulty is the identification of nodes which are located close to each other in a graph. We discuss this problem first.

[2] Vélib', www.velib.fr

[3] Capital Bike Share, www.capitalbicycleshare.com

[4] Autolib',www.autolib.eu

The nearest neighbor problem

Let $\mathbb{R}^n$ be the n-dimensional vector space over $\mathbb{R}$ and $P \supset \mathbb{R}^n$ a finite set of vectors. The set P is called *candidate points*. Let $d : \mathbb{R} \to \mathbb{R}$ be a metric on $\mathbb{R}^n$.

Definition 3.2.1. NEAREST NEIGHBOR PROBLEM. *Given a metric space $(\mathbb{R}^n, d)$, a set of candidate points P on $\mathbb{R}^n$ and a set Q of query points on $\mathbb{R}^n$ we ask for a map $f : Q \to P$ with the property*

$$f(q) = p \iff \forall p' \in P : p \neq p' \Rightarrow d(p', q) \geq d(p, q). \tag{3.3}$$

In other words, for a query point q, we try to find the *nearest* candidate point $p \in P$ with respect to d. To solve this problem, *Linear Search* can be applied. This is a naive approach and consists of scanning the list of candidate points P for each query point $p \in Q$. This requires the distance computation between each pair $(q, p) \in Q \times P$. While the implementation of this algorithm is straightforward, its runtime $O(|Q||P|)$ might be too high for large sets of P and Q. A better search strategy consists in using *k-d-trees* during the search process; k-d-trees, or k-dimensional trees, are a data structure designed for geometric search algorithms [20]. How to use k-d-trees to solve the NEAREST NEIGHBOR PROBLEM is briefly mentioned in [20] and is more extensively discussed in [92]. A good implementation is provided by [81]. It can compute the single nearest neighbor of q, and also the m closest points to q. The algorithm consists of two phases. First, a k-d-tree is created with all candidate points P, this can be done in $O(n \log n)$. Second, for each query point $q \in Q$ a query on the data structure is started which yields the nearest neighbor of q, this takes $O(\log |P|)$. Running time for finding the neighbors for all $q \in Q$ is thus $O(|Q| \log |P|)$.

Combining the networks

Given a number of uni-modal graphs $G_1, \ldots, G_n$ each having node, arc, and label sets $G_i = (V_i, A_i, \Sigma_i)$, the *combination* yields a multi-modal graph $G^{\text{mm}} = (V^{\text{mm}}, A^{\text{mm}}, \Sigma^{\text{mm}})$. The node and label sets are the union of the node and label sets of the input graphs, thus $V^{\text{mm}} = V_1 \cup \cdots \cup V_i$ and $\Sigma^{\text{mm}} = \Sigma_1 \cup \cdots \cup \Sigma_i$. The arc set is the union of the arc sets of the input graphs and a set of *transfer arcs* A^{transfer}, so $A^{\text{mm}} = A_1 \cup \cdots \cup A_i \cup A^{\text{transfer}}$. The transfer arcs are used to connect the uni-modal graphs together, i.e., to make it possible to transfer from one uni-modal graph to another. Transfer arcs to connect two graphs G_i and G_j are created in the following way:

1. Definition of two sets which contain the *link nodes*, i.e., nodes from which transfer to the other network should or may be possible: V_i^{link}, V_j^{link}.

2. Resolution of a NEAREST NEIGHBOR PROBLEM instance where V_i^{link} is the candidate set and V_j^{link} is the query set, or vice versa. Note that the geographical location of nodes given in x and y coordinates as latitude and longitude values have to be available.

Let us now look more in detail at how the networks are linked. Note that we will link each network exclusively to the foot network. This means that there are, for example, no transfer arcs between the car and the bicycle network and thus direct transfer from the car network to the bicycle network is not possible. This is reasonable, as most transfers involve

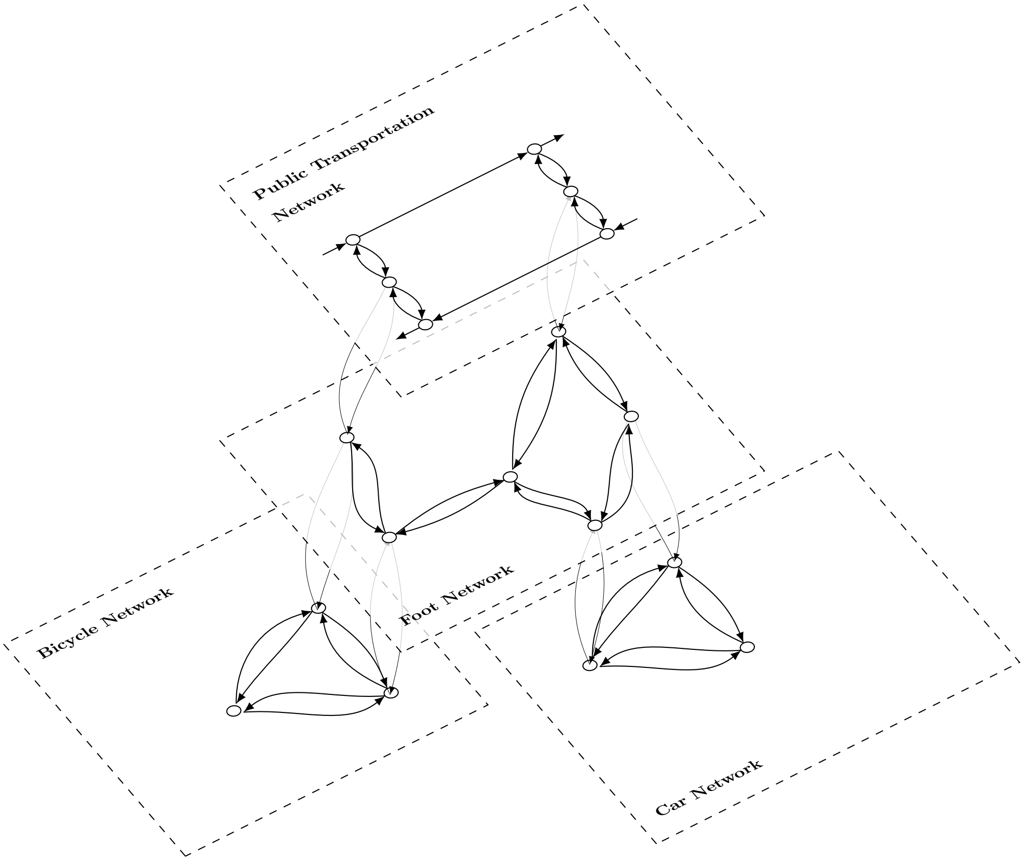


Figure 3.6: The graph consists of 4 layers which are connected by transfer arcs.

at least some walking. In the following, we illustrate how we chose the link nodes for each network. See Figure 3.6 for a schematic representation of the multi-modal graph.

Foot ↔ Bicycle: Private Bicycle, Rental Bicycle. The bicycle network can be accessed by private bicycle from anywhere in the foot layer. So we simply connect every node in the foot network to its nearest neighbor node in the bicycle network. On the other hand, accessing the bicycle network by rental bicycle is only possible at bicycle rental stations. For each bicycle rental station, we find the nearest node on the foot network and the nearest node on the bicycle network and connect the two nodes. To distinguish transfer arcs representing transfer at bicycle rental stations and transfer arcs representing access to the bicycle network by private bicycle, we assign different arc labels.

Foot ↔ Car: Private Car, Rental Car. The car network can only be accessed at points where a private car can theoretically be parked, e.g., parking is not allowed along motorways and high speed roads. Thus, we identify all nodes of the road network belonging to low road classes and connect them to their nearest neighbor nodes on the foot network. We handle rental car stations in the same way as we handle rental bicycle stations.

Foot ↔ Public Transportation. Switching from the foot to the public transportation network is only possible at public transportation stations, i.e., the station nodes. We insert transfer arcs from station nodes to the nearest nodes in the foot network.

Accuracy. Note that we introduced a series of inaccuracies. In the real world, changing from the foot network to the car or bicycle network can also be done at other locations than road junctions. Furthermore, we assign bicycle and car rental stations to the nearest junctions. This might displace rental stations for several meters from their actual location. A more accurate solution is to introduce additional nodes at the exact locations and to appropriately connect them to the foot, bicycle, and car network. Furthermore, coordinates are in geographical form (latitude and longitude). Using the Euclidean metrics lead to inaccurate results when computing the distance between two points. The error increases the greater the distance between the two points. To avoid this problem, geodetic distances on a solid resembling the form of the earth can be used, e.g., the GRS80-ellipsoid [94] which is also used by the Global Positioning System (GPS). Nevertheless, we believe that our model sufficiently mirrors reality so that our experimental results are representative. However, for real world applications, these issues have to be kept in mind.

3.3 Application

For the evaluation of our algorithms, we produced two multi-modal graphs: the graph IDF based on data of the French region Ile-de-France (including the city of Paris) and the graph NY based on data from New York City. In this section, we discuss how we produced these two multi-modal graphs and the data we used.

The graphs are divided into four layers (see Figure 3.6): 1) walking, 2) road, 3) cycling, and 4) public transportation layer. Each layer is connected to the walking layer through transfer arcs: arcs with label t_c and t_a mark transfer arcs to access the road network either by private or by rental car, arcs with label t_b and t_v mark transfer arcs to access the bicycle network either by private or rental bicycle, and arcs with label t_p mark transfer arcs to the public transportation network. 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) and we set it to 20 seconds. In addition, in both graphs we introduced twenty arcs with label z_{f_1} and another twenty arcs with label z_{f_2} between nodes of the foot layer. Furthermore, we introduced twenty arcs with label z_{c_1} and another twenty arcs with label 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 to pass by any pharmacy, grocery shop, etc. See Section 3.1.6.

3.3.1 Multi-modal transportation network IDF (Ile-de-France)

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), see map in Figure 3.7. It consists of four layers, bicycle, walking, car, and public transportation, and has circa 3.9m arcs and 1.2m nodes. See for more detailed information Table 3.1.

Data of the public transportation network have been provided by STIF[5]. They include geographical information, as well as timetable data on bus lines, tramways, subways, and regional trains. The public transportation layer is reachable through transfer arcs (label t_p) at public transportation stations, i.e., subway stations, bus stops, etc.

[5] Syndicat des Transports IdF, www.stif.info, data for scientific use (01/12/2010)

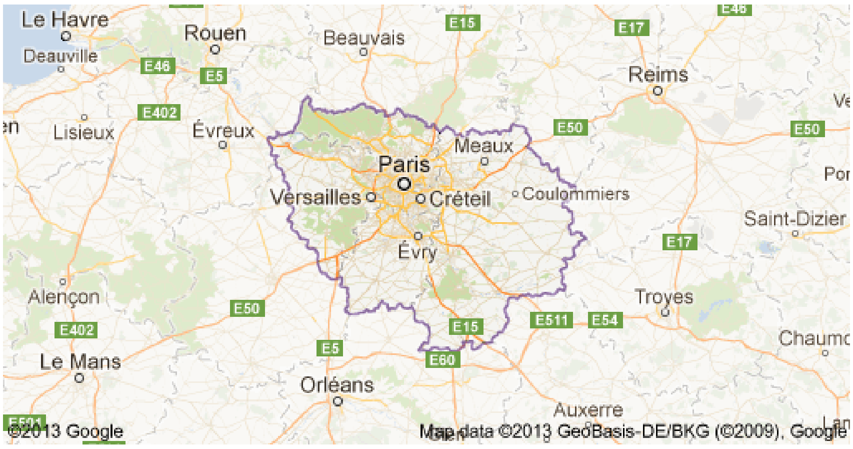


Figure 3.7: Ile-de-France (Google Maps).

Table 3.1: Graph IDF (Ile-de-France).

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 station intern, 176 790 (omitted in automata and regular expressions for simplicity)), time-dependent 82 833
bike	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)

Data for the car layer is based on road and traffic information provided by the French company Mediamobile[6]. Arc labels are set according to the road type. Arc cost equals travel time which depends on the type of road (motorway, side street, etc.). Circa 15% of the road arcs have a time-dependent cost function to represent changing traffic conditions throughout the day (see Section 3.3.1). Transfers from the car layer to the walking layer are possible at uniformly distributed transfer arcs (label t_c) or, if a rental car is used, at car rental stations[7] (label t_a). Car rental stations are located in Paris and its surroundings and cars are assumed to always be available.

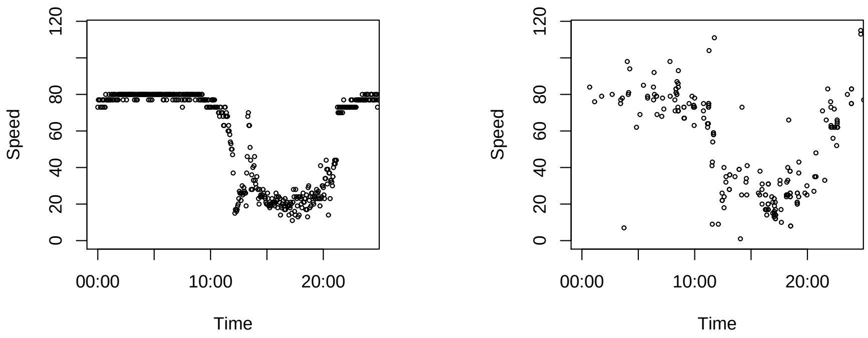
The walking as well as the bicycle layer are based on road data (walking paths, cycle paths, etc.) extracted from geographical data freely available from OpenStreetMap[8]. Arc cost equals walking or cycling time (pedestrians 4km/h, cyclists 12km/h). Arcs are replicated and inserted in each of the layers if both walking and biking are possible. Rental bicycle stations are located mostly in the area of Paris[9]; they serve as connection points between the walking layer and the bicycle layer, as rental bicycles have to be picked up and returned at bicycle rental stations (label t_v). The private bicycle layer is connected to the walking layer at common street intersections (label t_b).

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

[7] Autolib', www.autolib.eu

[8] See www.openstreetmap.org

[9] Vélib', www.velib.paris.fr



(a) Loop Sensor.

(b) FCD.

Figure 3.8: Example of a speed profile of a road section based on data provided by a loop sensor (left) and a FCD provider (right). Both graphs show vehicle speeds over the day. It can be seen that speed is much lower during the day than during the night because of higher traffic. FCD data contain high noise and many outliers and filter algorithms have to be applied. Loop sensor data is more precise but does not cover the whole network.

Traffic Information

Traffic information is provided by the French company Mediamobile. Figures 3.9 and 3.10 show the arcs of the data set we are using for which traffic information is available, i.e., the cost function of arcs is time-dependent. Figure 3.11 shows real-time traffic data of the city of Paris as it is diffused via the company's public website.

Producing traffic data involves collecting raw data and processing the data. While the collection step is straightforward, it is not trivial to produce high quality traffic data and to determine if raw data are significant and representative of real traffic conditions. For more details, see [82, 127]. Raw traffic data are provided by traffic data sources. They provide information about vehicle flow on stretches of roads, such as vehicle speeds, counts of vehicles, and travel time. Traffic data sources may rely on multiple technologies. Mediamobile works mainly with two types of traffic data sources: loop sensors and floating car data.

Loop sensors. Loop sensors are electromagnetic loops which are buried under the road. Whenever a heavy metallic item passes above a loop, a variation in the electromagnetic field occurs, an electrical power is created, and a voltage can be recorded. To spot vehicles, a detection threshold on this voltage is fixed. Loop sensors can count the number of passing vehicles and by setting two loops near each other on a stretch of road, they can also provide information about vehicle speed. See Figure 3.8a for an example of data produced by loop sensors.

Floating Car Data (FCD). Vehicles equipped with GPS devices communicate their positions to a central server at pre-defined intervals. These data are called floating car data. A map-matching algorithm calculates speeds $V(x, t)$ on a road link x at a time t from a series of successive positions and times by matching them on a network. One of the major differences between FCD and Loop Sensors is that FCD is not geographically dependent on counting stations. Connected devices are increasingly popular and this makes FCD a

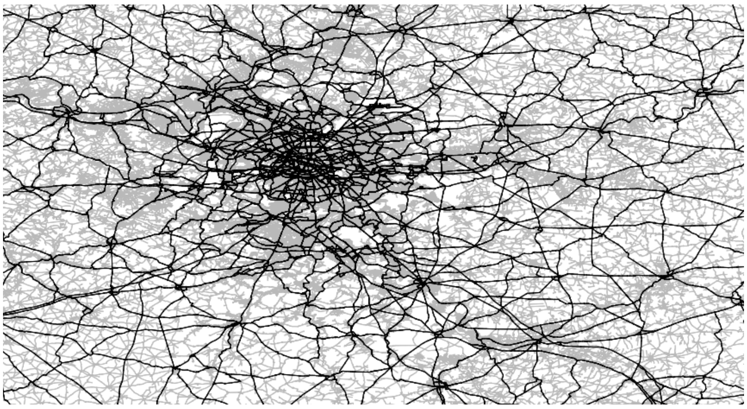


Figure 3.9: Roads with time-dependent cost functions (black), Ile-de-France (France).

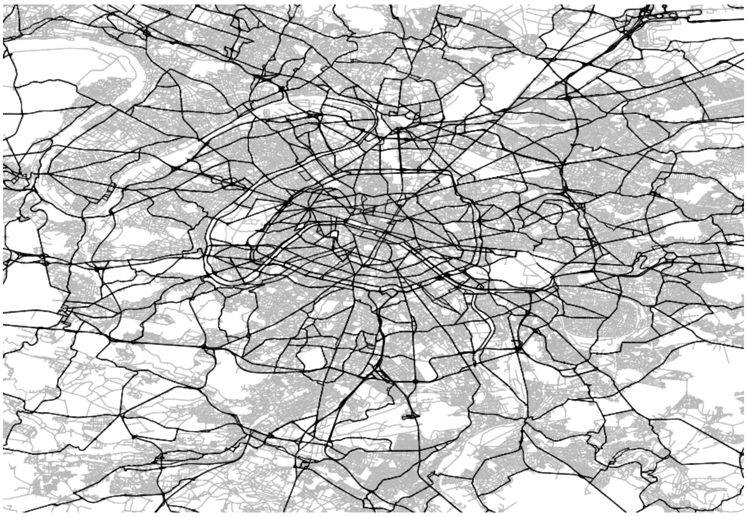


Figure 3.10: Roads with time-dependent cost functions (black), Paris (France).

relevant source of traffic data as it can cover potentially the entire road network. Unlike data collected by loop sensors, FCD data usually exhibit high noise and many outliers due to GPS logger accuracy and projection errors on the road network (see Figure 3.8b).

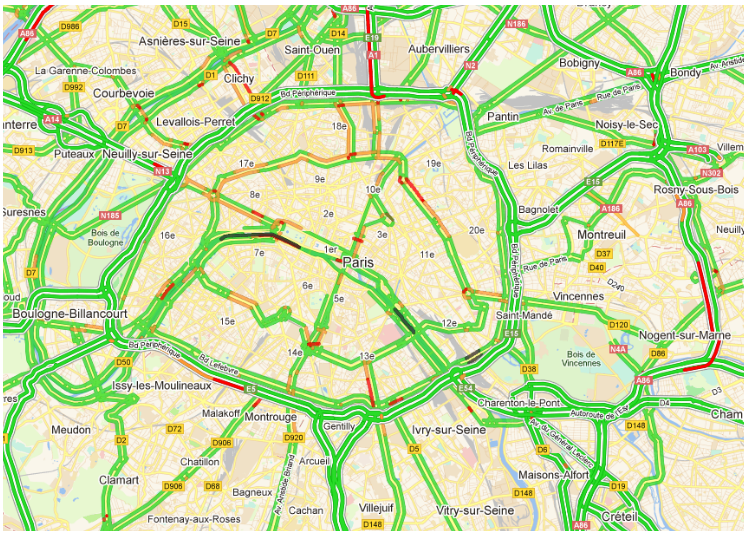


Figure 3.11: Real time traffic information of the city of Paris on the public website www.v-traffic.com provided by the company Mediamobile. Green, orange, and red roads indicate low, intermediate, and heavy traffic, respectively.

Table 3.2: Graph NY (New York City).

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 station intern, 52 956 (omitted in automata and RE)), time-dependent arcs 25 834
bike	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 (time points 3 572 498)

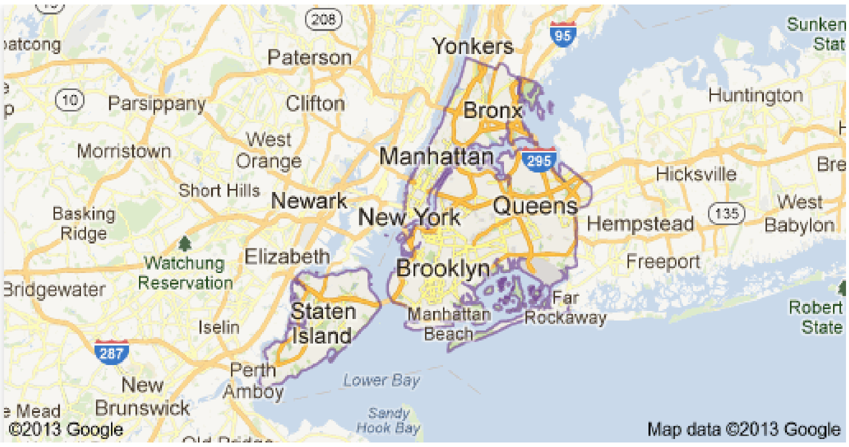


Figure 3.12: New York City (Google Maps).

3.3.2 Multi-Modal Transportation Network NY (New York City)

The graph NY is composed of data of the road and public transportation system of New York City (see map in Figure 3.12). 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[10]. See Table 3.2 for detailed information.

3.4 Summary

In this chapter, we showed how to produce a model of a multi-modal transportation network by means of a labeled graph. We used time-dependent edge costs to include traffic information and timetable information. Efficient algorithms exist to calculate shortest path on such a graph. This will be the topic of the next chapter.

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

Chapter 4

Shortest Path Problem

The shortest path problem (SP) is a widely studied research topic because of its high relevance in many practical applications. In particular, transportation theory provides many applications of variations of shortest path problems. Much of the early work on this topic has been carried out in the 1950s and 1960s at the RAND corporation. It focused mostly on transportation network analysis [18]. The first algorithms to optimally solve shortest path problems have been presented by Dijkstra [51], Bellman and Ford [19, 55], and Hart, Nilsson and Raphael [68]. They are based on the *labeling method* which we will introduce in Section 4.1.

While these algorithms compute optimal shortest paths, they are too slow to be employed in applications based on large real world data sets, such as the road network of a whole country. Therefore, research in recent years has focused on the development of *speed-up techniques* to accelerate these early algorithms by reducing their search space. Many techniques have been proposed for the computation of shortest paths on static road graphs, and algorithms capable of finding shortest paths in a few microseconds exist. Some of these ideas have been adapted to dynamic scenarios, i.e., scenarios where arc costs are updated regularly to represent the actual traffic situation, and the time-dependent scenario. An overview can be found in [40]. We will shortly introduce some of the most important works on this topic and we will treat bi-directional search and the ALT algorithm in detail, as we will use these techniques for our new algorithm SDALT, which we will present in Chapter 5. Some of the speed-up techniques for shortest path algorithms on road networks have also been applied to public transportation networks. We will discuss speed-up techniques for shortest paths on road networks and public transportation networks in Section 4.2.

We focus on finding shortest paths on multi-modal networks, which we will discuss in Section 4.3. Other than minimizing travel time, shortest paths on such networks must satisfy some additional constraints. First of all, feasibility has to be assured: private cars or bicycles can only be used when they are available. Second, passenger preferences should be respected: passengers may want to exclude some transportation modes, e.g., the bicycle when it is raining or the car at moments of heavy traffic. The regular language constrained shortest path problem (RegLCSP) deals with this kind of problem and can be solved in polynomial time by a generalization of Dijkstra's algorithm (which we will call $\mathsf{D}_{\mathsf{RegLC}}$). In Chapter 5, we will show how to adapt ALT and bi-directional search to speed-up $\mathsf{D}_{\mathsf{RegLC}}$.

4.1 Labeling Method

The *labeling method* for the shortest path [56] finds shortest paths from a source node r to all other nodes in a graph. It works as follows: it maintains for every node v a tentative distance label $d(v)$, a parent node $p(v)$, and a status $S(v)$. A status of a node can be `unreached`, `explored`, or `settled`. Initially, for every node v, $d(v) = \infty$, $p(v) = \mathsf{nil}$, and $S(v) =$ `unreached`. The algorithm starts by setting $d(r) = 0$ and $S(v) =$ `explored`. At every successive iteration, the algorithm selects a node with status `explored`, it *relaxes* all outgoing arcs, and sets its status to `settled`. Relaxing an arc (v, w) means to first check if $d(w) > d(v) + c_{vw}$, and, if that is the case, to set $d(w) = d(v) + c_{vw}$, $p(w) = v$, and $S(w) =$ `explored`. The algorithm terminates when there are no more nodes with status `explored` and if the graph does not contain cycles with negative cost, it produces a shortest path tree, i.e., all distances and shortest paths starting at the source node r to all other nodes of the graph. The shortest path to a node v can be retrieved by following the parent nodes backward starting at v.

4.1.1 Label setting methods and Dijkstra's algorithm

The order in which the next node to be scanned is selected highly influences the efficiency of the labeling method. On a static graph with non-negative arc cost, it is easy to see that if the algorithm always selects nodes v for which $d(v)$ corresponds exactly to the shortest distance between r and v, then each node is scanned at most once and arcs (w, v) for which the status of v is settled have to not be relaxed. In this case, the algorithm is called *label setting*. If the cost function is non-negative and the node to be explored at each step is the one with smallest distance label $d(v)$, then $d(v)$ will correspond exactly to the shortest distance between r and v. This has first been observed by Dijkstra [51] and the labeling method with minimum distance label selection rule is referred to as Dijkstra's algorithm, which we will call `Dijkstra`. It scans nodes in non-decreasing order of distance to the source node r. The algorithm terminates as soon as the target node t is assigned the status `settled`. Complexity depends on the priority queue used to hold the distance labels. Runtime is in $O(|E| + |V| \log |V|)$ if a Fibonacci heap [57] is used and $O((|E| + |V|) \log |V|)$ if a binary heap is used. Computing the shortest path from r with starting time τ_{start} on a graph with time-dependent arc costs can be solved by a slightly modified algorithm: when relaxing arc (v, w), it evaluates arc costs for time $\tau_{\mathsf{start}} + d(v)$, so $d(w) = d(v) + c_{vw}(\tau_{\mathsf{start}} + d(v))$ [27, 53]. `Dijkstra` can also be applied to dynamic networks [25, 26].

4.1.2 Label correction methods

We will call *label correcting methods* those labeling methods which may update distance labels for nodes with status `settled`. In this case, a distance label may be re-inserted in the priority queue, and the status of a node may change from `settled` back to `explored` multiple times. On a time-dependent graph, label-correcting algorithms may be used to compute the distant function between $d_*(r, t) : \mathcal{T} \to (R), d_*(r, t)(\tau) = d(r, t, \tau)$, which finds the cost of the shortest path for every starting time $\tau \in \mathcal{T}$ [32]. It is implemented similarly to `Dijkstra`, but uses functions instead of scalars as distance labels.

Dijkstra works correctly only on graphs with non-negative arc costs. To find shortest paths on graphs with negative arc costs, the Bellman-Ford algorithm [19,93] can be used. In its basic structure it is similar to Dijkstra algorithm, but instead of selecting the minimum-weight node not yet processed to relax, it simply relaxes all the arcs, and does this $|V| - 1$ times. Runtime of Bellman-Ford is in $O(|V||E|)$. Note that if a graph contains a negative cycle, i.e., a cycle with total negative arc cost, then paths of arbitrarily low weight can be constructed by repeatedly following the cycle. Such cases can be detected by the Bellman-Ford algorithm, but a correct shortest path cannot be produced. The label correction method is relevant for our algorithm SDALT which we will discuss in Chapter 5. One variant of SDALT will use the label correction method (Section 5.3).

4.2 Uni-Modal Routing

In this section, we introduce some of the most recent and advanced speed-up techniques for shortest paths on road and public transportation networks. We will discuss bi-directional search and the ALT algorithm in detail, as we will use these techniques for our algorithm SDALT, which we will present in Chapter 5.

Routing on road networks

Running times of Dijkstra on large road networks are far too high for practical application. In the early years of the 2000s, huge road networks were made publicly available which stimulated widespread research on speed-up techniques for Dijkstra. This culminated in the 9th DIMACS Challenge on shortest paths [45], where many new fundamental works on efficiently finding shortest paths were presented. Three basic components are common to many speed-up techniques of shortest path algorithms on road networks: bi-directional search, goal-directed search, and contraction [40, 116]. Also table lookups are becoming increasingly relevant [1, 12]. The fastest search techniques use a combination of these basic components. In the following, we will introduce these basic components as well as some of the more recent fast algorithms. Relevant for this work are bi-directional search and goal-directed search (the ALT algorithm). We will discuss them in detail in Sections 4.2.1 and 4.2.2.

Bi-directional search. When applying bi-directional search, two searches are started, one starting at the source node and another starting at the target node. The search stops when the two searches meet and the shortest path is the concatenation of the partial paths produced by the two searches. This approach has been introduced and refined by [31, 65]. Bi-directional search is quite a simple technique and works well on static graphs with no time-dependent cost functions. Its application on time-dependent graphs is more complicated [98].

Goal-directed search. The ALT algorithm [65, 66] is a goal directed search as it preferably settles nodes that are close to the target. It combines the characteristics of the A* algorithm [68], the use of landmarks, and the triangle inequality, and is very robust and works in dynamic and time-dependent scenarios [17,42]. A second goal directed approach on static networks is called Geometric Containers [128]. This approach has been enhanced by [85] yielding the *Arc-Flags* algorithm. Here arcs of the network are explored by the search only if they are relevant for shortest paths toward nodes near the target node. The algorithm PCD [89] exploits precomputed distances between clusters of the graph to produce upper and lower bounds on the distance to the target to reduce the search space. All these search techniques require a preprocessing phase. Usually there is a trade-off between memory size of preprocessed data and query time.

Contraction. The objective of contraction is to identify the most relevant parts of the road network. The search query then concentrates on this limited set of roads. Note for example that most side and secondary roads are not relevant for shortest paths where travel time is minimized, except when located near the source or target node. *Highway Hierarchies* [111, 112] exploit the implicitly present hierarchy of real world road networks (road classes). A different approach is *Contraction Hierarchies* [58,61]. It is an uncomplicated and quite simple yet very efficient speed-up technique on static road networks. The adaption of Contraction Hierarchies to time-dependent road networks is shown in [13, 14]. The authors of [60] adapt contraction to find shortest paths which minimize linear combinations of two different metrics, such as travel time and energy cost. A mobile implementation that also considers traffic jams is presented in [62].

Graph Separators. When using graph separators [70, 72, 79, 118] during preprocessing, a multi-level partition of the graph is calculated to create a series of interconnected overlay graphs. A query starts at the lowest level and moves to higher levels as it progresses. Early implementations reached just modest speed-ups in comparison to other methods. However, those fast techniques (such as Contraction Hierarchies or Transit Node Routing) rely heavily on the strong inherent hierarchy of road networks with respect to travel time. They run much slower on metrics with less-pronounced hierarchies [16]. Preprocessing and query times of graph separators are essentially metric-independent. This is exploited by the authors of [35] who apply graph separators to develop a very robust and fast algorithm which supports arbitrary cost functions (travel distance, travel distance combined with travel time, travel time combined with cost on U-turns, etc.), real-time traffic data, and turn costs. They report query times of about a millisecond over continental road networks by carefully engineering the implementation of their algorithm and by applying contraction, goal-directed approaches, and parallelization.

Table Lookups. Another approach for speeding up shortest path queries is the use of table lookups of preprocessed distance data. The authors of [11, 12, 113] discuss *Transit Node Routing*. Here, a set of nodes (transit nodes) is first algorithmically determined. Transit nodes have the characteristic that many shortest paths connecting distant nodes pass through these nodes. These might be nodes representing locations of access points to fast motorways. Then the distances between each pair of transit nodes is precomputed and will be exploited during the shortest path query.

Advanced techniques. Several techniques exist which use a combination of bi-directional and goal directed search, and contraction. See [16] for an overview. The authors of [36,96,99] introduce *Core-ALT* for time-dependent networks. They apply contraction to produce a much smaller core-graph, consisting of *short-cuts*, which combine stretches of roads on the original graph. The search query performs most of the search on the core-graph. The algorithm is further accelerated by applying `ALT` and bi-directional search. Core-ALT is robust to updates of the graph with real time traffic information. To further accelerate the algorithm, approximation can be applied.

The algorithm SHARC [15,33] is a uni-directional search technique which combines Arc-Flags and Contraction. It is a very efficient technique even on large scale networks and can be applied to time-dependent networks. The multi-criteria scenario has been studied in [43].

The fastest algorithms on static road networks include combinations of Contraction Hierarchies with Arc-Flags (CHASE) and Transit Node Routing with Arc-Flags (TNR+AF). These algorithms answer random point-to-point shortest paths queries six orders of magnitude faster than `Dijkstra` [16]. To gain further insight as to why these techniques work so well on road networks, the authors of [3] conducted a theoretical analysis based on modeling road networks as graphs with low *highway dimensions*. Roughly speaking, these are graphs with a very small set of important nodes being part of all long shortest paths. Based on this analysis, the same authors developed the Hub-based Labeling algorithm (HL) [1]. During preprocessing, it uses Contraction Hierarchies to compute distance labels for every node toward important nodes (hubs) of the network which are then exploited by the shortest path algorithm. Labels must obey a *cover property*: for any two nodes r and t, there exists a node h on the shortest r-t path that belong to both labels (of r and t). An even better technique to calculate these labels is presented in [2]. It reduces average label size to surprising 69 (distances to nodes) for a graph of Western Europe with 18 million nodes. All these techniques are carefully engineered to reduce memory space and to improve preprocessing quality, and the authors claim that their algorithm is the fastest currently known in (static) road networks.

Routing on public transportation networks

An important characteristic of routing in public transportation networks is the modeling of timetable information. Two major approaches emerged: the *time-expanded* and the *time-dependent* approach. For both, queries are answered by applying a shortest path algorithm to a suitable constructed directed graph. In the time-expanded approach, nodes represent departure or arrival events at a station and arcs between nodes represent elementary connections between two events (i.e., served by a train or bus that does not stop along the way). Arc cost represents travel time. In the time-dependent approach, nodes represent stations and there is an arc between stations, if there exists at least one elementary connection between the stations. Arc costs are time-dependent and are modeled with piecewise linear functions. The authors of [107] give a detailed description of the two approaches, evaluate their performance, and present basic route planning algorithms. They conclude that the time-dependent approach allows for faster queries and smaller graph sizes, whereas the time-expanded approach is more robust for modeling more complex scenarios (like train-transfers). In this work, we will use the time-dependent approach. See Section 3.1.4 for more detailed information.

In general, because timetable information has to be respected, speeding-up shortest path queries on public transportation networks results more difficult than speeding-up shortest path queries on road networks. More specifically, speed-up techniques which proved to be very efficient on road networks did not yield the same results on public transportation networks [9, 10, 17, 38, 59, 107]. One major difference is the lack of hierarchy in the network, for example in large bus networks [9].

Note that in public transportation systems, multi-criteria optimization is very important. Not only travel time, but also cost, convenience, number of transfers, etc., are relevant. Pareto-optimal paths between two points can be calculated by using augmented versions of `Dijkstra` [107] but these versions increase runtime significantly and the application of acceleration techniques is even more difficult [22,52,95]. Another interesting related problem is how to incorporate uncertainty in travel times into the shortest path calculation and to maximise the reliability of a journee [67], e.g., minimise the probabilty to miss a conncetion.

A recent work [39] presents the algorithm RAPTOR. It is not `Dijkstra`-based and operates directly on the timetable by adopting dynamic programming. It does not rely on preprocessed data and thus can be easily extended to dynamic scenarios. When considering two criteria (arrival time and number of transfers), RAPTOR is a magnitude faster than previous `Dijkstra`-based approaches. The authors show how to add even more criteria, such as fare zones, and how to accelerate the algorithm by adopting parallelization.

The algorithm presented in [49] is also not `Dijkstra`-based. It organizes the network data as a single array of elementary connections, which the algorithm scans once per query. By using this simple data structure and by careful engineering, the authors are able to gain high spatial data locality which in turn results in very low runtimes. Furthermore, the authors present an extension of their algorithm which is able to handle multi-criteria queries.

4.2.1 Bi-directional search

Mono-directional search techniques start at the source node and move until the target node is found. Bi-directional search techniques start two searches, one starting at the source node and another starting at the target node. The bi-directional version of `Dijkstra` uses two mono-directional `Dijkstra` queries and builds two shortest path trees, one routed in the source node (forward search) and the other in the target node (backward search). As soon as a node v has status `settled` in both forward and backward queries the algorithm stops and the concatenation of the paths r-v and v-t is a shortest r-t-path. Mono-directional `Dijkstra` explores nodes circular centered in r with increasing radius until t is reached. Generally, the search space of the bi-directional variant is smaller as it explores nodes circular centered in the two nodes r and t until the two circles meet [17].

Time-dependency. Adapting bi-directional search to time-dependent networks has one major difficulty: The arrival time at the target node for the backward query is not known. However, the backward search can still assist the forward search in the following way. The forward query is started with starting time τ, while the backward query works with minimum weight arc costs $\underline{c}_{ij}$. As soon as the two searches meet, i.e., a node v is settled by both queries, a preliminary r-t-path p' is produced by concatenating the r-v-path obtained by the forward search with the v-t-path obtained by the backward search. The correct cost of p' is calculated by a re-evaluating of the v-t path in respect to arrival time τ at v calculated by the forward search. The cost $\gamma(p', \tau)$ provides an upper-bound on the cost of the optimal

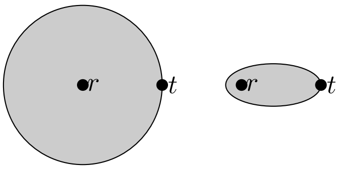


Figure 4.1: Schematic search space for the `Dijkstra` algorithm (left) and the `A`* algorithm (right).

path. Next, the two queries continue. As soon as the backward query settles a node with cost equal to or higher than $\gamma(p', \tau)$, it may stop. At this point it can be guaranteed that the shortest r-t-path is contained in the combined search spaces of the two queries. Now only the forward search continues but it only visits nodes settled by the backward search.

4.2.2 The `ALT` algorithm

The `ALT` algorithm [65,66] is a *goal directed search* as it preferably settles nodes that are close to the shortest path toward the target node. The `ALT` algorithm combines the characteristics of the `A`* algorithm, the use of *landmarks*, and the triangle inequality.

`A`*. The `A`* algorithm [68] is similar to `Dijkstra`. The difference lies in the order of selection of the next node v to be settled: A^* employs a key $k(v) = d(v) + \pi(v)$. At every iteration, the algorithm selects the node v with the smallest key $k(v)$. Ideally, the *potential function* π should *push* the search toward the target. In contrast, `Dijkstra` strictly explores all nodes in increasing distance from the source node r (see Figure 4.1).

Potential function. The potential function π is a function on nodes $\pi : V \to \mathbb{R}$. The *reduced cost* of an arc is defined by $c^{\pi}_{vw} = c_{vw} - \pi(v) + \pi(w)$. We denote the graph which uses the reduced cost on arcs as G_π. On G_π the length of any path $P = [v_1, \ldots, v_k]$ changes to $c_\pi(P) = c(P) - \pi(v_1) + \pi(v_k)$. In [74], it is shown that running A^* on G is equivalent to running `Dijkstra` on G_π. `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 $v, w \in V$. The following characteristics are important for the algorithms in this work:

Lemma 4.2.1 ([65]). *If π is feasible and for a node $t \in V$ we have $\pi(t) \leq 0$, then $\pi(v) \leq d(v,t), \forall v \in V$.*

Lemma 4.2.2 ([65]). *If π_1 and π_2 are feasible potential functions, then $\max(\pi_1, \pi_2)$ is a feasible potential function.*

The first lemma implies that a feasible potential function can be interpreted as a lower bound on the distance from v to t, if the potential of the target node is zero. The second lemma states that two feasible lower bound functions can be combined.

In the case where $\pi(v)$ gives an exact estimate, A^* only scans nodes on shortest paths to t and in general, the closer $\pi(v)$ is to the actual remaining distance, the faster the algorithm will find the target.

Landmarks and triangle inequality. On a road network, the Euclidean distance or air distance from node v to node t can be used to compute $\pi(v)$. A significant improvement

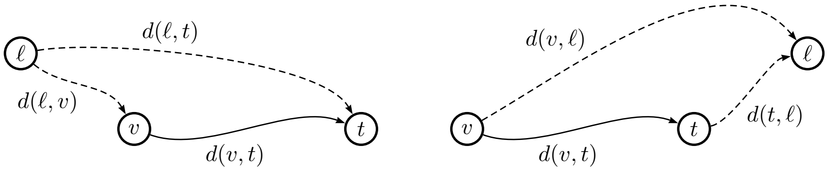


Figure 4.2: Landmark distances.

can be achieved by using landmarks and the triangle inequality [65]. The main idea is to select a small set of nodes, called landmarks, $\ell \in \mathcal{L} \subset V$, spread appropriately over the network, and precompute all distances of shortest paths $d(\ell, v)$ and $d(v, \ell)$ between these nodes and any other node, 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 4.2).

$$\pi(v) = \max_{\ell \in \mathcal{L}} (d(v, \ell) - d(t, \ell), d(\ell, t) - d(\ell, v)) \tag{4.1}$$

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 `ALT`.

Landmark selection. A crucial point of the algorithm is the quality of landmarks. A good landmark should yield constantly high lower bounds for as many queries and for as many nodes possible. Several heuristics have been evaluated [41, 65, 66]. The most used heuristics are Avoid [65] and MaxCover [66]. The quality of landmarks produced by MaxCover is better than the quality of landmarks produced by Avoid, but runtime of MaxCover are too high for larger graphs.

Query. The difference of `ALT` with respect to `Dijkstra` is that it employs a key $k(v) = d(v) + \pi(v)$. Note that calculating Equation 4.1 with respect to all landmarks $\ell \in \mathcal{L}$ is not efficient. For that reason, the potential function is usually calculated on a subset $\mathcal{L}_{\texttt{active}} \subset \mathcal{L}$ of *active* landmarks. A good strategy is to start with the two landmarks which yield the highest potentials for r at the beginning of the query and to add new landmarks at every k iterations of the algorithm, if non-active landmarks yield better potentials for the currently settled nodes than the active landmarks. Whenever the set of active landmarks changes, all keys in the priority queue have to be updated. The authors of [65, 66] propose a unidirectional (`uniALT`) and bi-directional variant of `ALT`. We will treat bi-directional `ALT` in more detail in Section 5.4.

Time-dependency. 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* $\underline{c}_{ij} = \min_{\tau \in \mathcal{T}} c_{ij}(\tau)$. In this case, the values of the potential function represent a lower bound of a lower bound. The quality of the landmark distances depends on the gap between the lower and upper bound of the arc cost functions. Typically, `ALT` on time-dependent networks does not perform as well as on time-independent networks. See [98] for efficient implementations of `uniALT` and `ALT` as well as experimental data on continental size road networks which include traffic information.

Dynamic networks. As observed in [17,44], potentials stay feasible as long as arc weights only increase and do not drop below a minimal value. Therefore, ALT can be applied to dynamic networks, such as road networks which are periodically updated with real time traffic information [42,98], without the need to re-run the calculation of landmark distances.

4.3 Multi-Modal Routing

The goal of multi-modal routing is to provide information about the best way to reach a destination by considering all available modes of transportation in a multi-modal transportation network. These are car, rental car, public transportation, bicycle, rental bicycle, taxis, walking, etc. Other than minimizing travel time, shortest paths in such networks must typically satisfy some additional constraints. First of all, feasibility has to be assured: private cars or bicycles can only be used when they are available. Second, passenger preferences should be respected: passengers may want to exclude some modes of transportation, e.g., the bicycle when it is raining or the car at moments of heavy traffic. Furthermore, they may want to limit the number of changes when using different modes of transportation or may not want to use toll roads. Such customization is an important characteristic of multi-modal routing services.

The regular language constrained shortest path problem (RegLCSP) can be applied to this kind of problem. It uses regular languages to model constraints on paths. 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. RegLCSP can be solved by a generalization of Dijkstra which we will call $\mathtt{D}_{\mathsf{RegLC}}$. The adaption of RegLCSP to multi-modal transportation networks has successfully been shown in various studies [7,69,121,122]. The expressibility of regular languages proved to be sufficient for modeling most reasonable path constraints, which might arise when searching valid and customized shortest paths on multi-modal networks. A difficulty when applying RegLCSP is that runtimes of $\mathtt{D}_{\mathsf{RegLC}}$ on large graphs (like Dijkstra) are too high for real-time applications. Furthermore, because of the labeled graph and the regular language, it is not straight-forward to apply known speed-up techniques which work well for Dijkstra to $\mathtt{D}_{\mathsf{RegLC}}$. In addition, as discussed in the previous section, different techniques have to be applied when speeding-up Dijkstra on road networks or public transportation networks. $\mathtt{D}_{\mathsf{RegLC}}$ works on a multi-modal network, which includes both roads and public transportation.

In order to cut runtime, some recent works on multi-modal routing isolate the public transportation network from road networks so that they can be treated individually or do not use a labeled graph and limit a priori the range of allowed types of paths [34,37,50]. In this way, they are able to substantially accelerate runtime but lose much of the flexibility which the use of regular language as a means to constrain the paths offers. Some attempts to accelerate $\mathtt{D}_{\mathsf{RegLC}}$ have been made in [69]. One of the major contributions of this thesis is the algorithm SDALT which solves RegLCSP but runs considerably faster then $\mathtt{D}_{\mathsf{RegLC}}$. In this section, we will discuss RegLCSP and $\mathtt{D}_{\mathsf{RegLC}}$ in detail. SDALT will be presented in Chapter 5.

Related Work

Early works on the use of regular languages as a model for constrained shortest path problems include [90,110,130], with applications to database queries and web searches. A finite state automaton is used in [86] to model path constraints (called path viability) on a multi-modal transportation network for the bi-criteria multi-modal shortest path problem.

The authors of [8] present a systematic theoretical study of formal language constrained shortest path variants with respect to problem complexity for different classes of languages. 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 ($\mathtt{D}_{\mathsf{RegLC}}$) to solve RegLCSP. Also context-free languages are shown to permit polynomial algorithms. Moreover, the paper provides a collection of problems in transportation science that can be handled by using formal languages, such as finding alternative paths, handling turn-penalties, and multi criteria shortest paths. Experimental data on efficient implementations of $\mathtt{D}_{\mathsf{RegLC}}$ on multi-modal networks including time-dependent arc costs can be found in [7,121,122]; [122] introduces turn penalties. In [69], various speed-up techniques and their combinations including bi-directional and goal-directed search have been applied to $\mathtt{D}_{\mathsf{RegLC}}$ on rail and road networks. The performance of the algorithm depends on the network properties and on the restrictivity of the regular language.

The authors of [5] apply bi-directional search and $\mathtt{A}^*$ to speed-up a bi-criteria algorithm which minimizes travel time and the number of modal transfers. The authors of [109] use contraction on a large 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 $\mathtt{D}_{\mathsf{RegLC}}$.

An advantage of using regular languages is its flexibility: it is quite simple to forbid unfeasible patterns of paths, e.g., private bicycle followed by metro followed by private bicycle, 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 known speed-up techniques to $\mathtt{D}_{\mathsf{RegLC}}$. 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 [37,50].

The authors of [37] 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 [50], 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 [34] apply a recent fast algorithm [39] to public transportation and contraction to the walking network. Their algorithms are able to compute full Pareto sets of multi-modal shortest paths, optimizing arrival, trip, and walking time, as well as trip cost. Significant journeys are identified by applying techniques from fuzzy logic. The authors present experimental data on a large metropolitan area including walking, taxi, rental bicycle, and public transportation. Runtimes are fast enough for practical applications.

4.3.1 Regular language constrained shortest path problem

The objective of the formal language constrained shortest path problem [8] is to find a path on a labeled graph which minimizes some cost function (distance, time, etc.) and which respects constraints on the word produced by concatenating the labels on the arcs along the shortest path. The constraints are given by a formal language and the word must form an element of that language. In [8], a systematic theoretical study of the formal language constrained shortest path problem can be found. The authors provide algorithmic and complexity-theoretical results on the use of various types of languages. When *regular languages* are used to model the constraints then the problem is called regular language constrained shortest path problem, which we will denote by `RegLCSP`.

Definition 4.3.1 (Regular language constrained shortest path problem (`RegLCSP`))**.** *Given an alphabet Σ, a regular language $L_0 \in \Sigma^*$, a directed, labeled graph $G = (V, A, \Sigma)$, a source node $r \in V$ and a target node $t \in V$, find a shortest path from r to t, where the sequence of labels along the arcs of the path forms a word $w \in L_0$ (`Word(p)` $\in L_0$).*

Note that in this work, we consider the time-dependent variant of `RegLCSP`: we are looking for shortest r-t-paths constrained by a regular language L_0 with departure time $\tau_0 \in T$ (see Definition 2.3.2). Note that any shortest r-t-path subject to some restriction is at least as long as a shortest unrestricted r-t-path.

4.3.2 Algorithm to solve `RegLCSP`

To efficiently solve `RegLCSP`, a generalization of Dijkstra's algorithm has been proposed by [8]. We will call this algorithm $\mathtt{D}_{\mathtt{RegLC}}$. It works on a *product graph* $G^\times = G \times \mathcal{A}$. The product graph is constructed as follows.

Definition 4.3.2 (Product graph)**.** *Given a labeled directed graph $G = (V, A, \Sigma)$, and a non deterministic finite automaton $\mathcal{A} = (S, \Sigma, \tau, s_0, F)$, the product graph is $G^\times = G \times \mathcal{A} = (V^\times, A^\times)$ where*

- *the node set is $V^\times = \{v^\times = (v, s) | v \in V, s \in S\}$*
- *and the arc set is $A^\times$. An arc $((v, s)(w, s')) \in A^\times$ exists for $(v, s) \in V^\times$, $(w, s') \in V^\times$ if there is an arc $(v, w, l) \in A$ and a transition such that $s' \in \delta(l, s)$. The cost and label of an arc $((v, s)(w, s')) \in A^\times$ corresponds to the cost and label of $(v, w, l) \in A$.*

The `RegLCSP` can be solved in deterministic polynomial time because there is a one-to-one correspondence between constrained r-t-paths in G and shortest paths in $G^\times$ (see Algorithm 1).

Theorem 4.3.3 ([8])**.** *Finding a shortest r-t-path constrained by language L for some $L \subseteq \Sigma^*$, $r \in V$, and $t \in V$ is equivalent to finding a shortest path in the product graph $G^\times = G \times \mathcal{A}$ from node (r, s_0) and ending at node (t, s_f) for some $s_f \in F_0$ in $G^\times$.*

Algorithm 1 [8]

Input: labeled graph $G = (V, A, \Sigma)$, source r, target t, regular language $L_0 \subseteq \Sigma^*$
Output: constrained shortest path

```
1 construct a NFA $\mathcal{A}_0 = (S, \Sigma, \delta, s_0, F)$ from $L_0$
2 construct the product graph $G^\times = G \times \mathcal{A}_0$
3 starting from product node $(r, s_0)$, find a shortest path to all product nodes $(t, s_f)$ with
  $s_f \in F$.
4 from the resulting paths pick the path with minimal cost
```

The output of the algorithm will be a sequence of nodes:

$$p^\times = [(r, s_0), (v_i, s_i), \dots, (t, s_f)], s_f \in F \tag{4.2}$$

The output path p on G can be produced by maintaining the same sequence of nodes of $p^\times$ but omitting the states, $p = (r, v_i, \dots, t)$. The function `Word(p)` returns the sequence of labels along a path p. `Word(p)` will be an element of L_0.

Complexity. It can be seen that step 3 dominates the other steps. The size of $G^\times$ is $O(|\mathcal{A}||G|)$ and so step 3 is in $O(T(|\mathcal{A}||G|))$ where $T(n)$ is the runtime of a shortest path algorithm on a graph with n nodes.

Implicit computation of the product graph. The size of memory required to hold the product graph $G^\times$ is in $O(T(|\mathcal{A}||G|))$. This might result in being too large for some applications, especially when working with large networks. Therefore, the authors of [7] propose a modification to avoid the explicit calculation of the product graph in advance: the algorithm works on an *implicit* product graph by generating all the neighbors which have to be explored only when necessary. Note that in the worst case scenario, the algorithm may still have to visit all the nodes of the product graph and so the entire product graph will be produced. This should rarely be the case.

Time-dependency. The algorithm can easily be adapted to the time-dependent scenario as shown in [7], see Algorithm 2.

Our implementation. Algorithm 2 shows the pseudo code for our implementation of $\mathsf{D_{RegLC}}$. It produces the product graph implicitly (line 10) and works with time-dependent arc costs (line 11). It is the basis of the algorithm `SDALT` which we will introduce in Chapter 5.

Speed-up techniques for $\mathsf{D_{RegLC}}$

In [6, 69], various speed-up techniques and their combinations including bi-directional and goal-directed search have been applied to the time-independent version of $\mathsf{D_{RegLC}}$ on rail and road networks. The authors use average costs (average travel times). The performance of the algorithm depends on the network properties and on how restrictive the regular language is. The authors of [5] apply bi-directional search and `A*` to speed-up a bi-criteria algorithm which minimizes travel time and the number of modal transfers.

Algorithm 2 Algorithm $\mathsf{D_{RegLC}}$ to solve $\mathtt{TDRegLCSP}$

```
Input: labeled graph $G = (V, A, \Sigma)$, source $r$, target $t$, regular language $L_0 \subseteq \Sigma^*$ represented as automaton $\mathcal{A}_0$, start time $\tau_{start}$
function $\mathsf{D_{RegLC}}(G, r, t, \tau_{start}, \mathcal{A}_0)$
    $d(v,s) \leftarrow \infty$, $p(v,s) \leftarrow -1$, $\forall (v,s) \in V \times S$
    pathFound $\leftarrow$ false, $d(r,s_0) \leftarrow 0$, $k(r,s_0) \leftarrow 0$, $p(r,s_0) \leftarrow -1$
    insert $(r,s_0)$ in priority queue $Q$
    while $Q$ is not empty do
        extract $(v,s)$ with smallest key $k$ from $Q$
        if $v == t$ and $s \in F_0$ then
            pathFound $\leftarrow$ true
            break
        for each $(w,s')$ s.t. $(v,w,l) \in \mathcal{A}_0 \wedge s' \in \delta(l,s)$ do
            $d_{tmp} \leftarrow d(v,s) + c_{vwl}(\tau_{start} + d(v,s))$                ▷ time-dependency
            if $d_{tmp} < d(w,s')$ then
                $p(w,s') \leftarrow (v,s)$
                $d(w,s') \leftarrow d_{tmp}$
                $k(w,s') \leftarrow d(w,s')$
                if $(w,s')$ not in $Q$ then                                              ▷ insert
                    insert $(w,s')$ in $Q$
                else                                                                     ▷ decrease
                    decreaseKey $(w,s')$ in $Q$
```

$\mathtt{A^*}$**.** When adapting $\mathtt{A^*}$ [68] to $\mathsf{D_{RegLC}}$, the calculation of the key changes to $k(v,s) = d(v,s) + \pi(v,s)$. The choice of the potential function depends on the type of arc cost. If a distance metric is used, the Euclidean distance $\underline{dist}_{ij}$ from node i to node j represents a lower bound on the length of the shortest i-j-path. Hence, $\pi(v) := \underline{dist}_{vt}$ yields a feasible potential function. For travel time metrics, the potential function $\pi(v) := \frac{\underline{dist}_{vt}}{v_{max}}$ can be used, where $v_{max} = \max_{(i,j)\in E} \frac{\underline{dist}_{ij}}{c_{ij}}$ can be seen as the maximum speed in the graph.

Sedgewick-Vitter Heuristic. If exact shortest paths are not essential, a canonical extension of the $\mathtt{A^*}$ is to enforce the potential function: the Sedgewick-Vitter heuristic [119] uses a modified key:

$$k(v,s) = d(v,s) + \alpha\pi(v,s), \alpha \geq 1 \tag{4.3}$$

The parameter α determines the trade-off between speed-up of runtime and path length increase: the greater α the narrower the search space. However, some nodes essential to the shortest r-t-path may not be visited. See [76] for another study of this heuristic.

$\mathtt{ALT}$**.** The adaption of $\mathtt{ALT}$ to $\mathsf{D_{RegLC}}$ is straightforward. On a labeled graph G for every language L and every r-t-path P constrained by L the following holds:

$$c(P) \geq c(P_{free}) \tag{4.4}$$

where P_{free} is a non restricted r-t-path, i.e., it may have any sequence of labels, or r-t-path on the un-labeled graph G. In other words, applying constraints to shortest paths never yields shorter paths. The landmark distances calculated on the un-labeled graph G remain valid on the labeled graph. However, on a time-dependent multi-modal graph the lower bounds now represent lower bounds on travel times *and* on modes of transportation.

Bi-directional search. Two queries of $\mathtt{D}_{\mathsf{RegLC}}$ are started simultaneously, one forward query starting at source node (r, s_0) and one backward query, starting from all final nodes $(t, s_f), s_f \in F_0$. For static graphs, the algorithm may stop as soon as a node is settled by both searches. For the time-dependent scenario, the application of bi-directional search is more complicated. We will discuss bi-directional search in more detail in Section 5.4. Note that bi-directional search can be combined with A*, with the Sedgewick-Vitter Heuristic, as well as with ALT.

4.4 Summary

In this chapter, we introduced routing techniques on road and public transportation networks, as well as on multi-modal transportation networks. We discussed in detail the regular language constrained shortest path problem and the algorithm $\mathtt{D}_{\mathsf{RegLC}}$, which can be used to find multi-modal shortest paths. In the next chapter, we will present the algorithm SDALT which runs considerably faster than $\mathtt{D}_{\mathsf{RegLC}}$ in many scenarios. In Chapter 6, we will use $\mathtt{D}_{\mathsf{RegLC}}$ and SDALT to solve the 2-way multi-modal shortest path problem.

Chapter 5

SDALT

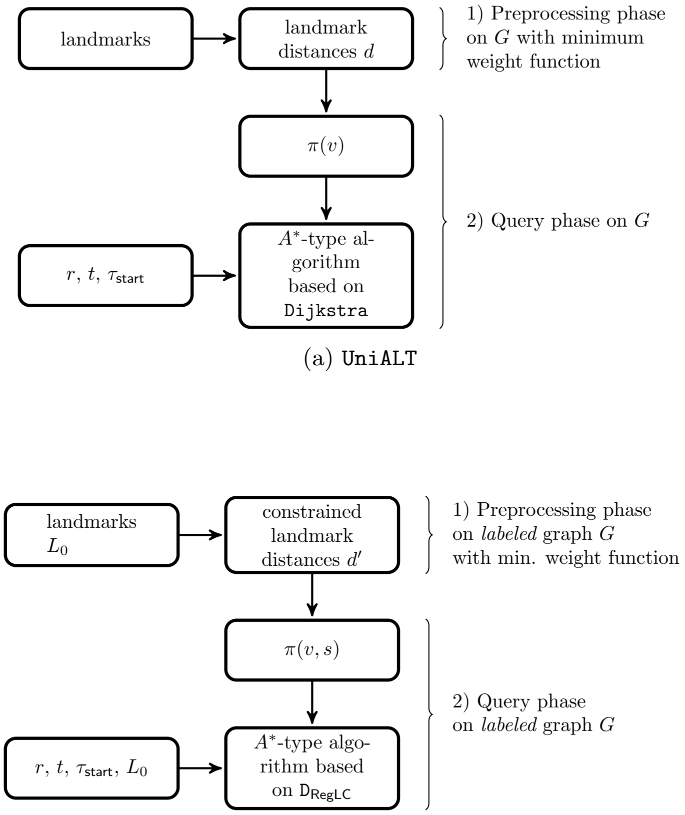
5.1 State Dependent ALT: `SDALT`

To speed up $\mathsf{D}_{\mathsf{RegLC}}$, [6] employs among other techniques goal directed search (`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 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 $\mathsf{D}_{\mathsf{RegLC}}$ 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 5.1). The main differences consist in the way landmark distances are calculated and on `SDALT` being based on $\mathsf{D}_{\mathsf{RegLC}}$ and not on `Dijkstra`. Potentials depend on the pair (v, s).

5.1.1 Query phase

The query phase deploys a $\mathsf{D}_{\mathsf{RegLC}}$ algorithm enhanced by the characteristics of the `ALT` algorithm. As priority queue Q we use a binary heap. The pseudo code in Algorithm 3 works as follows: the algorithm maintains, for every visited node (v, s) in the product graph $G^{\times}$, a tentative distance label $d(v, s)$ and a parent node $p(v, s)$. It starts by computing the key $k(r, s_0) = \pi(r, s_0)$ for the start 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 $s' \in \delta(l, s)$. More precisely,


landmarks
landmark distances d
1) Preprocessing phase on G with minimum weight function
$\pi(v)$
2) Query phase on G
r, t, τ_{start}
A^*-type algorithm based on Dijkstra
landmarks L_0
constrained landmark distances d'
1) Preprocessing phase on *labeled* graph G with min. weight function
$\pi(v, s)$
2) Query phase on *labeled* graph G
r, t, τ_{start}, L_0
A^*-type algorithm based on $\mathtt{D}_{\text{RegLC}}$

(a) `UniALT`

(b) `SDALT`

Figure 5.1: Comparison `uniALT` and `SDALT`

a new temporary distance label $d_{\mathsf{tmp}} = d(v,s) + c_{vwl}(\tau_{start} + d(v,s))$ is compared to the currently assigned tentative distance label (line 10). If it is smaller, it either calculates the key $k(w,s') = \pi(r,s_0) + d_{\mathsf{tmp}}$ 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,l) might be time-dependent and thus has to be evaluated for time $\tau_{start} + d(v,s)$. 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 3 Pseudo-code SDALT

```
Input: labeled graph $G = (V, A, \Sigma)$, source $r$, target $t$, start time $\tau_{\mathsf{start}}$, regular language
    $L_0 \subseteq \Sigma^*$ represented as automaton $\mathcal{A}_0$
 1 function SDALT$(G, r, t, \tau_{start}, L_0)$
 2     $d(v,s) \leftarrow \infty$, $p(v,s) \leftarrow -1$, $\pi_{v,s} \leftarrow 0$, $\forall (v,s) \in V \times S$
 3     pathFound ← false, $d(r,s_0) \leftarrow 0$, $k(r,s_0) \leftarrow \pi(r,s_0)$, $p(r,s_0) \leftarrow -1$
 4     insert $(r,s_0)$ in priority queue $Q$
 5     while $Q$ is not empty do
 6         extract $(v,s)$ with smallest key $k$ from $Q$
 7         if $v == t$ and $s \in F_0$ then
 8             pathFound ← true
 9             break
10         for each $(w,s')$ s.t. $(v,w,l) \in \mathcal{A}_0 \wedge s' \in \delta(l,s)$ do
11             $d_{tmp} \leftarrow d(v,s) + c_{vwl}(\tau_{start} + d(v,s))$                ▷ time-dependency
12             if $d_{tmp} < d(w,s')$ then
13                 $p(w,s') \leftarrow (v,s)$
14                 $d(w,s') \leftarrow d_{tmp}$
15                 if $(w,s')$ not in $Q$ then                                    ▷ insert
16                     $\pi_{w,s'} \leftarrow \pi_(w,s')$
17                     $k(w,s') \leftarrow d(w,s') + \pi_{w,s'}$
18                     insert $(w,s')$ in $Q$
19                 else                                                          ▷ decrease
20                     $k(w,s') \leftarrow d(w,s') + \pi_{w,s'}$
21                     decreaseKey $(w,s')$ in $Q$
```

5.1.2 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 [65] (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 will derive from L_0. We refer to the travel time of the shortest path from (i,s) to (j,s'), for some $s' \in F$, which is *constrained* by the regular language $L_s^{i \to j}$, as *constrained distances* $d'_s(i,j)$ and to the distances calculated during the preprocessing phase 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 $s' \in F$.

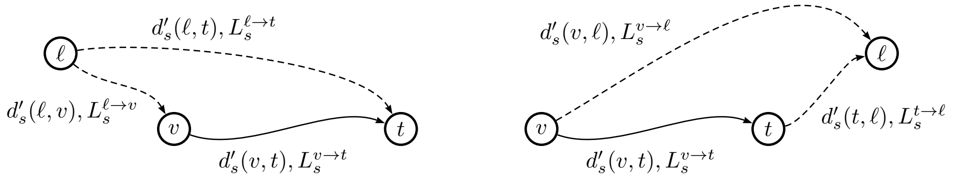

ℓ
v
t
$d'_s(\ell,t), L_s^{\ell\to t}$
$d'_s(\ell,v), L_s^{\ell\to v}$
$d'_s(v,t), L_s^{v\to t}$
$d'_s(v,\ell), L_s^{v\to \ell}$
ℓ
v
t
$d'_s(t,\ell), L_s^{t\to \ell}$
$d'_s(v,t), L_s^{v\to t}$

Figure 5.2: Landmark distances for SDALT, $L_s^{i\to j}$ represents the regular language which constrains the shortest paths from (i,s) to (j,s'), $s'\in F$.

The constrained landmark distances are used to calculate the potential function $\pi(v,s)$, and to provide a lower bound on the distance $d'_s(v,t)$:

$$\pi(v,s) = \max_{\ell\in\mathcal{L}}(d'_s(\ell,t) - d'_s(\ell,v), d'_s(v,\ell) - d'_s(t,\ell)) \tag{5.1}$$

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 $L_s^{\ell\to t}$, $L_s^{\ell\to v}$, $L_s^{v\to \ell}$, $L_s^{t\to \ell}$ 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 5.2).

5.1.3 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 $L_s^{\ell\to t}$, $L_s^{\ell\to v}$, $L_s^{v\to \ell}$, $L_s^{t\to \ell}$ used to constrain the calculation of $d'_s(\ell,t)$, $d'_s(\ell,v)$, $d'_s(v,\ell)$, $d'_s(t,\ell)$ in order that $d'_s(\ell,t) - d'_s(\ell,v)$, $d'_s(v,\ell) - d'_s(t,\ell)$ are valid lower bounds for $d'_s(v,t)$ (see Figure 5.2 and Equation 5.1). A first answer gives Proposition 5.1.1:

Proposition 5.1.1. *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. $d'_s(\ell,t) - d'_s(\ell,v) > d'_s(v,t)$ or $d'_s(\ell,v) + d'_s(v,t) < d'_s(\ell,t)$ means that the word $w_1\circ w_2\notin L_s^{\ell\to t}$, as $d'_s(\ell,t)$ is a cost of a shortest path. But this means $L_s^{\ell\to v}\circ L_s^{v\to t}\nsubseteq L_s^{\ell\to t}$. (ii) The same can be proven in a similar way for $d'_s(v,\ell) - d'_s(t,\ell)$. □

Proposition 5.1.1 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 will 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 5.1.1, in order to gain valid distance bounds for a generic node (v,s) of $G^\times$ (see also Table 5.1):

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

Table 5.1: 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 (n, 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 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.

Procedure 3 produces four distinct languages for a node (v, s) of G^x. To compute the bound $d'_s(l, t) - d'_s(l, v)$ the distance calculation of $d'_s(l, t)$ is limited by *all* constraints of $\mathcal{A}_0$, i.e., it will be constrained by $\mathcal{A}_0$, and that of $d'_s(l, v)$ is constrained by the part of the constraints on $\mathcal{A}_0$ occurring *before* state s. Similar, to compute the bound $d'_s(v, l) - d'_s(t, l)$, the distance calculation of $d'_s(v, l)$ is limited by all constraints on $\mathcal{A}_0$ occurring *after* state s, and that of $d'_s(t, l)$ 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.

The following Proposition 5.1.2 gives indications on when Procedure 1 will generally produce better bounds than ALT.

Proposition 5.1.2. *Given a labeled graph $G = (V, A_1 \cup A_2, \Sigma)$ 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 exclusively allowing arcs with label ℓ_1, $L_0 = \{\ell_1^*\}$, bounds calculated by using Procedure 1 are at least a factor α greater than bounds calculated using `ALT`.

Proof. We have $\Sigma = \{\ell_1, \ell_2\}$. For `ALT` the landmark distances calculation is not constrained which is equal to constraining the landmark distance calculation by $L_{\mathsf{ALT}} = \{(\ell_1|\ell_1)^*\}$. For Procedure 1 the language $L_{\mathsf{P1}} = \{\ell_1^*\}$ is used to constrain the landmark distance calculation. We gain bounds $b_{\mathsf{ALT}} = d'_{s,\mathsf{ALT}}(\ell, t) - d'_{s,\mathsf{ALT}}(\ell, v)$ and $b_{\mathsf{SDALT}} = d'_{s,\mathsf{P1}}(\ell, t) - d'_{s,\mathsf{P1}}(\ell, v)$. As landmark distances are shortest paths $b_{\mathsf{SDALT}} > \alpha \leq b_{\mathsf{ALT}}$, as shortest paths by considering only arcs A_1 are at least a factor α greater than when considering arcs $A_1 \cup A_2$. □

With reference to a multi-modal transportation network, Proposition 5.1.2 states that for a RegLCSP where some *fast* modes of transportation are excluded, Procedure 1 produces better bounds than `ALT`. Note, that Procedure 2 yields better bounds than Procedure 1 on a graph where modes of transportation hierarchically dominate each other (car over trains over biking over walking) and which are excluded in decreasing order of speed by L_0. Procedure 3 on the other hand works for those instances of RegLCSP, which not only totally or partially exclude arcs with certain labels, which is the case for Procedures 1 and 2, but which impose specific conditions on the use of arcs with certain labels which are likely to inflict a major detour from the unconstrained shortest path. These can be, for example, obligations on the visit of arcs with infrequent labels, limitations on the use of public transportation to only one ride, etc. See Proposition 5.1.3.

Proposition 5.1.3. *Given a labeled graph $G = (V, A_1 \cup A_2, \Sigma)$. The arcs of A_1 are assigned label l_i and the arcs of A_2 are assigned l_2. For every arc (i, j, l_2), there is a parallel arc (i, j, l_1) with $c_{ijl_1} \leq c_{ijl_2}$. For a `RegLCSP` with L_0 on G which imposes that a label l_2 has to be visited at least once, bounds calculated by Procedure 3 are better than bounds calculated by Procedure 1 or Procedure 2.*

Proof. With reference to Figure 5.3d, let us suppose that there is only one arc (v', v'', l_2) which has label l_2 and is close to node v and landmark ℓ for which we will examine the bounds. This means that a shortest path from (v, s_0) to the target satisfying L_0 (solid line) will necessarily have to include arc (v', v'', l_2). By applying Procedure 1, the landmark distances $d'_{s,\mathsf{P1}}(\ell, v)$ and $d'_{s,P1}(\ell, t)$ are both constrained by the automaton in Figure 5.3c. Both distances are shown as dotted lines. By applying Procedure 3, as stated before, the automaton in Figure 5.3a is used for the constrained landmark distance calculation of $d'_{s,\mathsf{P3}}(\ell, t)$ and automaton in Figure 5.3b for $d'_{s,\mathsf{P3}}(\ell, v)$ (dashed lines). This yields $d'_{s,\mathsf{P1}}(\ell, t) - d'_{s,\mathsf{P1}}(\ell, v) \leq d'_{s,\mathsf{P3}}(\ell, t) - d'_{s,\mathsf{P3}}(\ell, v)$ as $d'_{s,\mathsf{P1}}(\ell, t) = d'_{s,\mathsf{P3}}(\ell, t)$ and $d'_{s,\mathsf{P1}}(\ell, v) \leq d'_{s,\mathsf{P3}}(\ell, v)$. It follows that Procedure 3 produces better bounds for node (v, s) than Procedure 1. □

Time-dependency

Similar to `Dijkstra`, $\mathsf{D}_{\mathsf{RegLC}}$ and also `SDALT` can easily be adapted to the time-dependent scenario by selecting landmarks and calculating landmark distances by using the *minimum weight cost function* $\underline{c_{ijl}} = \min_{\tau \in \mathcal{T}} c_{ijl}(\tau)$. Note that in a dynamic scenario, potentials stay valid as long as arc weights only increase and do not drop below a minimal value [7, 44].

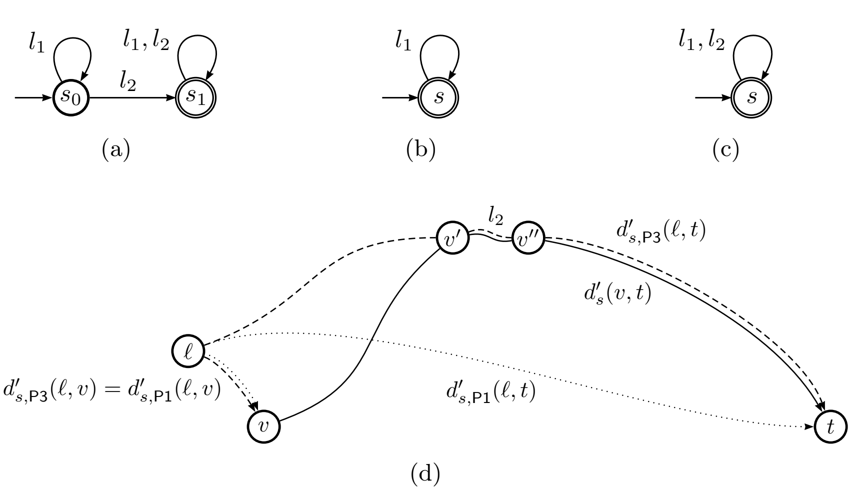

l_1
l_1, l_2
l_2
s_0
s_1
(a)
l_1
s
(b)
l_1, l_2
s
(c)
l_2
v'
v''
$d'_{s,\mathsf{P3}}(\ell, t)$
$d'_s(v, t)$
ℓ
$d'_{s,\mathsf{P3}}(\ell, v) = d'_{s,\mathsf{P1}}(\ell, v)$
$d'_{s,\mathsf{P1}}(\ell, t)$
v
t
(d)

Figure 5.3: Example for Procedure 3. Automaton 5.3a is used to constrain $d'_{s,\mathsf{P1}}(v, t)$ and $d'_{s,\mathsf{P3}}(\ell, t)$, automaton 5.3b is used to constrain the calculation of $d'_{s,\mathsf{P1}}(\ell, b)$ and $d'_{s,\mathsf{P3}}(\ell, b)$, and automaton 5.3c is used to constrain $d'_{s,\mathsf{P1}}(v, t)$.

5.2 Label Setting SDALT: lsSDALT

One condition that the A* and the ALT algorithm work correctly is that reduced costs are positive, i.e., the potential function is feasible (see Section 4.2.2). 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.

5.2.1 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 5.2.1.

Proposition 5.2.1. *By using the regular languages produced by applying Procedure 3 (see Table 5.1) for the constrained landmark distance calculation for all nodes (v, s), the potential function $\pi(v, s)$ in Equation 5.1 is feasible.*

Proof. If $\pi(v, s)$ is feasible, then the reduced cost c^{π}_{ijl} 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 (f, g, l), let us suppose $\pi(v, s)$ is not feasible and that the reduced cost is $c_{fgl}(\tau) - \pi(f, s_f) + \pi(g, s_g) < 0$. We have that $c_{fgl}(\tau) + (d'_{s_g}(\ell, t) - d'_{s_g}(\ell, g)) < (d'_{s_f}(\ell, t) - d'_{s_f}(\ell, f))$. Let us consider two cases.

(case 1) If $s_f = s_g = s$, then $c_{fgl}(\tau) + d'_s(\ell, f) < d'_s(\ell, g)$. But as $d'_s(\ell, g)$ is a shortest path and $s \in \delta(l, s)$, this is a contradiction.

(case 2) If $s_g \neq s_f$ then as for (3b), $\mathcal{A}^{\ell \to t}_{s_f}$ includes $\mathcal{A}^{\ell \to t}_{s_g}$ we have $d'_{s_f}(\ell, t) \leq d'_{s_g}(\ell, t)$. So we have that $c_{fgl}(\tau) + d'_{s_f}(\ell, f) < d'_{s_g}(\ell, g)$. But as, for rules (3a), $\mathcal{A}^{\ell \to g}_{s_g}$ includes all states and transitions of $\mathcal{A}^{\ell \to f}_{s_f}$ 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 $a = ((f, s_f)(g, s_g), l)$ let us suppose $\pi(v, s)$ is not feasible and that $c_{fgl}(\tau) - \pi(f, s_f) + \pi(g, s_g) < 0$. We have that $c_{fgl}(\tau) + (d'_{s_g}(g, \ell) - d'_{s_g}(t, \ell)) < (d'_{s_f}(f, \ell) - d'_{s_f}(t, \ell))$. Let us consider two cases.

(case 1) If $s_f = s_g = s$, then $c_{fgl}(\tau) + d'_s(g, \ell) < d'_s(f, \ell)$. But as $d'_s(f, \ell)$ is a shortest path and $s \in \delta(l, s)$, this is a contradiction.

(case 2) If $s_g \neq s_f$ then as for 3c and 3d, $\mathcal{A}_{s_f}^{t\to\ell}$ is included in $\mathcal{A}_{s_g}^{t\to\ell}$ we have $d'_{s_f}(t,\ell) \geq d'_{s_g}(t,\ell)$. Thus $c_{fgl}(\tau) + d'_{s_g}(\ell, g) < d'_{s_f}(\ell, f)$. But as, for (3c), $\mathcal{A}_{s_f}^{f\to\ell}$ includes all states and transitions of $\mathcal{A}_{s_g}^{g\to\ell}$ 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 5.6. 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 $\mathsf{D}_{\mathsf{RegLC}}$.

5.2.2 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 5.2.2. *If solutions exist, `lsSDALT` finds a shortest path.*

5.2.3 Complexity and memory requirements

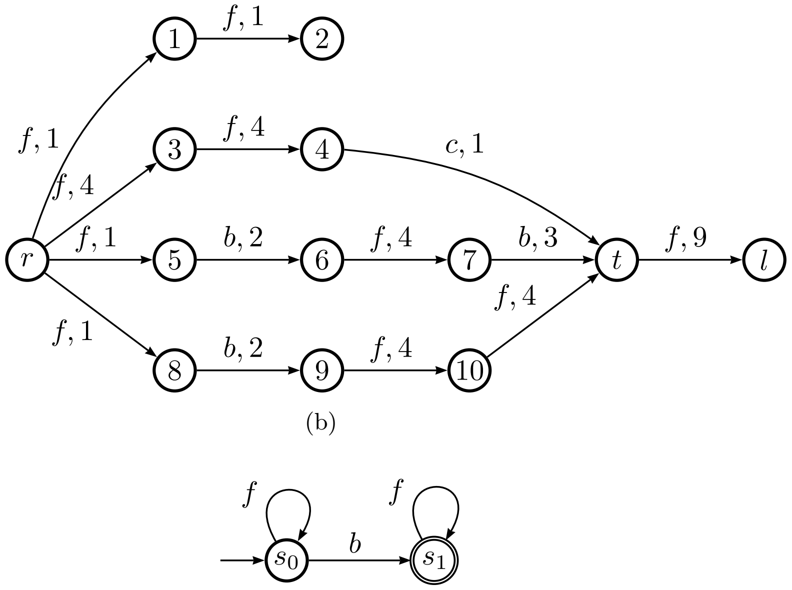
Complexity of `lsSDALT` is equal to the complexity of $\mathsf{D}_{\mathsf{RegLC}}$, which is equal to the complexity of `Dijkstra` on the product graph $G^\times$: $O(m \log n)$; $m = |A||S|^2$ and $n = |V||S|$ are the number of arcs and nodes of $G^\times$. The amount of memory needed to hold the distance data computed during the preprocessing phase varies depending on 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 $\pi(v,s)$ introduces a strong algorithmic overhead for `lsSDALT`. The number of calculated bounds to compute the potential function 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.

algo	regular language to constrain distance calculation $L_{s_0}^{v\to\ell}$	$L_{s_0}^{t\to\ell}$	$L_{s_1}^{v\to\ell}$	$L_{s_1}^{t\to\ell}$	settled nodes in $G^\times$ (key in square brackets)
$\mathsf{D}_{\mathsf{RegLC}}$	no potential				all
`std`	not constrained				$(r, s_0)[9], \ldots$ all except $(1, s_0)$, $(2, s_0)$
`bas_ls`	$(f\|b)^*$				$(r, s_0)[10]$, $(5, s_0)[10]$, $(6, s_1)[10]$, $(7, s_1)[11]$, $(8, s_1)[11]$, $(9, s_1)[11]$, $(10, s_1)[11]$, $(t, s_1)[11]$
`adv_ls`	$(f\|b)^*$		f^*		$(r, s_0)[10]$, $(5, s_0)[10]$, $(8, s_0)[11]$, $(9, s_1)[11]$, $(10, s_1)[11]$, $(t, s_1)[11]$
`spe_ls`	f^*bf^*	f^*			$(r, s_0)[11]$, $(8, s_0)[11]$, $(9, s_1)[11]$, $(10, s_1)[11]$, $(t, s_1)[11]$

(a)



(b)

(c)

Figure 5.4: Example showing the application of `lsSDALT` on a labeled graph. The shortest r-t-paths is constrained by the regular expression f^*bf^* (NFA in Figure 5.4c). Table 5.4a shows the regular languages used to constrain the landmark distance calculation, as well as the settled nodes by each algorithm. The optimal path is (r, s_0), $(8, s_0)$, $(9, s_1)$,$(10, s_1)$, (t, s_1) with cost 11. Node l is the landmark.

5.3 Label Correcting SDALT: `lcSDALT`

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. We name the new algorithm Label Correcting `SDALT` or shortly `lcSDALT`.

5.3.1 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 4, the missing lines are the same as in Algorithm 3). See Figure 5.5 for an example.

Algorithm 4 Pseudo-code `lcSDALT`

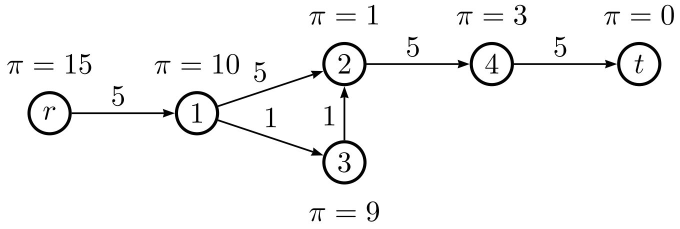
```
if $(w, s')$ not in $Q$ and never visited then        ▷ insert
    $\pi_{w,s'} \leftarrow \pi(w, s')$
    $k(w, s') \leftarrow d(w, s') + \pi_{w,s'}$
    insert $(w, s')$ in $Q$
else if $(w, s')$ not in $Q$ then                     ▷ re-insert
    $k(w, s') \leftarrow d(w, s') + \pi_{w,s'}$
    insert $(w, s')$ in $Q$
else                                                  ▷ decrease
    $k(w, s') \leftarrow d(w, s') + \pi_{w,s'}$
    decreaseKey $(w, s')$ in $Q$
```

5.3.2 Correctness

The algorithm `lcSDALT` is based on $\mathsf{D}_{\mathsf{RegLC}}$ 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 5.3.1 and Proposition 5.3.2). 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 5.3.1. *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



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
step 9: output path: (r,1,3,2,4,t) with cost 17.

Figure 5.5: Application of algorithm `lcSDALT` on a simple graph (not labeled, without constraints). The 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 (Q). Note that lsSDALT would have found the non-optimal path (r,1,2,4,t) with cost 20.

least one new node $p_{n+1} = (j, s')$ with $d(j, s') = d^*(j, s') = d^*(i, s) + c_{(i,s)(j,s')l}(\tau)$ is inserted in the queue by lines 18, 21, 24. □

Proposition 5.3.2. *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 $d(t, s) \neq d^*(t, s)$. Node (t, s) has key $k(t, s_f) = d(t, s_f) + \pi(t, s) \neq d^*(t, s)$. By Lemma 5.3.1, 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 $k(i, s') = d^*(i, s') + \pi(i, s') > d(t, s) + \pi(t, s) = k(t, s)$, which is a contradiction. □

5.3.3 Constrained landmark distances

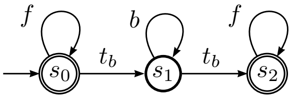
The methods (bas), (adv), and (spe) may be used with `lcSDALT`. However, `lcSDALT` produces a slight overhead in respect to `lsSDALT` as it unnecessarily checks if newly inserted nodes in Q have previously been extracted from the priority queue (line 18). Now we will present two new methods which can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

(adv$_{\mathsf{lc}}$) Equal to (adv), this method applies Procedure 2 to all nodes (v, s) of $G^\times$. Different to (adv) it uses Equation 5.1 as potential function and thereby considerably reduces the number of potentials to be calculated.

(spe$_{\mathsf{lc}}$) 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$_{\mathsf{lc}}$) 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 Table 5.6 for an example.

5.3.4 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 the worst case is equal to `spe_ls` and `adv_ls`, respectively.



(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 $\Sigma = \{f, b, t_b\}$.

$$L_0 : f^*|(f^*t_bb^*t_bf^*)$$

(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.

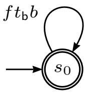
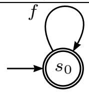
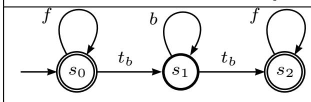
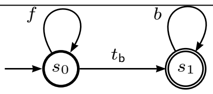
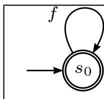
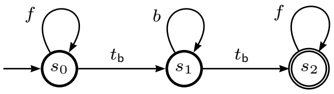
<table>
<tr><td>methods:</td><td>(bas)</td><td>(adv)/(adv_{lc})</td><td>(spe_{lc})</td><td>(spe)</td></tr>
<tr><td>$L_{s_0}^{\ell\to v}$</td><td colspan="3" rowspan="2"> $(b|f|t_b)^*$</td><td> f^*</td></tr>
<tr><td>$L_{s_0}^{\ell\to t}$</td><td> $f^*|(f^*t_bb^*t_bf^*)$</td></tr>
<tr><td>$L_{s_1}^{\ell\to v}$</td><td colspan="2" rowspan="2"></td><td colspan="2" rowspan="2"> $f^*t_bb^*$</td></tr>
<tr><td>$L_{s_1}^{\ell\to t}$</td></tr>
<tr><td>$L_{s_2}^{\ell\to v} = L_{s_2}^{\ell\to t}$</td><td></td><td> f^*</td><td colspan="2"> $f^*t_bb^*t_bf^*$</td></tr>
</table>

Figure 5.6: Example of a regular language L_0 and its representation as an automaton (Figure 5.6a) and regular expression (Figure 5.6b). 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): $L_{s_0}^{\ell\to v} = L_{s_0}^{\ell\to t} = L_{s_1}^{\ell\to v} = L_{s_1}^{\ell\to t} : (b|f|t)^*$, $L_{s_2}^{\ell\to v} = L_{s_2}^{\ell\to t} : f^*$. Preprocessing space is proportional to the number of automata used. Note that as walking is much slower than biking it is likely that the bounds for state s_0 when using (spe$_{\mathsf{lc}}$) are better than when using (spe). Memory space is also reduced as for (spe$_{\mathsf{lc}}$) only three automata are used during pre-processing instead of four. However, the potential function for (spe$_{\mathsf{lc}}$) is not feasible.

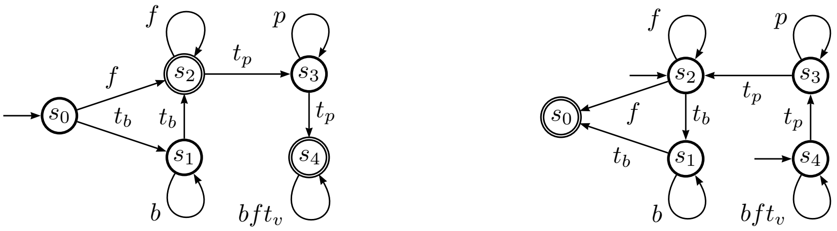


Figure 5.7: Example of an automaton (left) and its backward automaton (right). Shortest paths start either by walking (label f) or by taking a private bicycle: transfer to private bicycle (t_b) and moving on bicycle network (b). Once the private bicycle is discarded (s_1), the path can be continued by 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.

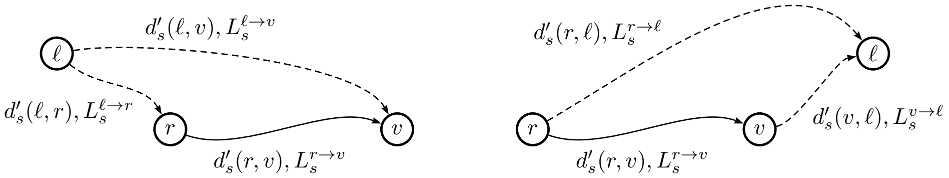

$d'_s(\ell, v), L_s^{\ell\to v}$
ℓ
$d'_s(\ell, r), L_s^{\ell\to r}$
r
v
$d'_s(r, v), L_s^{r\to v}$
$d'_s(r, \ell), L_s^{r\to \ell}$
ℓ
r
v
$d'_s(v, \ell), L_s^{v\to \ell}$
$d'_s(r, v), L_s^{r\to v}$

Figure 5.8: Landmark distances for backward search.

5.4 Bi-directional SDALT: biSDALT

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 [65, 97, 105] and we describe how we adapted them to SDALT.

5.4.1 Query

In general, bi-directional SDALT (biSDALT) works as follows. It alternates between running a lsSDALT query from source (r, s_0) to target $(t, s_f), s_f \in F$ (forward search) and a second lsSDALT query from all $(t, s_f), s_f \in F$ to (r, s_0) (backward search). Note that the backward search works on the *backward automaton*: all arcs of $\mathcal{A}_0$ are reversed, final states become initial states and initial states become final states (see Figure 5.7 for an example). The potential function for the backward search, π_B (see Figure 5.8), is a slight modification of the potential function for the forward search, π_F (equal to Equations 5.1):

$$\pi_F(v, s) = \max_{\ell\in\mathcal{L}}(d'_s(\ell, t) - d'_s(\ell, v), d'_s(v, \ell) - d'_s(t, \ell)) \tag{5.2}$$

$$\pi_B(v, s) = \max_{\ell\in\mathcal{L}}(d'_s(\ell, v) - d'_s(\ell, r), d'_s(r, \ell) - d'_s(v, \ell)) \tag{5.3}$$

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 [65]. We will discuss the non time-dependent and the time-dependent case.

Non time-dependent case. For networks without time-dependent arc costs, the authors of [105] 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 [65] 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 $k(v, s) \leq \mu$ 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 [97] 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 $(v, s) - (t, s_f)$ (backward search). Note that the cost of path $(v, s) - (t, s_f)$ 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 [97] 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

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

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') 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 includ 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

Table 5.2: 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 $G^\times$ for the backward query.

proc.		regular language and/or NFA
1B		equal to Procedure 1
2B		$L_s^{\ell\to v} = L_s^{\ell\to r} = \quad L_s^{r\to\ell} = L_s^{v\to\ell} = L_{\mathsf{proc2,s}} = \{\overleftarrow{\Sigma}(s, \mathcal{A}_0)^*\}$ $L_{\mathsf{proc2,s}} : \mathcal{A}_{\mathsf{proc2,s}} = \quad (\{s\}, \overleftarrow{\Sigma}(s, \mathcal{A}_0), \delta : \{s\} \times \overleftarrow{\Sigma}(s, \mathcal{A}_0) \to \{s\}, s, \{s\})$
3B	a)	$L_s^{\ell\to r} : \mathcal{A}_s^{\ell\to r} = \quad (S_0, \Sigma_0, \delta_0, s_0, s_0)$
	b)	$L_s^{\ell\to v} : \mathcal{A}_s^{\ell\to v} = \quad (S_0, \Sigma_0, \delta_0, s_0, s)$
	c)	$L_s^{r\to\ell} : \mathcal{A}_s^{r\to\ell} = \quad \mathcal{A}_0$
	d)	$L_s^{v\to\ell} : \mathcal{A}_s^{v\to\ell} = \quad (S_0, \Sigma_0, \delta_0, s, F_0 \cap \overleftarrow{S}(s, \mathcal{A}_0))$
	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

function π_F can be enhanced by using the distance already calculated by the backward search. In the second variation of $\mathtt{bi}_{v0}$, which we call $\mathtt{bi}_{v2}$, as soon as the backward search stops we switch to `lcSDALT` for the forward search and use the potential $\pi_F(v, s) = d(v, s)$ for every visited node; $d(v, s)$ 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 functon is not feasible and therefore the forward search has to switch lo `lcSDALT`.

5.4.2 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 5.8). Similar to Proposition 5.1.1, the following Proposition 5.4.1 gives first indications.

Proposition 5.4.1. *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 5.2 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$_\mathsf{B}$) applies Procedure 1B to determine the constrained distance calculation and is equal to Procedure 1. The *advanced method* (adv$_\mathsf{B}$) 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:

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

Finally, the *specific method* (spe$_\mathsf{B}$) 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$_\mathsf{B}$), (adv$_\mathsf{B}$) or (spe$_\mathsf{B}$) 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.

5.4.3 Correctness

The variants of `biSDALT` are based on the principles outlined in [65, 97] and Section 5.3.2.

Proposition 5.4.2. *If solutions exist, the variants of `biSDALT` find a shortest path.*

5.4.4 Memory requirements

Memory requirements to hold preprocessing data for `bas-bi`$_{\mathsf{vx}}$ and `spe-bi`$_{\mathsf{vx}}$ are equal to requirements of (bas$_{\mathsf{ls}}$) and (spe$_{\mathsf{ls}}$), 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$_{\mathsf{ls}}$).

5.5 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 [97], 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). See for more details Section 3.3. 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.

5.5.1 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 5.9, 5.11, 5.13, 5.15, 5.17, and 5.19. The regular expressions of all 26 scenarios can be found in Tables 5.5 and 5.7.

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 [65]. 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 $\mathsf{D}_{\mathsf{RegLC}}$ 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.) 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 5.3 and 5.4 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 5.6) with $\mathsf{D}_{\mathsf{RegLC}}$ [8] and `std` (which is based on the goal directed search algorithm `go` presented in [6]). To the best of our knowledge, no other comparable methods on finding constrained shortest paths on multi-modal networks exists in the literature. A direct comparison to the methods presented in [109] and [50] is not possible as they do not consider time-dependent arc costs on the road network and are only applicable to specific scenarios.

Table 5.3: 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 5.4: 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

Table 5.5: Regular expressions of test scenarios for experimental evaluation.

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

Table 5.6: 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$_{v2}$

5.5.2 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 5.7. Speed-ups in respect to $\mathsf{D}_{\mathsf{RegLC}}$ 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 $\mathsf{D}_{\mathsf{RegLC}}$ and `std`, and that the faster the modes of transportation which are excluded, the higher the speed-up. This can be explained intuitively by the observation that `std` guides the search toward arcs with the lowest cost on the shortest *un-constrained* path to the target. Furthermore, time-dependency has a negative impact on runtime and especially on the runtime of bi-directional search. Lower bounds are calculated using the minimum weight cost function and thus are sometimes very different to the real travel times especially for public transportation at night time as connections are not served as frequently as during the day. The stopping condition for bi-directional search for scenarios involving time-dependent arc costs is weaker than the stopping condition used when no time-dependent arc costs are involved. That is why bi-directional search performs often worse than uni-directional search. However, the advantage of bi-directional search is that approximation can be applied. By applying approximation bi-directional search runs in most scenarios faster than uni-directional search.

Complex constraints

Let us now look at the scenarios with more complex constraints. In Figures 5.10, 5.12, 5.14, 5.16, 5.18, and 5.20, 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 5.21 and 5.21). See Figure 5.10 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 $\mathsf{D}_{\mathsf{RegLC}}$. Again, 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 preprocessing 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 5.7 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`. For the latter, `adv_ls` provides better bounds for states s_1 and s_3 than `bas_ls` because it excludes public transportation and the use of a rental car, respectively, during the calculation of the constrained landmark distances. Version `spe_ls` has a positive impact on runtimes for scenarios where the constrained shortest path

Table 5.7: 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$_{\text{RegLC}}$ [ms]	std [ms]	bas_ls [ms]	bas_bi$_{v0}$ [ms]	10%[a] [ms]	20%[a] [ms]
$(f)^*$	only foot	IDF	19s	88	117	5	*4	4	4
		NY	6s	27	38	*1.6	2.4	1.8	1.8
$(b\|f\|t_b)^*$	bike	IDF	32s	199	248	13	9	*8	8
		NY	12s	75	96	5.4	3.2	*2.9	2.9
$(c\|f\|t_c)^*$	car	IDF	57s	356	130	124	261	179	*117
		NY	11s	68	96	3.8	2.6	*2.4	2.4
$(f\|p_c\|p_m\|p_t\|p_r\|$ $p_b\|t_p)^*$	public trans	IDF	34s	182	186	*116	291	269	251
		NY	9s	63	76	*37	89	69	58
$(f\|p_c\|p_m\|p_t\|t_p)^*$	tram,metro	IDF	24s	135	175	23	44	24	*22
		NY	9s	48	64	*14	30	26	20
$(f\|p_c\|p_r\|t_p)^*$	trains	IDF	29s	166	172	*73	177	162	155
		NY	8s	42	57	*17	35	26	23
$(f\|p_b\|p_c\|t_p)^*$	bus	IDF	28s	174	216	*157	431	419	408
		NY	9s	61	79	*35	90	89	81
$(b\|f\|t_v)^*$	rental bike	IDF	30s	223	300	10	5	*4	4
$(c\|f\|t_a)^*$	rental car	IDF	51s	509	623	90	96	16	*11
$(c_f\|c_p\|c_u\|f\|t_c)^*$	private car, no toll roads	IDF	57s	347	126	108	219	132	*90
$(c_p\|c_u\|f\|t_c)^*$	private car, no toll/fast roads	IDF	55s	340	209	*134	349	251	184

[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.

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). The regular languages used to calculated the constrained landmark distances for spe_ls include information about these constraints, thus spe_ls is able to anticipate and guide the search faster to the target than bas_ls and adv_ls. 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 out-balanced 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 search which employs approximation runs 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 un-constrained 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.

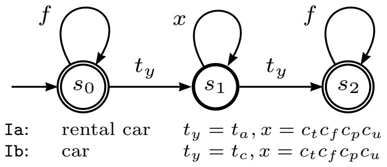

f
x
f
t_y
t_y
s_0
s_1
s_2
Ia: rental car $t_y = t_a, x = c_t c_f c_p c_u$
Ib: car $t_y = t_c, x = c_t c_f c_p c_u$

Figure 5.9: Scenarios `I`: a path starts and ends by walking. A car (scenario `Ia`) or rental car (scenario `Ib`) may be used once.

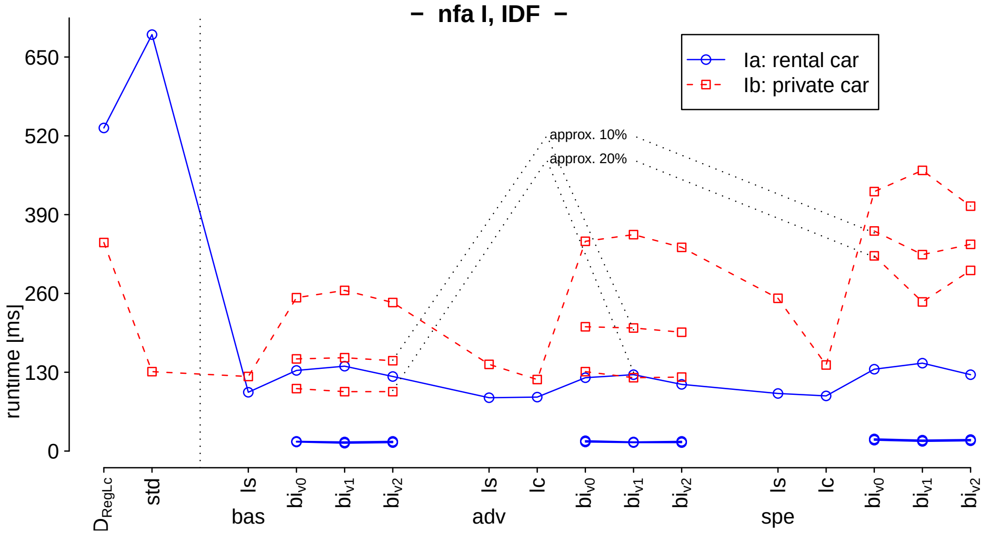


Figure 5.10: Experimental results for scenarios `I`. The different line-types indicate average runtimes (in milliseconds [ms]) of the different `SDALT` variants when varying the allowed modes of transportation. In this example, the continuous blue and dashed red lines indicate average runtimes for the different `SDALT` variants for scenarios `Ia` and `Ib`. We provide average runtimes for $\mathsf{D_{RegLC}}$, `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 $\mathsf{D_{RegLC}}$ 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 $\mathsf{D_{RegLC}}$ are about 360ms. `std` and `bas_ls` provide a speed-up of about factor 3. The other algorithms do not provide better results.

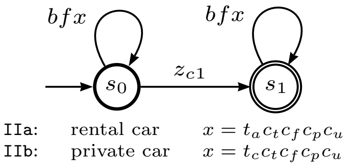

bfx
bfx
s_0
z_{c1}
s_1
IIa: rental car $x = t_a c_t c_f c_p c_u$
IIb: private car $x = t_c c_t c_f c_p c_u$

Figure 5.11: 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.

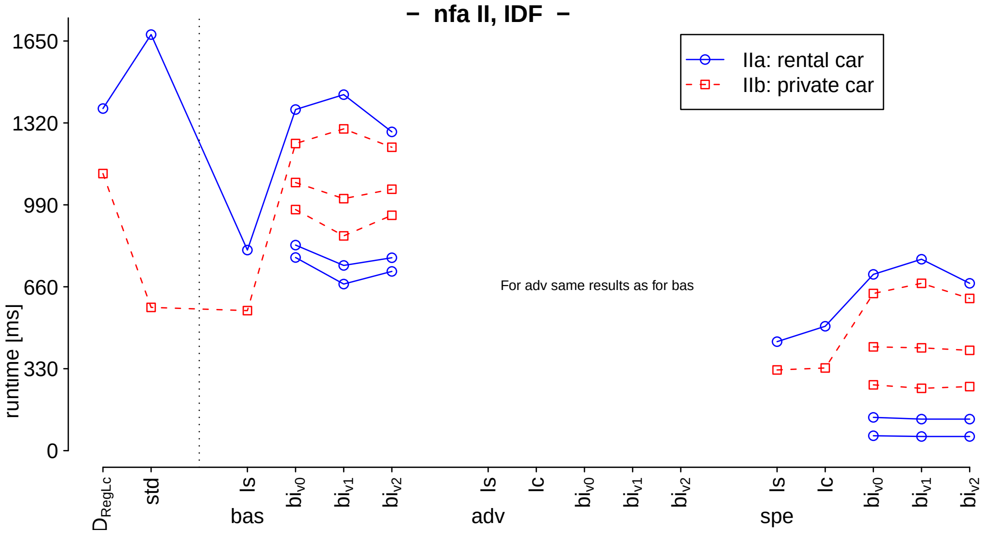
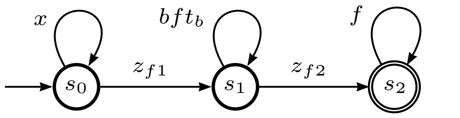


Figure 5.12: Experimental results for scenarios II. For scenario IIa std is slower than $\mathsf{D}_{\mathsf{RegLC}}$. bas_ls and bas_bi$_{vx}$ provide a speed-up of about factor 2. spe_ls runs slightly faster. The bi-directional algorithms spe_bi$_{vx}$ work very well and provide average runtime of 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	$x = t_a c_t c_f c_p c_u$
`IIIb`:	private car, no fast roads	$x = t_a c_p c_u$
`IIIc`:	public trans	$x = t_p p_b p_m p_r p_t$
`IIId`:	metro/tram	$x = t_p p_m p_t$

Figure 5.13: 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.

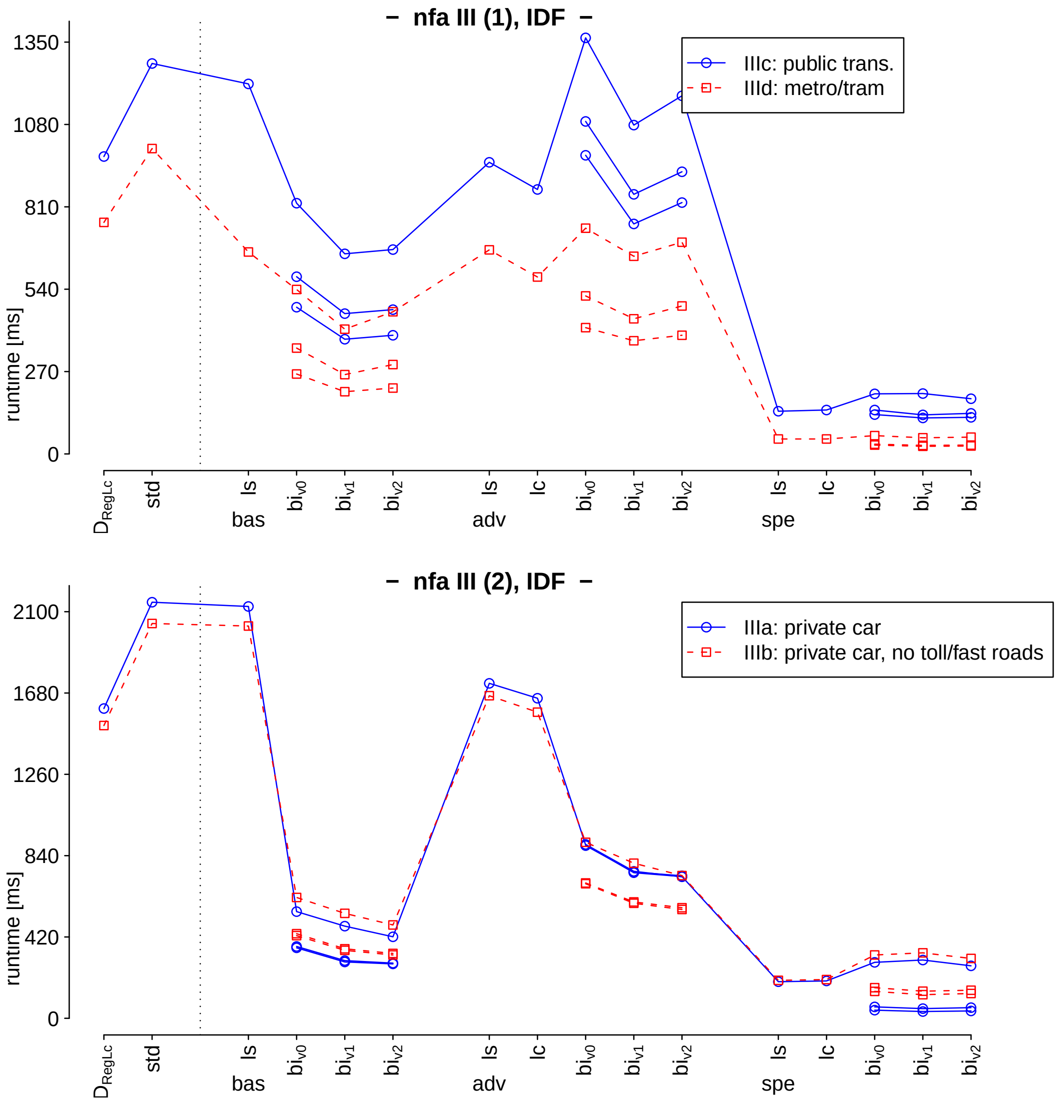
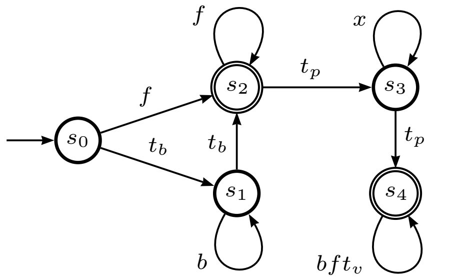


Figure 5.14: Experimental results for scenarios `III`. For all scenarios the algorithms `std`, `bas_ls`, `adv_ls`, and `adv_ls` are not very efficient. Instead, `spe_ls` and `spe_lc` and the bi-directional versions work very well. They provide a speed-up of a factor of 10 to 15.



`IVa`:	public trans, no changing	$x = p_b p_m p_r p_t$
`IVb`:	public trans	$x = p_b p_m p_r p_t p_c$
`IVc`:	metro/tram, no changing	$x = p_t p_m$

Figure 5.15: 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.

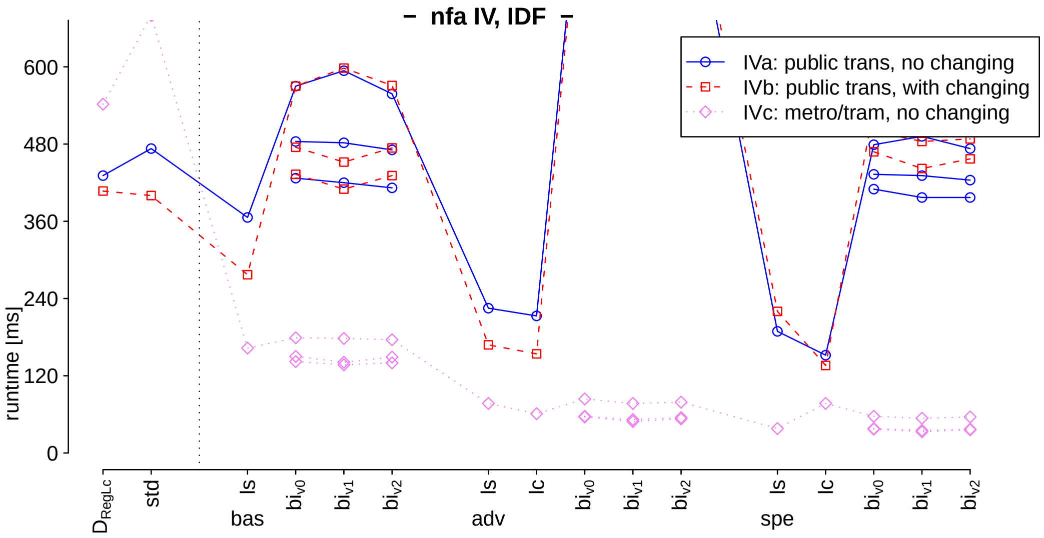


Figure 5.16: Experimental results for scenarios `IV`. The bi-directional versions of the algorithm and `std` are not efficient. Instead, `bas_ls`, `adv_ls`, and `spe_ls` provide speed-ups of a factor between 2 and 10.

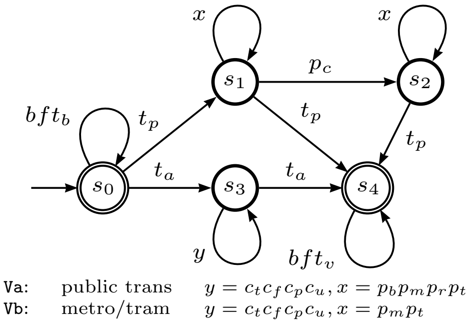

x
x
s_1
p_c
s_2
bft_b
t_p
t_p
t_p
s_0
t_a
s_3
t_a
s_4
y
bft_v
Va: public trans $y = c_t c_f c_p c_u, x = p_b p_m p_r p_t$
Vb: metro/tram $y = c_t c_f c_p c_u, x = p_m p_t$

Figure 5.17: 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.

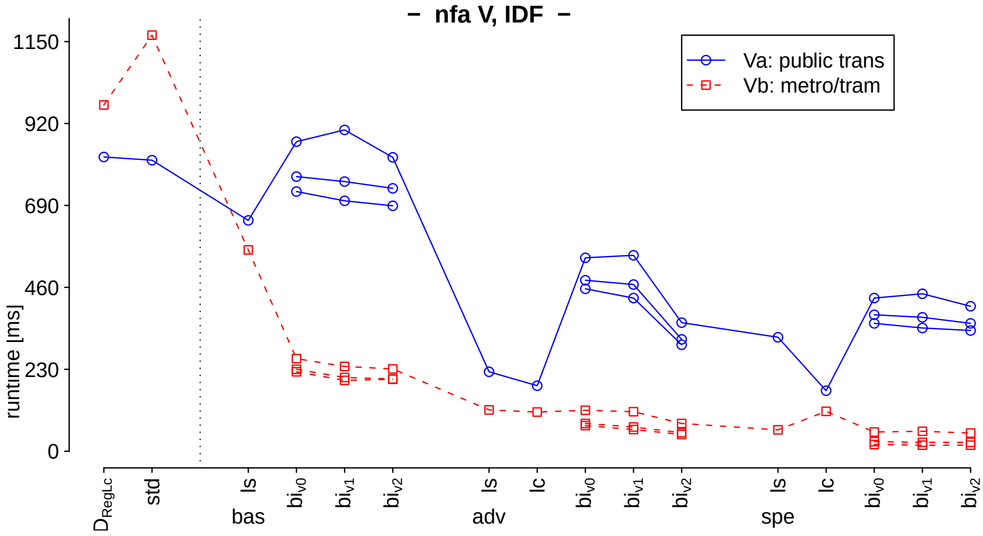
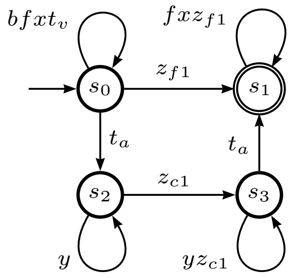


Figure 5.18: Experimental results for scenarios V. Bi-directional search does not work well if public transportation can be used (scenario Va). Instead, if public transportation is restricted (scenario Vb) bi-directional search is very fast. For scenario Vb, bi-directional search with approximation of 20% provides a speed-up of about a factor of 60, spe_ls of a factor of 15.



VIa:	metro/tram	$y = c_t c_f c_p c_u$, $x = p_m p_t$
VIb:	no public trans	$y = c_t c_f c_p c_u$, no x

Figure 5.19: 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.

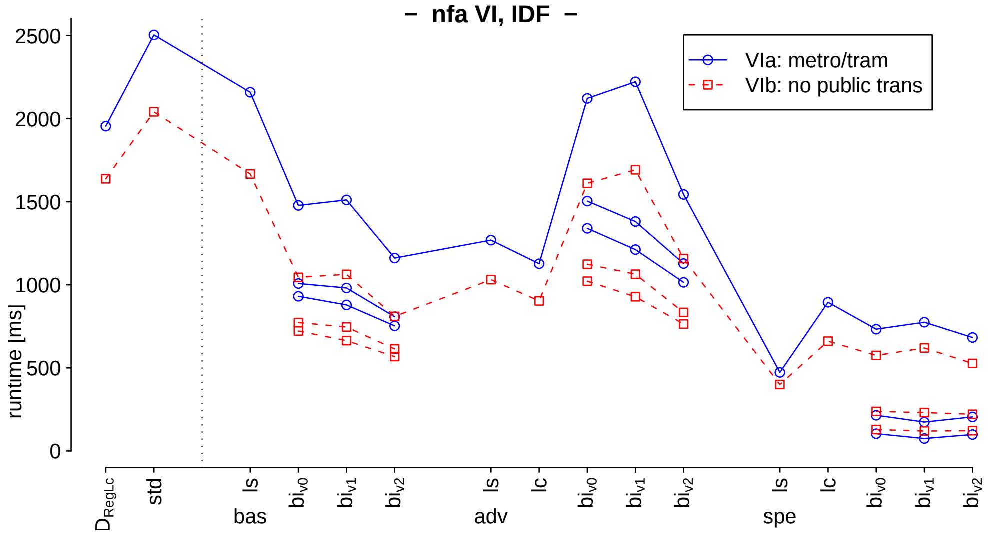


Figure 5.20: Experimental results for scenarios VI.

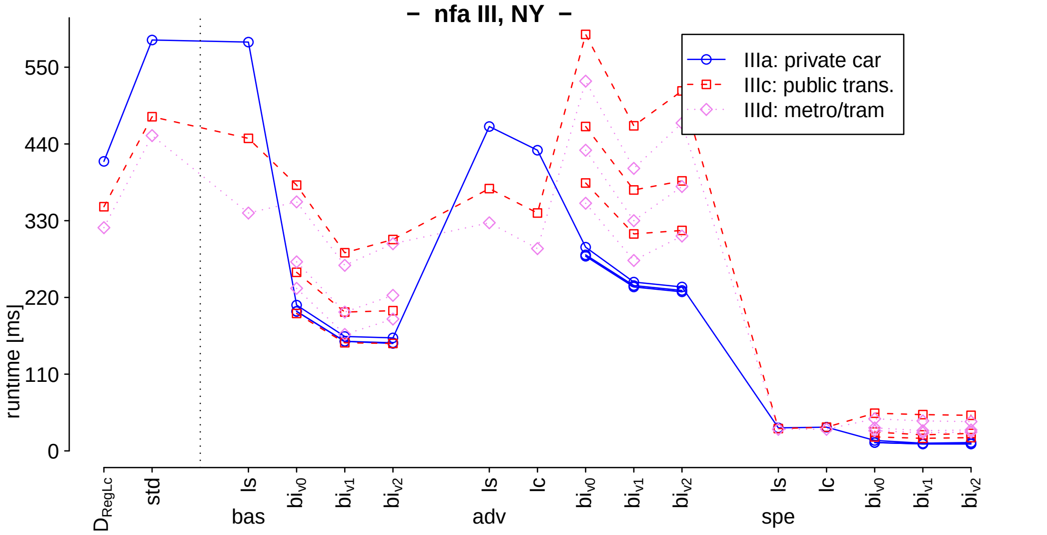


Figure 5.21: Experimental results for scenarios III on network NY.

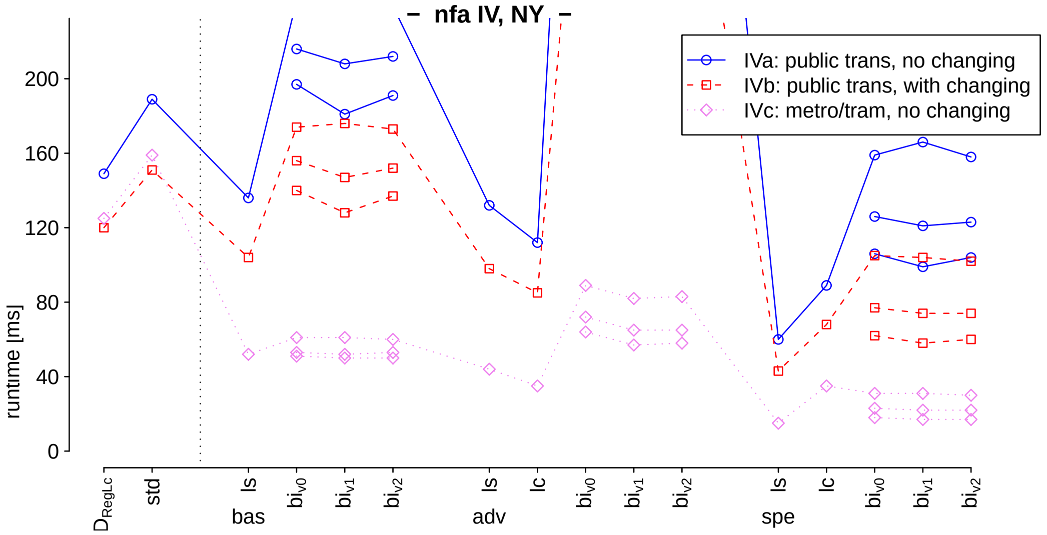


Figure 5.22: Experimental results for scenarios IV on network NY.

5.6 Summary

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 (D_{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 [109] 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 5.7. Here, SDALT provides maximum speed-ups of about factor 20. However, besides limiting the range of applicable regular languages, [109] 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 [50] 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. Furthermore, they do not discuss how to integrate other modes of transportation which use the road network, such as rental or private bicycle.

SDALT is a general method to speed-up D_{RegLC} for all regular languages and which can be applied to multi-modal 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.

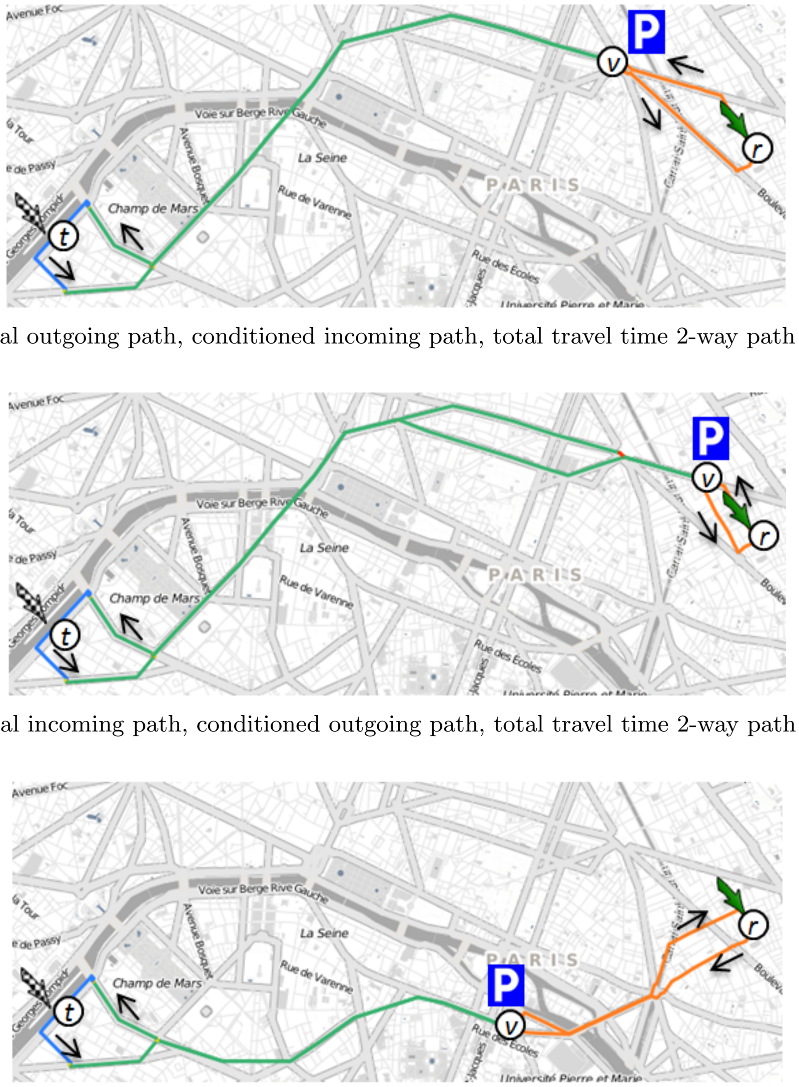
Chapter 6

2-Way Multi-Modal Shortest Path Problem

In this chapter, we present an efficient algorithm to solve the 2-way multi-modal shortest path problem (2WMMSP). Its goal is to find an optimal return path on a multi-modal network. It consists of an outgoing path from the source to the target location and an incoming path from the target to the source location. The shortest incoming path is often not equal to the shortest outgoing path as traffic conditions and timetables of public transportation vary throughout the day. The main difficulty lies in finding an optimal parking location for a private bicycle or private car, since they may be used for parts of the outgoing path and will need to be picked up during the incoming path. We present experimental results on a real-world multi-modal transportation network. Our algorithm outperforms a previous algorithm [24]. Note that parts of this work have been presented in [73].

6.1 Problem Definition

Multi-modal transportation networks include various modes of transportation, such as cars, public transportation, bicycles, etc. Consequently, a path on such a network may be composed of parts, each of which is covered by different modes of transportation. Other than finding shortest multi-modal paths from a source location to a target location which optimally combine the use of several modes of transportation, yet another problem arises: finding an optimal multi-modal *return path* or *2-way path.* A 2-way path consists of two paths, an *outgoing path* from the source to the target location and an *incoming path* from the target to the source location. On a multi-modal transportation network, the shortest incoming path is often not equal to the shortest outgoing path as traffic conditions and timetables of public transportation vary throughout the day and one-way roads prevent the same path from being taken in the opposite direction. Furthermore, in some cases the user will start from the source location with a private car or bicycle and then transfer at a later point to public transportation or walk in order to reach the target location. At some intermediate parking location, the user has to park his car or bicycle which he will want to pick up on the incoming path in order to take it home. The travel time of the incoming path may be heavily negatively influenced by a parking location which is not picked wisely. Therefore, it



(a) Optimal outgoing path, conditioned incoming path, total travel time 2-way path 1h31min.

(b) Optimal incoming path, conditioned outgoing path, total travel time 2-way path 1h29min.

(c) Optimal 2-way path, total travel time 2-way path 1h25min.

Figure 6.1: Three different 2-way paths between a source location r and a target location t. The parking location is marked by v. The blue, orange, and green lines represent walking, bicycle, and public transportation, respectively. In Figure 6.1a, the 2-way path has been determined by first calculating the optimal outgoing path and then the incoming path. The incoming path is *conditioned* as it has to pass by the parking location v used by the outgoing path. In Figure 6.1b, the optimal incoming path has been determined first and then the conditioned outgoing path. Figure 6.1c shows the optimal 2-way path.

is important to choose a parking location which optimizes the combined travel times of the outgoing and incoming path, see Figure 6.1. We call this problem the 2-way multi-modal shortest path problem (2WMMSP).

Let us define the concepts of a 2-way path and the 2WMMSP problem in a more formal way (with reference to Figure 6.2):

Definition 6.1.1 (2-way path)**.** *A 2-way path*

$$\overleftrightarrow{p} = (v, p_1^o \circ p_2^o \circ p_1^i \circ p_2^i) \tag{6.1}$$

between two nodes, r and t, consists of a parking node v and the concatenation of four paths $p_1^o = (r, \ldots, v)$, $p_2^o = (v, \ldots, t)$, $p_1^i = (t, \ldots, v)$, and $p_2^i = (v, \ldots, r)$. $p^o = p_1^o \circ p_2^o$ is the outgoing path and $p^i = p_1^i \circ p_2^i$ is the incoming path. The cost $\gamma(\overleftrightarrow{p}, \tau_r^o, \tau_t^i)$ of a 2-way path $\overleftrightarrow{p}$ with departure time τ_r^o at r for path p_1^o and departure time τ_t^i at t for path p_1^i is

$$\gamma(\overleftrightarrow{p}, \tau_r^o, \tau_t^i) = \gamma(p_1^o, \tau_r^o) + \gamma(p_2^o, \gamma(p_1^o, \tau_r^o) + \tau_r^o) + \gamma(p_1^i, \tau_t^i) + \gamma(p_2^i, \gamma(p_1^i, \tau_t^i) + \tau_t^i). \tag{6.2}$$

As we are interested in multi-modal paths, we *constrain* the four partial paths by the regular languages R_1^o, R_2^o, R_1^i, and R_2^i. In this way, we can control the use of modes of transportation which can be employed for the four parts of the 2-way path. See Figure 6.3 for a full example. Generally, by using the regular languages, the product graph is split into two parts, where the common nodes of both parts represent the parking nodes. Note that the parking node can be equal to the source or target node.

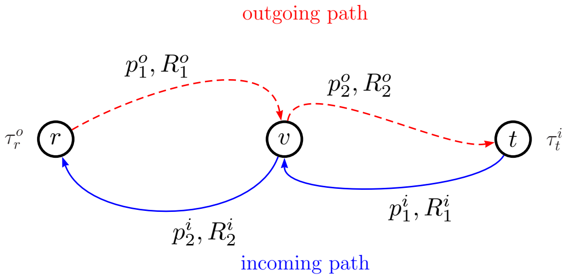
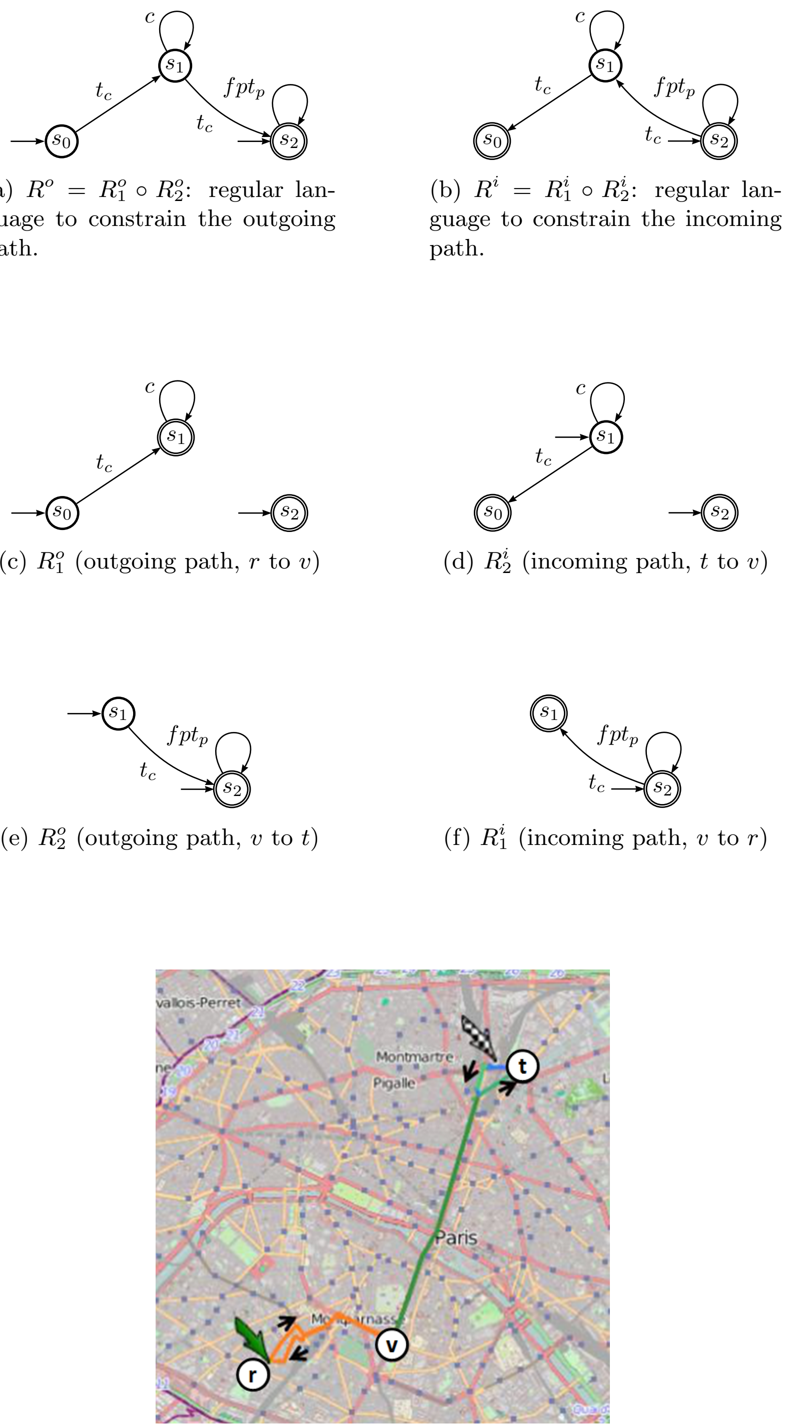


Figure 6.2: Schema 2-way path.

Definition 6.1.2 (2-way multi-modal shortest path problem (2WMMSP))**.** *Given a directed, labeled graph $G = (V, A, \Sigma)$ with a cost function $c : A \to \mathbb{F}$, a source node $r \in V$, a target node $t \in V$, a departure time τ_r at r, a departure time τ_t at t, find a 2-way path $\overleftrightarrow{p}$ such that $\gamma(\overleftrightarrow{p}, \tau_r^o, \tau_t^i)$ is minimal.*

The definition can be further generalized by introducing the possibility to specify not the departure time τ_r^o at r but the arrival time τ_t^o at t, and/or for the incoming path, not the departure time τ_t^i at t but the arrival time τ_r^i at r. Note that in these cases only the cost function has to be slightly modified. Our algorithm can handle all these cases but for simplicity, when describing our algorithm, we will only consider the case where departure times are given for r and for t, for the incoming path and for the outgoing path, respectively, as specified in Definition 6.1.1.

To the best of our knowledge, the 2WMMSP has only been previously studied in [24]. The authors of [24] adopt a brute force algorithm by calculating all paths between the source and target location, and a pre-determined set of possible parking nodes.



(a) $R^o = R^o_1 \circ R^o_2$: regular language to constrain the outgoing path.

(b) $R^i = R^i_1 \circ R^i_2$: regular language to constrain the incoming path.

(c) R^o_1 (outgoing path, r to v)

(d) R^i_2 (incoming path, t to v)

(e) R^o_2 (outgoing path, v to t)

(f) R^i_1 (incoming path, v to r)

(g) Example of a 2-way path (Map from OpenStreetMap)

Figure 6.3: Example of a 2-way path with parking node v between source node r and target node t. We suppose that all paths start and end at the foot network. The outgoing path is constrained by the regular language R^o (represented by an automaton). It states that the car can be used at the beginning of the journey (by accessing the car layer from the foot layer via a transfer arc with label t_c). Once the car is discarded, the journey may continue by public transportation (labels p and t_p) or by walking (label f). For partial paths p^n_1, p^o_2 only the car may be used, for partial paths p^i_2, p^o_1 only public transportation and walking may be used. Figure 6.3g shows a valid 2-way path. Orange lines mark the car, light and dark green lines mark public transportation, and blue lines mark walking.

6.2 Basic Algorithm

We call our algorithm `2-WAY-PATH-SEARCH`. The basic version of the algorithm (*basic*) works as follows (with reference to pseudo-code in Algorithm 5). We alternate the execution of four $\mathsf{D_{RegLC}}$ algorithms (see Section 4.3.1) on the labeled graph G (see Chapter 3) of the transportation network: algorithms D_1^o and D_2^o for the outgoing path, and D_1^i and D_2^i for the incoming path (see Figure 6.4). D_1^o and D_2^i calculate the shortest paths from r to all other nodes, and D_2^o and D_1^i from t to all other nodes. Algorithms D_1^o and D_1^i apply forward search, and algorithms D_2^o and D_2^i apply backward search (lines 3 to 12).

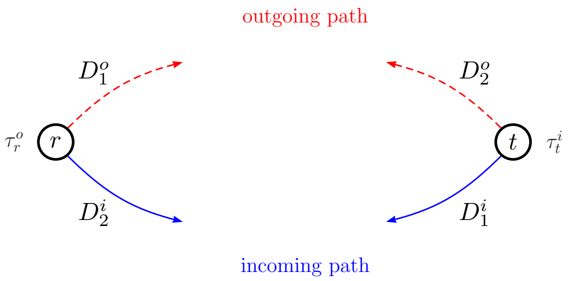


Figure 6.4: Schema algorithm `2-WAY-PATH-SEARCH`.

At each iteration, we choose the algorithm D out of D_1^o, D_2^o, D_1^i, and D_2^i, which node x to be settled next, has the lowest key among the nodes yet to be settled by the four algorithms (line 21). D executes one $\mathsf{D_{RegLC}}$ step: node x is settled and all outgoing edges are relaxed (line 27). Now, if node x has been settled by all four algorithms, a new 2-way path has been found, for which x is the parking node (line 29). To evaluate the cost of the 2-way path, it suffices to add the costs of the 4 shortest paths to x as calculated by the 4 $\mathsf{D_{RegLC}}$ algorithms (line 41). If the total cost is lower than the cost μ of the best 2-way path found up to this point, then μ is updated and x is memorized. The algorithm may stop as soon as the key δ of the next node to be settled is greater than or equal to μ (line 24) or the four priority queues of the four $\mathsf{D_{RegLC}}$ algorithms are empty (line 20).

Time-dependency. In scenarios involving time-dependent arc costs, the starting times of the four algorithms have to be specified. However, starting times for the backward searches D_2^o and D_2^i are not known. For this reason, we employ the minimum weight cost function $\underline{c_{ijl}} = \min_{\tau \in \mathcal{T}} c_{ijl}(\tau)$ when evaluating arc costs. However, because of this, when calculating the cost of a 2-way shortest path, the cost of the paths produced by D_2^o and D_2^i have first to be *re-evaluated* by two $\mathsf{D_{RegLC}}$ algorithms starting in x. Correct starting times are given by the keys of x in D_1^o and D_1^i (lines 14-15 and lines 36-39).

6.2.1 Correctness

At each step, the node with minimum key is settled (among the next nodes to be settled by the four $\mathsf{D_{RegLC}}$ algorithms D_1^o, D_2^o, D_1^i, and D_2^i). In this way, the algorithm may stop as soon as the key of the next node to be settled is higher than the cost of the current best 2-way path (line 24). Furthermore, all four paths connecting the parking node with the source and target node are shortest paths. Therefore it is easy to see that:

Theorem 6.2.1. *If a solution exists, algorithm* `2-WAY-PATH-SEARCH` *returns an optimal 2-way path.*

6.2.2 Complexity

Let n be the number of nodes in the product graph $G^\times$, and m the number of arcs. The runtime of $\mathsf{D_{RegLC}}$ is $O(m \times \log(n))$. Note that `2-WAY-PATH-SEARCH` uses 4 $\mathsf{D_{RegLC}}$, so its run time is in $O(4 \times m \times \log(n))$. In cases where paths have to be re-evaluated because of time-dependent arcs costs, the complexity becomes $O(4 \times m \times \log(n) + 2n \times m \times \log(n))$ which is dominated by $O(2n \times m \times \log(n))$.

6.3 Speed-up Techniques

In this section, we present some techniques to speed-up `2-WAY-PATH-SEARCH`. Improvement I1 can be applied to non time-dependent as well as time-dependent scenarios. The other improvements are aimed for time-dependent scenarios for which re-evaluations of parking nodes are necessary. Note that the runtime of the re-evaluation process largely dominates the runtime of the rest of the algorithm.

I1: Improved stopping condition. We produce a lower bound λ on the cost of the 2-way paths which have not yet been found. As soon as $\lambda > \mu$, the algorithm may stop. μ is the cost of the current best 2-way path. We define the temporary cost $\gamma(x)$ as the sum of the keys of node x of the algorithms which already settled node x,

$$\gamma(x) = \sum_{D \in \{D_1^o, D_2^o, D_1^i, D_2^i\} \wedge \text{settled}(D,x)=1} \text{key}(D, x). \tag{6.3}$$

Function key(D, x) returns the key assigned to node x by algorithm D. Function settled(D, x) returns 1 if node x has been settled by algorithm D, 0 otherwise. We define α_i as the minimum $\gamma(x)$ over all nodes $x \in V^\times$ settled by exactly i of the 4 algorithms:

$$\alpha_i = \min_{x \in V^\times \wedge \text{numSet}(x)=i} \gamma(x) \tag{6.4}$$

$$\text{numSet}(x) = \sum_{D \in \{D_1^o, D_2^o, D_1^i, D_2^i\}} \text{settled}(D, x) \tag{6.5}$$

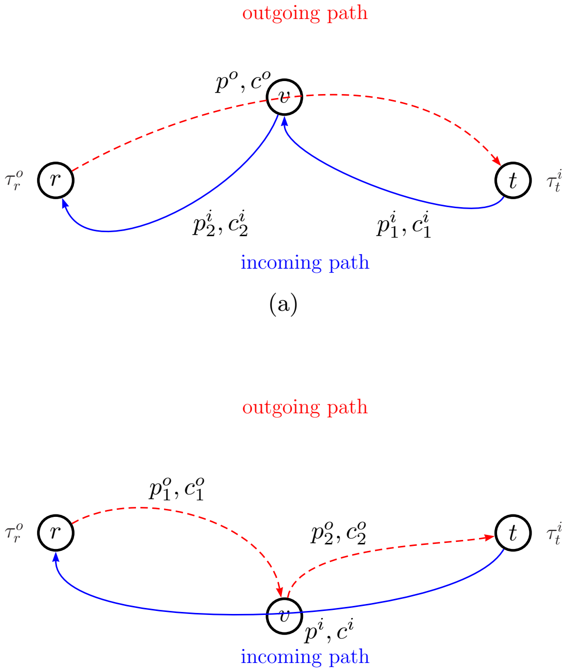
Proposition 6.3.1.

$$\lambda = \min_{i \in 1..3} (\alpha_i + (4-i)\delta), \tag{6.6}$$

is a valid lower bound on the costs of the 2-way paths not yet found, where δ is the cost of the next node to be settled by Algorithm 5.

Proof. Any node x which has not yet been settled by all four algorithms must have been settled either 0, 1, 2, or 3 times and its cost is at least 4δ, $\alpha_1 + 3 * \delta$, $\alpha_2 + 2 * \delta$, or $\alpha_3 + \delta$. Note that $4\delta \geq \alpha_1 + 3 * \delta$. Thus, as soon as $\lambda > \mu$, μ is the cost of the optimal 2-way path. □

I2: Upper bound / lower bound. Instead of initializing μ to ∞, we use an upper bound μ_{ub}. In this way, we enforce the condition in line 34 and fewer potential parking nodes have to be re-evaluated. We determine the upper bound in the following way (with reference to Figure 6.5). First, the cost c^o of the shortest outgoing path from r to t and the used parking node v is determined. Then, the costs c_1^i and c_2^i of the shortest t-v-path and v-r-path are calculated. $\mu'_{ub} = c^o + c_1^i + c_2^i$ is an upper bound on the cost μ^* of the optimal 2-way path. Similarly, we calculate the cost c^i of the incoming path from t to r, and we determine the used parking location v. By calculating the costs c_1^o and c_2^o of the shortest r-v-path and v-t-path, we obtain $\mu''_{ub} = c^i + c_1^o + c_2^o$. Then we set $\mu_{ub} = \min\{\mu_{ub}^1, \mu_{ub}^2\}$. Note that $\mu_{lb} = c^o + c^o$ is a valid lower bound of the cost of the optimal 2-way path. The algorithm may stop as soon as $\mu = \mu_{lb}$. The calculation of the lower bound μ_{lb} and upper bound μ_{ub} is done by running six times $\mathsf{D}_{\mathsf{RegLC}}$. Experimental results show that this effort is largely compensated by a consistent reduction of the number of parking nodes to be re-evaluated.



(a)

(b)

Figure 6.5: Path calculation for improvement I2.

I3: Minimum cost in time interval. Instead of calculating the minimum cost of an arc over the entire time horizon $\mathcal{T} = [0, P]$, we may use the minimum cost of a more restricted time interval $[\tau_1, \tau_2]$, see Figure 6.6. For algorithm D_2^o we may use cost:

$$c_{jkl}^{\tau_1,\tau_2} = \min_{\tau \in [\tau_1, \tau_2]} c_{jkl}(\tau) \tag{6.7}$$

where

$$\begin{aligned} \tau_1 &= \tau_r^o \\ \tau_2 &= \tau_r^o + \mu_{up} - c^i - \mathrm{key}(D_2^o, j) \end{aligned} \tag{6.8}$$

The starting time at r is τ_r^o, values prior to this time may be ignored. The cost of the outgoing path is bound by $\tau_r^o + \mu_{up} - c^i$ minus the key of the node j. c^i is the cost of the shortest path from t to r as defined for I2. The interval for evaluating the cost function for D_2^i is determined in a similar way.

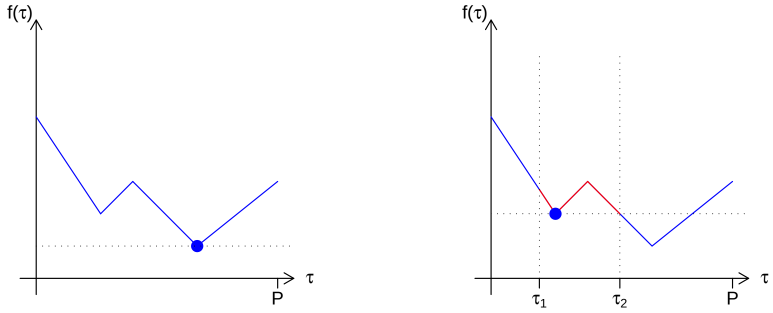


Figure 6.6: Minimum value over entire time period and in time interval $[\tau_1, \tau_2]$.

I4: Postponed re-evaluation. The re-evaluation of parking nodes may be postponed until the stopping condition applies (line 24). Note that in this case, the upper bound μ stays constant and is not updated, lines 36-44 are not executed. The parking nodes are re-evaluated in order of increasing temporary cost (as calculated on line 33). The re-evaluation stops as soon as μ is greater than or equal to the temporary cost of the next parking node to be re-evaluated.

I5: Parallel computing. When applying I4, the re-evaluation of parking nodes can be parallelized. Groups of parking node are assigned to a process which re-evaluates their costs.

I6: `SDALT` for re-evaluation. The `SDALT` (`bas_ls`) algorithm (see Chapter 5) instead of D_{RegLC} is used for the re-evaluation of nodes in lines 14-15.

I7: Multi-criteria algorithm for re-evaluation. For a list of parking nodes, let us suppose that only one path out of the four paths p_o^1, p_o^2, p_i^1, and p_i^2, which compose a 2-way path, is not known. In order to find the parking node which minimizes the cost of the optimal 2-way path a multi-criteria-type $\mathsf{D}_{\mathsf{RegLC}}$ algorithm may be applied, instead of re-evaluating every parking node one by one. We call this algorithm `Multi-Source-`$\mathsf{D}_{\mathsf{RegLC}}$. For each node, it keeps track of a key which is a tuple composed of the starting time τ at the node and its cost c. However, `Multi-Source-`$\mathsf{D}_{\mathsf{RegLC}}$ only optimizes the cost of the shortest path. The algorithm is initialized by inserting all the parking nodes with their respective starting times and costs into the priority queue. At each step the tuple with the lowest cost is extracted and its outgoing arcs are relaxed. As soon as a tuple belonging to the target node is extracted from the priority queue, the algorithm may stop.

The following dominance rule applies (based on the FIFO property, see Section 2.3). A tuple (τ_i, c_i) dominates a tuple (τ_j, c_j) if

$$\tau_i < \tau_j \wedge c_i - c_j \leq \tau_i - \tau_j. \tag{6.9}$$

Approximation. Finally, approximation can be applied. The algorithm is stopped when

$$\mu_{lb} \times (1 + \alpha) \geq \mu, \tag{6.10}$$

where $\alpha > 0$ is the approximation factor and μ_{lb} a lower bound of the cost of the optimal 2-way path (see I2). By applying approximation, we have that

$$\mu \leq (1 + \alpha)\mu^*. \tag{6.11}$$

The cost μ of the found 2-way path will not be higher than $(1 + \alpha) * \mu^*$, where μ^* is the cost of the optimal 2-way path.

Algorithm 5 `2-WAY-PATH-SEARCH` to calculate an optimal 2-way path. $\mathsf{D}_{\mathsf{RegLC}}$ (G, R, d) returns a $\mathsf{D}_{\mathsf{RegLC}}$ algorithm instance working on graph G, with regular language R, and direction d (f for forward search, b for backward search). Method init(D,r,t,τ) initializes the algorithm D with source node r, target node t, and start time τ, -1 is used if a parameter is not specified. Function settled(D, x) returns 1 if node x has been settled by algorithm D, 0 otherwise.

Input: labeled graph $G = (V, A, \Sigma)$, source r, target t, starting times τ^i and τ^o, regular languages R^o_1, R^o_2, R^i_1, and R^i_2

```
function 2-WAY-PATH-SEARCH($G, r, t, \tau^i, \tau^o, R^o_1, R^o_2, R^i_1, R^i_2$)
                                        ▷ create and initialize algos calculating outgoing path
    $D^o_1 \leftarrow \mathsf{D}_{\mathsf{RegLC}}\ (G, R^o_1, f)$
    $D^o_2 \leftarrow \mathsf{D}_{\mathsf{RegLC}}\ (G, R^o_2, b)$
    init($D^o_1, r, -1, \tau^o$)
    init($D^o_2, t, -1, -1$)

                                        ▷ create and initialize algos calculating incoming path
    $D^i_1 \leftarrow \mathsf{D}_{\mathsf{RegLC}}\ (G, R^i_1, f)$
    $D^i_2 \leftarrow \mathsf{D}_{\mathsf{RegLC}}\ (G, R^i_2, b)$
    init($D^i_1, t, -1, \tau^i$)
    init($D^i_2, r, -1, -1$)
                                   ▷ create algos for re-evaluation of paths (time-dependency)
    $D^o_{re} \leftarrow \mathsf{D}_{\mathsf{RegLC}}\ (G, R^o_2, f)$
    $D^i_{re} \leftarrow \mathsf{D}_{\mathsf{RegLC}}\ (G, R^i_2, f)$

    $\mu \leftarrow \infty$                          ▷ holds cost of current best 2-way path
    $p \leftarrow \emptyset$                              ▷ holds current best 2-way path

    while priority queues of $D^o_1, D^o_2, D^i_1, D^i_2$ are not empty do
        $D \leftarrow$ getAlgoMinNextKey($D^o_1, D^o_2, D^i_1, D^i_2$)
        $x \leftarrow$ extractNextNodeFromPriorityQueue($D$)
        $\delta \leftarrow$ getKey($D, x$)                  ▷ Returns key of node $x$ in algorithm $D$
        if $\mu \leq \delta$ then                                      ▷ Stopping condition
            return

        runOneStep($D$)                 ▷ Settles next node and relaxes outgoing edges

                                                  ▷ check if new 2-way path has been found
        if settled($D^o_1, x$)$\wedge$ settled($D^o_2, x$)$\wedge$ settled($D^i_1, x$)$\wedge$ settled($D^i_2, x$) then

                                                ▷ re-evaluation of paths (time-dependency)
            $\mu_{tmp} = \sum_{D \in \{D^o_1, D^o_2, D^i_1, D^i_2\}}$ getKey($D, x$)           ▷ temporary cost
            if $\mu_{tmp}$ >= $\mu$ then
                continue
            init($D^o_{re}, x, t$,getKey($D^o_1, x$) + $\tau^o_r$)
            tmp$^o_2 \leftarrow$ getCostShortestPath($D^o_{re}$)
            init($D^i_{re}, x, r$,getKey($D^i_1, x$) + $\tau^i_t$)
            tmp$^i_2 \leftarrow$ getCostShortestPath($D^i_{re}$)

            $\mu_{tmp}$ =getKey($D^o_1, x$)+tmp$^o_2$+getKey($D^i_1, x$) + tmp$^i_2$
            if $\mu_{tmp} < \mu$ then                           ▷ new best 2-way path found
                $\mu \leftarrow \mu_{tmp}$
                $p \leftarrow$ getPath($D^o_1$) $\circ$ getPath($D^o_{re}$) $\circ$ getPath($D^i_1$) $\circ$ getPath($D^o_{re}$)
```

6.4 Experimental Results

The algorithm 2-WAY-PATH-SEARCH was implemented in C++ and compiled with GCC 4.1. Experiments are run on a Bi-Xeon quad-core 3 Ghz, 64Gb RAM. We used the multimodal transportation network of Ile-de-France (IDF) as presented in Section 3.3.1. Approximately 20 000 nodes can be used as parking nodes for cars and approximately 220 000 nodes for bicycles. In addition, we ran tests on a smaller graph which only includes roads and public transportation of the city of Paris. We refer to this graph as PARIS. It consists of 137 459 nodes and 505 198 arcs (of which 34 221 are time-dependent). Approximately 3 000 nodes can be used as parking nodes for cars and approximately 25 000 nodes as parking nodes for bicycles.

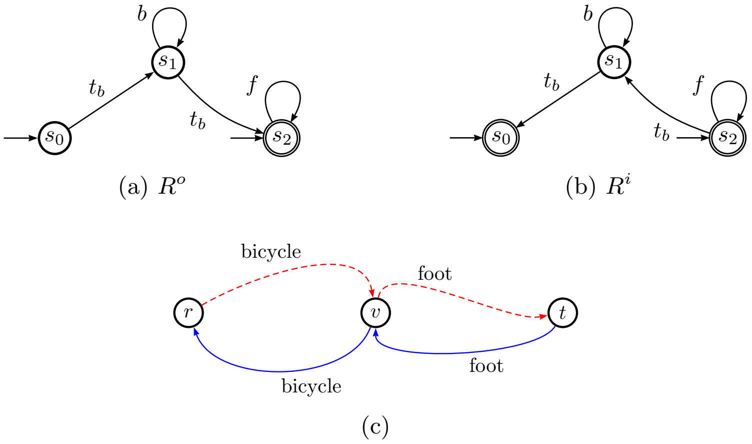
Scenarios

To evaluate runtimes of our algorithm, we generated test instances for five scenarios, see Figures 6.7, 6.8, and 6.9. Source node r and target node t have been determined randomly on the walking network. For each scenario, we created three test sets by varying how to set departure and arrival times of the 2-way path. We set departure and arrival times

(morning/evening) to 9h and 17h, to simulate the commute to work,

(random) to random times in $[0h, 23.59h]$,

(day/night) to times during the night (in $[23h, 5h]$) for incoming path, and times during the day (in $[5h, 23h]$) for outgoing path, or vice versa. This should be the most difficult scenario as public transportation timetables and traffic conditions differ substantially between night and day times.

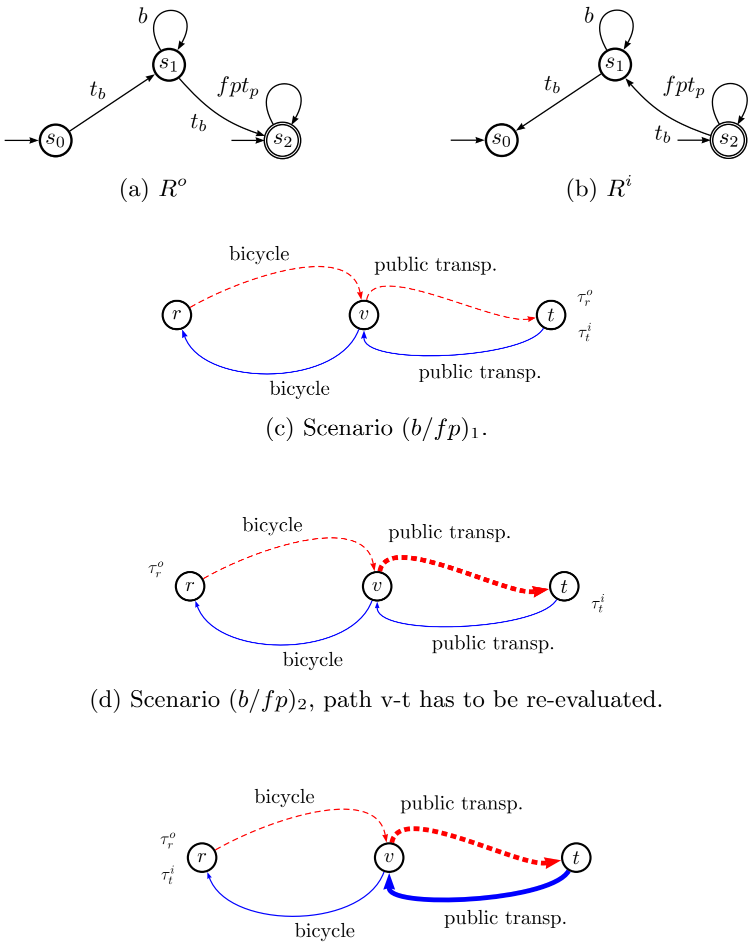


(a) R^o

(b) R^i

(c)

Figure 6.7: Scenario (b/f): The parking may be reached by bicycle, the rest of the journey is done by walking. No time-dependent edges are used in this scenario so departure or arrival times are not relevant.



(a) R^o

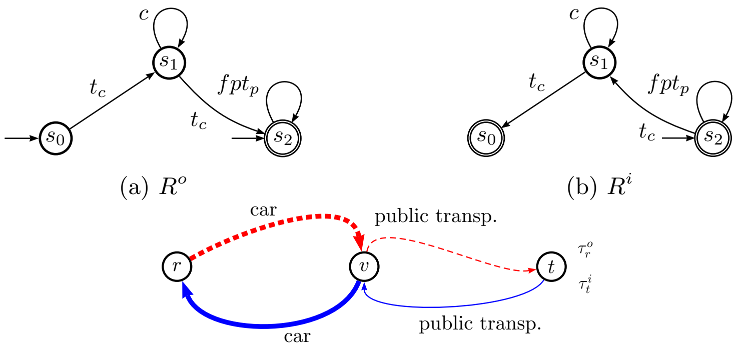
(b) R^i

(c) Scenario $(b/fp)_1$.

(d) Scenario $(b/fp)_2$, path v-t has to be re-evaluated.

(e) Scenario $(b/fp)_3$, paths v-t and t-v have to be re-evaluated.

Figure 6.8: Scenarios $(b/fp)_x$: the parking may be reached by bicycle, the rest of the journey is done either by public transportation or walking. We distinguish three versions. For scenario $(b/fp)_1$ the arrival time τ_t^o at t for the outgoing path and the departure time τ_t^i at t for the incoming path, for scenario $(b/fp)_2$ the departure times τ_r^o, τ_t^i, and for scenario $(b/fp)_3$ the departure time τ_r^o and the arrival time τ_r^i are given. Thick lines indicate paths which have to be re-evaluated.



(a) R^o

(b) R^i

(c) (c/fp), paths r-v and v-r have to be re-evaluated.

Figure 6.9: Scenario (c/fp): the parking may be reached by car, the rest of the journey is done either by public transportation or walking. The arrival time τ_t^o at t for the outgoing path and the departure time τ_t^i at t for the incoming path are given. Thick lines indicate paths which have to be re-evaluated.

Table 6.1: Average runtimes and maximal runtimes (in parenthesis) for scenarios which do not require re-evaluation of parking nodes on the graphs PARIS and IDF. Runtimes are in seconds and are reported for the version *basic* of the algorithm and *basic* including improvement I1.

Graph	Scenario	basic	+I1
PARIS	(b/f)	0.08, (0.13)	0.04, (0.09)
	$(b/fp)_1$	0.33, (1.22)	0.23, (1.16)
IDF	(b/f)	0.73, (1.28)	0.37, (0.73)
	$(b/fp)_1$	1.65, (2.16)	0.78, (1.81)

Results for scenarios without re-evaluation of parking nodes

Table 6.1 reports average runtimes and maximal runtimes (in parenthesis) of the algorithm for scenarios which do not require re-evaluations of parking nodes. We provide results for the basic version of the algorithm `2-WAY-PATH-SEARCH` as well as the version which includes the improved stopping condition (IP1). IP1 improves average runtimes slightly for scenario (b/f) and by approximately factor 2 for scenario $(b/fp)_1$.

Results for scenarios with re-evaluation of parking nodes

Tables 6.2 and 6.3 report average runtimes for scenarios which require re-evaluation of parking nodes on the graphs of Paris (PARIS) and Ile-de-France (IDF), respectively. We provide results for the basic version of the algorithm `2-WAY-PATH-SEARCH` as well as relevant versions where we enhanced the basic version with combinations of the improvements which have been discussed in Section 6.3. Note that a10% indicates that approximation of a factor of 10% has been applied. In parenthesis, we report maximal runtimes, the number of instances with incorrect solutions (e), and the maximal deviation from the optimal solution (m). Values in italic indicate runtimes of the different versions of `2-WAY-PATH-SEARCH` which include `SDALT` (`bas_ls`) for the re-evaluation of parking nodes. For improvement (I5), we used 4 processes in parallel for the re-evaluations.

Average runtimes of the basic version are very high, especially if the number of parking nodes to be re-evaluated is high. Improvements I1, I2, I3, I5, and I7 prove to be very effective. In general, instances of scenarios $(b/fp)_2$ and $(b/fp)_3$ seem to be more difficult to be solved than instances of scenario (c/fp). This can be explained by the data provided in Table 6.4. They summarizes the gaps between lower and upper bounds for the test instances of the three scenarios. We found that in scenario (c/fp) the car almost always dominates public transportation. Consequently, the parking node is often located near the destination. Lower and upper bound coincide in many cases, or only a few parking nodes have to re-evaluated, and so the algorithm stops early. This is also often the case for scenarios $(b/fp)_2$ and $(b/fp)_3$, but less frequently. Generally, for short trips, the bicycle dominates public transportation, and for long trips public transportation dominates the bicycle. Interesting for us, becaue more difficult to solve, are those cases where the two modes of transportation complement each other or when one mode of transportation dominates one path and another mode of transportation the other path (incoming or outgoing). This is often the case for scenarios $(b/fp)_2$ and $(b/fp)_3$ where start times have been set (random) and (night/day). If the departure time is during the day, for long trips, public transportation generally will dominate the bicycles, but during the night, even for long trips the bicycle will dominate public transportation.

Improvement I4 has a positive impact on runtimes, speed-ups up to factor 2 can be observed for tests on the graph IDF. Overall, the impact on runtimes of SDALT and parallelization (I5) is as expected. They improve runtimes on average by a factor 2. SDALT is more efficient for scenario (c/fp) as it includes the car which dominates public transportation. SDALT works better for road networks than for public transportation networks. In some rare cases the application of SDALT on networks including public transportation can increase runtime. Note that parallelization is only applied to scenarios which require two re-evaluations per parking node, i.e., scenarios (c/fp) and $(b/fp)_3$.

Figure 6.10 reports the test results for $(b/fp)_3$ (night/day) on the network PARIS in more detail. It can be observed that 59 *difficult* instances require 98% of the total runtime for the basic algorithm and over 2 000 re-evaluations. The maximal number of parking nodes to be re-evaluated is 25 400. Applying I1, I2, I3, I4, and I7 improves the maximal runtime over all instances by a factor 4 and the maximal number of re-evaluations required is reduced to 12 206. Average runtime drops from 66s to about 10s, but unfortunately the maximal runtime is still 171s and 14 *difficult* instances still require 90% of runtime. By applying SDALT, parallelization, and an approximation of 10%, average and maximal runtime are reduced to 1.40s and 42s, respectively. In general, we observed that in all test sets, runtime is dominated by a few difficult instances, while the majority of instances can be handled in reasonable time.

Scenario $(b/fp)_3$ proves to be challenging. Maximal runtimes for the network PARIS still reach 42s even when parallelization and approximation is applied. For the larger network IDF results are worse, with average runtimes exceeding 300s. To solve these instances in reasonable times future research is required, such as the investigation of stronger stopping conditions and of techniques to further decrease the number of parking nodes which have to be re-evaluated. Average runtimes for scenario $(b/fp)_2$ are around 2s on the network IDF. For this scenario only one re-evaluation per parking node is required, which can be done efficiently by applying improvement I7. However, maximal runtimes still reach 20s.

Nevertheless, we are able to report faster runtimes than those reported by the authors of [24]. Runtimes of their brute force algorithm are quite high even when limiting the number of possible parking nodes. They report average runtimes of 60s when considering 20 parking nodes and of 900s for 80 parking nodes (on a Pentium M, 1.86 Ghz). Note that we work on a considerably larger graph than [24].

6.5 Summary

In this chapter, we presented the algorithm 2-WAY-PATH-SEARCH which is able to solve the 2-way multi-modal shortest path problem. We proposed a basic version of the algorithm and a series of improvements (including the application of SDALT) and presented experimental results on a realistic multi-modal transportation network. We were able to report better runtimes than those reported in the literature. The majority of the instances can be solved in a few seconds of calculation time. Future reseach directions include the investigation of stronger stopping conditions and of techniques to further decrease the number of parking nodes which have to be re-evaluated.

Table 6.2: Average runtimes in seconds of different versions of algorithm 2-WAY-PATH-SEARCH on the network PARIS. In parenthesis maximal runtime. (a10%) indicates that approximation has been applied, (e) gives the number of instances with wrong results and (m) the maximal gap between calculated and optimal solution. Values in italic indicate that SDALT has been applied. → and ↔ indicate that one and two re-evaluation of parking nodes are required, respectively.

Scenario		basic	+I1+I2+I3+I7	+I1+I2+I3+I7	+I1+I2+I3+I7		+I1+I2+I3+I7	+I1+I2+I3+I7	
				+I4	+I4+a10%		+I4+I5	+I4+I5+a10%	
(morning/evening)									
$(b/fp)_2$	→	5.89, (59)	0.27, (1.04)	0.24, (0.76)	0.22, (0.42)	(e: 1, m: 1%)	-	-	-
		1.05, (19)	*0.35, (1.16)*	*0.35, (1.09)*	*0.27, (0.57)*	*(e: 1, m: 1%)*	-	-	-
$(b/fp)_3$	↔	32, (235)	1.02, (41)	1.01, (41)	0.22, (0.45)	(e: 2, m: 1%)	0.49, (12)	0.23, (0.47)	(e: 2, m: 1%)
		9.37, (101)	*0.58, (10)*	*0.69, (10)*	*0.27, (0.63)*	*(e: 2, m: 1%)*	*0.31, (4.63)*	*0.21, (0.48)*	*(e: 2, m: 1%)*
(c/fp)	↔	0.51, (1.82)	0.15, (0.57)	0.15, (0.57)	0.15, (0.57)	(e: 1, m: 2%)	0.17, (0.57)	0.14, (0.57)	(e: 1, m: 2%)
		0.15, (0.70)	*0.10, (0.62)*	*0.12, (0.72)*	*0.10, (0.61)*	*(e: 1, m: 2%)*	*0.10, (0.59)*	*0.09, (0.67)*	*(e: 1, m: 2%)*
(random)									
$(b/fp)_2$	→	15, (928)	0.38, (4.10)	0.32, (2.20)	0.23, (2.05)	(e: 5, m: 2%)	-	-	-
		18, (1 661)	*0.47, (4.61)*	*0.43, (3.12)*	*0.34, (2.92)*	*(e: 5, m: 2%)*	-	-	-
$(b/fp)_3$	↔	67, (1 197)	3.95, (104)	3.91, (93)	1.00, (32)	(e: 3, m: 8%)	1.60, (31)	0.52, (12)	(e: 3, m: 8%)
		44, (1 704)	*3.12, (83)*	*3.45, (91)*	*1.27, (54)*	*(e: 3, m: 8%)*	*0.85, (14)*	*0.41, (10)*	*(e: 3, m: 8%)*
(c/fp)	↔	1.70, (18)	0.18, (0.56)	0.18, (0.57)	0.16, (0.30)	(e: 0, m: 0%)	0.20, (0.56)	0.16, (0.30)	(e: 0, m: 0%)
		0.60, (4.36)	*0.14, (0.47)*	*0.14, (0.47)*	*0.13, (0.29)*	*(e: 0, m: 0%)*	*0.13, (0.40)*	*0.12, (0.28)*	*(e: 0, m: 0%)*
(night/day)									
$(b/fp)_2$	→	12, (129)	0.47, (3.99)	0.34, (1.35)	0.25, (1.41)	(e: 7, m: 9%)	-	-	-
		5.05, (76)	*0.56, (3.62)*	*0.50, (1.96)*	*0.37, (1.90)*	*(e: 7, m: 7%)*	-	-	-
$(b/fp)_3$	↔	66 (534)	9.62, (177)	10, (171)	4.96, (167)	(e: 2, m: 4%)	3.02, (46)	1.57, (46)	(e: 2, m: 4%)
		33, (437)	*8.14, (212)*	*8.76, (203)*	*5.52, (193)*	*(e: 2, m: 4%)*	*2.22, (41)*	*1.40, (42)*	*(e: 2, m: 4%)*
(c/fp)	↔	1.89, (15)	0.17, (0.65)	0.17, (0.65)	0.16, (0.33)	(e: 0, m: 0%)	0.17, (0.53)	0.16, (0.35)	(e: 0, m: 0%)
		0.66, (8.04)	*0.13, (0.73)*	*0.13, (0.72)*	*0.12, (0.34)*	*(e: 0, m: 0%)*	*0.12, (0.53)*	*0.12, (0.32)*	*(e: 0, m: 0%)*

Table 6.3: Average runtimes in seconds of different versions of algorithm `2-WAY-PATH-SEARCH` on the network IDF. In parenthesis maximal runtime. (a10%) indicates that approximation has been applied, (e) gives the number of instances with wrong results and (m) the maximal gap between calculated and optimal solution. Values in italic indicate that `SDALT` has been applied. → and ↔ indicate that one and two re-evaluation of parking nodes are required, respectively. We do not report detailed results for scenarios $(b/fp)_3$ (random) and $(b/fp)_3$ (night/day) as average runtimes exceed 300s.

Scenario		basic	+I1+I2+I3+I7	+I1+I2+I3+I7	+I1+I2+I3+I7		+I1+I2+I3+I7	+I1+I2+I3+I7	
				+I4	+I4+a10%		+I4+I5	+I4+I5+a10%	
(morning/evening)									
$(b/fp)_2$	→	495, (10 163)	2.68, (28)	2.34, (8.80)	1.88, (3.77)	(e: 5, m: 3%)	-	-	-
		385, (10 570)	*2.51, (26)*	*2.21, (7.31)*	*1.83, (3.65)*	*(e: 5, m: 3%)*	-	-	-
$(b/fp)_3$	↔	2 763, (24 829)	12, (362)	13, (337)	1.69, (5.05)	(e: 3, m: 7%)	5.77, (113)	1.75, (5.42)	(e: 3, m: 7%)
		1 616, (18 501)	*7.47, (183)*	*10, (200)*	*2.30, (4.72)*	*(e: 1, m: 4%)*	*4.05, (50)*	*1.71, (3.44)*	*(e: 1, m: 4%)*
(c/fp)	↔	129, (1 590)	3.15, (7.53)	3.21, (7.45)	3.05, (6.71)	(e: 0, m: 0%)	2.15, (4.83)	2.14, (4.83)	(e: 0, m: 0%)
		50, (673)	*2.29, (5.63)*	*2.07, (5.17)*	*2.14, (5.44)*	*(e: 0, m: 0%)*	*1.92, (4.64)*	*1.94, (4.86)*	*(e: 0, m: 0%)*
(random)									
$(b/fp)_2$	→	906, (13 692)	5.02, (67)	3.30, (21)	2.62, (21)	(e: 10, m: 6%)	-	-	-
		503, (14 506)	*3.71, (41)*	*2.86, (13)*	*2.33, (13)*	*(e: 10, m: 6%)*	-	-	-
(c/fp)	↔	108, (2 129)	3.76, (9.67)	3.33, (8.67)	2.98, (6.29)	(e: 0, m: 0%)	2.86, (6.99)	2.41, (5.53)	(e: 0, m: 0%)
		28, (348)	*2.09, (5.56)*	*2.05, (4.78)*	*1.76, (4.22)*	*(e: 0, m: 0%)*	*1.87, (4.44)*	*2.09, (5.03)*	*(e: 0, m: 0%)*
(night/day)									
$(b/fp)_2$	→	744, (16 043)	5.05, (64)	3.11, (16)	2.34, (16)	(e: 14, m: 4%)	-	-	-
		406, (13 026)	*3.74, (37)*	*2.86, (15)*	*2.18, (14)*	*(e: 14, m: 4%)*	-	-	-
$(b/fp)_3$	↔						222, (2 900)	104, (2 110)	(e: 6, m: 9.8%)
							150, (3 882)	*51, (1 412)*	*(e: 4, m: 3%)*
(c/fp)	↔	80, (1 442)	2.98, (21)	3.30, (19)	2.80, (6.44)	(e: 1, m: 0.03%)	2.28, (9.22)	2.14, (4.58)	(e: 1, m: 0.03%)
		21, (285)	*1.64, (5.74)*	*1.65, (4.91)*	*1.56, (3.87)*	*(e: 1, m: 0.03%)*	*1.66, (4.73)*	*1.59, (3.90)*	*(e: 1, m: 0.03%)*

Table 6.4: Maximal gap (max) between lower bound μ_{lb} and upper bound μ_{ub}. Number of instances with a gap which lies in the specified intervals (Network IDF).

Scenario	max	0%	]0%,1%]	]1%,5%]	]5%,10%]	]10%,20%]	>20%
morning/evening							
$(b/fp)_2$	39min (13%)	79	5	10	4	2	0
$(b/fp)_3$	35min (10%)	86	2	6	5	1	0
(c/fp)	1min (2%)	98	2	0	0	0	0
random							
$(b/fp)_2$	113min (35%)	60	4	9	10	10	7
$(b/fp)_3$	223min (33%)	67	2	5	9	13	4
(c/fp)	1min (2%)	94	5	1	0	0	0
night/day							
$(b/fp)_2$	120min (30%)	57	5	12	10	11	5
$(b/fp)_3$	261min (47%)	54	1	12	9	19	5
(c/fp)	1min (2%)	92	8	0	0	0	0

algo: basic
average runtime: 66s
max runtime: 534s
max #re-evaluations: 25 400

#re-evaluations	#instances	% total runtime	average runtime
2	0	0%	0.0
3-10	1	<1%	0.1
11-99	7	<1%	0.1
11-499	9	<1%	0.2
500-999	9	<1%	1.6
1 000-1 999	15	1.8	8.0
>2 000	59	97.9	110.4

algo: basic+I1+I2+I3+I4+I7
average runtime: 10s
max runtime: 171s
max #re-evaluations: 12 206

#re-evaluations	#instances	%total runtime	average runtime
2	66	1.2%	0.2
3-10	4	<1%	0.4
11-99	3	<1%	0.4
11-499	5	<1%	0.9
500-999	2	1.5%	7.2
1 000-1 999	6	7.0%	11.7
>2 000	14	89.5%	64.1

Figure 6.10: Detail for tests on scenario $(b/fp)_3$ (night/day) on the network PARIS. The tables report number of instances, percentage of runtime with respect to total runtime over all 100 instances, and average runtime in function of the number of re-evaluations of parking nodes.

Chapter 7

Dial-A-Ride

In addition to combining existing modes of transportation in a better way, it is also important to study innovative transportation services to provide better mobility to passengers. The Dial-a-Ride (DAR) system is such a service. It offers passengers the comfort and flexibility of private cars and taxis at a lower cost and higher eco-efficiency by combining similar transportation demands. It is thus in line with the demands of the sustainable development requirement and is already employed in several cities. It works as follows: a fleet of vehicles without fixed routes and schedules carries people from their pick-up point to their delivery point. Pre-specified time windows must be respected, and service levels for passengers as well as operation costs should be optimized. The resulting routing and scheduling problem is $\mathcal{NP}$-hard and can be modeled by a mixed integer linear programming formulation. In this chapter, we discuss a Granular Tabu Search algorithm for the static Dial-a-Ride Problem with the objective of producing good solutions in a short amount of time (less than 1 minute). We evaluate the algorithm on test instances from the literature: For most instances, our results are close to the results of another approach and we report new best solutions for some instances.

7.1 Introduction

A Dial-a-Ride (DAR) system organizes the routing of a fleet of vehicles in order to satisfy transportation demands. Passengers may request the service by calling a central unit. They have to specify their pick-up point, their delivery point, the number of passengers, and some limitations on the service time (e.g., the earliest departure time). The routes and schedules of the vehicles vary depending on the received requests. In the *static* DAR, the passenger asks for the service in advance and the routing of the vehicles is determined before the system starts to operate; in the *dynamic* DAR, the passenger can call during the service time and the routes are updated in real time. A DAR system offers passengers the comfort and flexibility of private cars and taxis at a lower cost. It is suited to serve sparsely populated areas, weak demand periods, or passengers with specific requirements (elderly, disabled). Applications have been reported from several cities, e.g., Bologna [125], Copenhagen [88], Milan [129].

The resulting Dial-a-Ride Problem (DARP) is $\mathcal{NP}$-hard, as it generalizes the Pickup and Delivery Problem with Time Windows (PDPTW). Various solution strategies have been described in the literature and various types of objective functions have been studied. They include the minimization of the number of vehicles used, the mean travel or waiting time of passengers, the maximization of the number of passengers served, and the level of service provided.

Several exact algorithms have been proposed for the single-vehicle case. Among the first were [46, 106, 120]. Multi-vehicle cases have been studied in [54] as well as in [28]. In the latter the author describes a Branch and Cut algorithm. To speed up running times, several heuristics for different versions of DARP have been produced. Parallel insertion algorithms have been presented in [77, 88]. A heuristic for the transportation of handicapped people with a homogeneous fleet of vehicles has been proposed in [75]. The algorithm presented by [125] deals with a heterogeneous fleet of vehicles. The authors of [29] propose a Tabu Search to solve a static DARP with a homogeneous fleet of vehicles. A DARP with a fixed number of vehicles and whose objective function maximizes service quality for passengers is discussed in [129]. The authors of [47] propose a constructive heuristic for the DARP based on the assignment of passengers to vehicles according to a regret function. Moreover, the authors of [48] propose a method for the fleet sizing of a transport service on-demand. The authors of [87] present an insertion-based constructive heuristic and the authors of [78] use a genetic algorithm. Recent approaches using Variable Neighborhood Search and Hybrid Large Neighborhood Search to solve the DARP are presented in [103, 104]. The authors of [101] discuss multi-criteria optimization within a Tabu Search mechanism for the DARP. For a broader overview of existing solution techniques and DARP-variants we refer to [30,102]. In [21] an overview of dynamic DARP-variants can be found.

We address the static DARP with time windows and a fixed fleet of vehicles. Our objective is to maximize the number of passengers served and the quality of service, as well as to minimize overall system cost. There is a growing interest in fast methods for obtaining high quality DARP solutions, since DARPs often occur in a dynamic real-world setting. Therefore, the main contribution of this paper is the development of an efficient and fast heuristic to produce good solutions in a short amount of time (up to 3 minutes). We propose a new Granular Tabu Search which uses information provided by the solution of a simple and useful sub-problem to guide the local search process. This sub-problem provides distance information and clusters of close requests. The idea is that passengers who are close both spatially (in terms of the distance between pick-up and delivery points) and temporally (with respect to time windows) are probably best served by the same vehicle in order to produce good solutions. This intuition has been introduced in [129] to produce good initial solutions. In this paper, we go a step further and exploit this information during the improvement phase of a Granular Tabu Search algorithm in order to limit the local search neighborhood. We compare the results of our new algorithm with the results of a Variable Neighborhood Search (VNS) algorithm presented in [103] and a Genetic Algorithm (GA) presented in [78]. Results are based on instances from [29]. Our algorithm performs well and produces better results than the GA for all instances. In the long run, the VNS performs better than our Granular Tabu Search algorithm, but we are able to report better results on test instances after 60s of optimization time.

This chapter is organized as follows. The problem definition and its mixed integer linear programming formulation are given in Section 7.2. Section 7.3 describes the Granular Tabu Search approach and how it has been applied to solve DARP. Computational results and conclusions close the chapter.

7.2 The dial-a-ride problem

Let $R = \{1, \ldots, n\}$ be a set of requests. Each request i consists of two nodes, i^+ and i^-. A load q_i must be taken from i^+ to i^-. Let $N^+ = \{i^+, i \in R\}$ be the set of pick-up nodes and $N^- = \{i^-, i \in R\}$ be the set of delivery nodes ($N' = N^+ \cup N^-$ and $R' = \{(i^+, i^-) : i \in R\} \cup \{(0, 2n+1)\}$). A positive amount $q_{i^+} = q_i$ is associated with the pick-up node, a negative amount $q_{i^-} = -q_i$ with the delivery node. A time window is also associated with each node, i.e., $[e_{i^+}, l_{i^+}]$ for the pick-up node and $[e_{i^-}, l_{i^-}]$ for the delivery node. The fleet of vehicles is denoted as V and all vehicles have the same capacity Q. Let $G = (N, A)$ be a directed graph with a set of nodes $N = N^+ \cup N^- \cup \{0, 2n+1\}$ and a set of arcs $A = \{(i,j) : i, j \in N, i \neq j\}$. Nodes 0 and $2n+1$ represent the start and end depot, and have time windows $[e_0, l_0]$ and $[e_{2n+1}, l_{2n+1}]$, which denote the earliest departure time and the latest return time at the depots. Travel time t_{ij} is assigned to each arc $(i,j) \in A$. Service time at nodes is d_i. Ride time constraints for passengers have to be respected and may not exceed T^{ride}. Maximal route time for vehicles is bounded by T^{route}.

The problem consists in finding a set of routes starting and ending at the depots 0 and $2n+1$, respectively, such that the objective function is optimized and routing, capacity, and time window constraints are respected. We use the following variables: x_{ij}^v is 1 if vehicle v uses arc (i,j) and 0 otherwise; A_i, B_i, and D_i represent arrival, start of service, and departure time at node $i \in N$; D_0^v and A_{2n+1}^v represent departure and arrival time at the start and end depots for vehicle $v \in V$; variable y_i holds the load of the vehicle after leaving node i.

$$\min f'(s) = \min(\omega_1 \cdot c(s) + \omega_2 \cdot r(s) + \omega_3 \cdot l(s) + \omega_4 \cdot g(s) + \omega_5 \cdot e(s) + \alpha \cdot k(s)) \tag{7.1}$$

subject to

$$\sum_{v \in V} \sum_{j \in N} x_{ij}^v \leq 1 \qquad \forall i \in N^+ \cup \{0\} \tag{7.2}$$

$$\sum_{j \in N} x_{ij}^v - \sum_{j \in N} x_{ji}^v = 0 \qquad \forall v \in V, \forall i \in N' \tag{7.3}$$

$$\sum_{j \in N} x_{i^+j}^v - \sum_{j \in N} x_{ji^-}^v = 0 \qquad \forall v \in V, \forall (i^+, i^-) \in R' \tag{7.4}$$

$$x_{ij}^v (y_i + q_j) \leq y_j \qquad \forall v \in V, \forall (i,j) \in A \tag{7.5}$$

$$q_i \leq y_i \leq Q \qquad \forall i \in N^+ \tag{7.6}$$

$$x_{ij}^v (D_i + t_{ij}) = x_{ij}^v A_j \leq B_j \qquad \forall v \in V, \forall (i,j) \in N' \times N' \tag{7.7}$$

$$x_{0j}^v (D_0^v + t_{0j}) = x_{0j}^v A_j \leq B_j \qquad \forall v \in V, \forall j \in N^+ \tag{7.8}$$

$$x_{i,2n+1}^v (D_i + t_{i,2n+1}) = x_{i,2n+1}^v A_{2n+1}^v \qquad \forall v \in V, \forall i \in N^- \tag{7.9}$$

$$e_i \leq B_i \leq l_i \qquad \forall i \in N' \tag{7.10}$$

$$B_{i^-} - D_{i^+} \leq T^{\mathsf{ride}} \qquad \forall i \in R \tag{7.11}$$

$$A_{2n+1}^v - D_0^v \leq T^{\mathsf{route}} \qquad \forall v \in V \tag{7.12}$$

$R = \{1, \dots, n\}$	set of requests
$V = \{1, \dots, m\}$	set or vehicles
N	set of nodes
$\{i^+, i^-\}$	a transportation request
t_{ij}	travel time for arc (i, j)
Q	vehicle capacity
T^{route}	maximum vehicle route duration
T^{ride}	maximum passenger ride time
T_i^{ride}	ride time of request i
q_i	number of passengers to be picked up at node i
e_i	beginning of time window at node i
l_i	end of time window at node i
d_i	service time at node i
A_i	arrival time at node i
B_i	beginning of service at node i
$D_i = B_i + d_i$	departure time from node i
$w_i = B_i - A_i$	vehicle waiting time at node i
D_0^v	departure time at start depot for vehicle v
A_{2n+1}^v	arrival time at end depot for vehicle v
y_i	load when leaving node i
s	a solution (routing plan)

Table 7.1: Notations.

$$D_0^v \geq e_0, A_{2n+1}^v \leq l_{2n+1} \qquad \forall v \in V \tag{7.13}$$

$$\sum_{v \in V} \sum_{j \in N^+} x_{0j}^v \leq m \tag{7.14}$$

$$x_{ij}^v \in \{0, 1\} \qquad \forall v \in V, \forall (i, j) \in A \tag{7.15}$$

$$D_i \geq 0, A_i \geq 0 \qquad \forall i \in N^+ \cup N^- \tag{7.16}$$

The objective function (7.1) minimizes routing cost $c(s) = \sum_{(i,j) \in A, v \in V} x_{ij}^v t_{ij}$, excess ride time $r(s) = \sum_{i \in R} (B_{i^-} - D_{i^+} - t_{i^+ i^-})$, waiting time of passengers on board $l(s) = \sum_{i \in N'} w_i (y_i - q_i)$, route durations $g(s) = \sum_{v \in V} (A_{2n+1}^v - D_0^v)$, early arrival times at pickup and delivery nodes $e(s) = \sum_{i \in N'} (e_i - A_i)^+$, and finally the number of unserved requests $k(s) = n - \sum_{v \in V, (i,j) \in N^+ \times N} x_{ij}^v$. The notation is summarized in Figure 7.1. This objective function has first been introduced by [78].

The first three groups of constraints (7.2), (7.3), and (7.4) impose that each request is served by at most one vehicle. Constraints (7.5) and (7.6) ensure the feasibility of the loads. Time constraints (7.7), (7.8), (7.9), and (7.10) ensure correct arrival, service, and departure time and (7.11, 7.12) ensure that passenger ride time and bus route duration do not exceed T^{ride} and T^{route}, respectively. Finally, constraint (7.14) limits the number of vehicles which can be used. Note that constraints (7.5) and (7.7) can be linearized as $M(1 - x_{ij}^v) \geq y_i + q_j - y_j$ and $M(1 - x_{ij}^v) \geq D_i + t_{ij} - B_j$, respectively. The same is true for constraints (7.8) and (7.9). These equations are a generalization of the classical TSP sub-tour elimination constraints proposed by [91].

7.3 Solution Framework

We apply a *Granular Tabu Search* (GTS) to solve the DARP. It is based on the well known *Tabu Search* (TS) algorithm which is a memory-based search method introduced by [64]. The TS is able to escape local optima by allowing the objective function to deteriorate and it avoids cyclic moves in the search space by keeping track of recent moves through the use of a memory structure called *tabu list.* The size, contents, and management policies of the tabu list depend on the specific problem and algorithm. To improve the effectiveness of the TS, diversification and intensification strategies are employed.

The Granular Tabu Search is a Tabu Search with a particular focus on the local search phase. It uses a reduced neighborhood (*granular neighborhood*) which ideally should not include moves which are unlikely to belong to good solutions. The size of the granular neighborhood is regulated by a granular threshold. The GTS has first been proposed by [126]. The authors apply the method to the Vehicle Routing Problem (VRP). Based on the assumption that good solutions rarely contain long edges (in terms of travel time), their algorithm explores only edges with a length up to a certain threshold during the local search phase. Whenever no improving solutions can been found for a certain time during execution, the threshold is gradually increased. The authors show that, in their scenario, this approach significantly reduces computation time to find good solutions in comparison to the classical Tabu Search. See [84] for another application of a granular neighborhood in a Variable Neighborhood Search framework to solve the Team Orienting problem.

7.3.1 The granular neighborhood

The neighborhood $N(s)$ of a solution s includes all feasible solutions that can be obtained by applying a single simple move to the current solution s. We consider the following simple moves: move a request from a route into another one, insert an unserved request in a route, or remove a request from a route. More complex transformations can be achieved through sequences of simple moves. Since we are considering a granular neighborhood, the only moves allowed are those with *reduced cost* $\bar{c}_{ij} < T_{\mathsf{Gran}}$, where T_{Gran} is the *granular threshold.* We construct the granular neighborhood by solving an assignment problem, which is based on the value $\bar{D}_{ij}$ and which provides the reduced costs $\bar{c}_{ij}$.

Average departure time $\bar{D}_{ij}$ To measure the spatial and temporal distance between two requests i and j, we introduce the average departure time $\bar{D}_{ij}$. It provides an evaluation of the feasibility and the cost of serving requests i and j by using the same vehicle and by starting at node i^+. The possible sequences of nodes for a vehicle to serve the two requests are: $\Pi^1 = (i^+, i^-, j^+, j^-)$, $\Pi^2 = (i^+, j^+, i^-, j^-)$, and $\Pi^3 = (i^+, j^+, j^-, i^-)$ (see Figure 7.1).

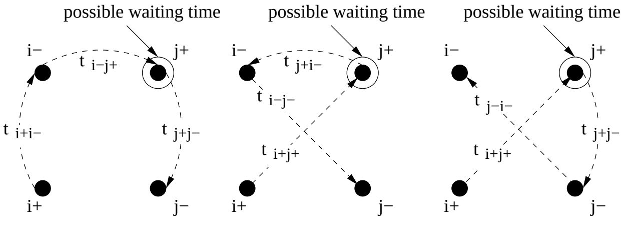


Figure 7.1: Possible sequences of nodes to serve two requests i and j.

For a generic sequence of s nodes $\Pi = (\pi_1, \dots, \pi_s)$, the departure time at the last node of the sequence (node π_s) can be determined using the following equation [129]

$$D^{\Pi} = D_{\pi_1} + T_{\pi_1,\pi_s} + W_{\pi_1,\pi_s}, \tag{7.17}$$

where T_{π_1,π_s} is the total travel time along the sequence, including service time, and W_{π_1,π_s} is the total waiting time. By applying this equation to the three sequences in Figure 7.1, we obtain:

$$\begin{cases} D^{\Pi_1} = e_{i^+} + d_{i^+} + t_{i^+i^-} + d_{i^-} + t_{i^-j^+} + d_{j^+} + t_{j^+j^-} + w_{j^+} + d_{j^-} \\ D^{\Pi_2} = e_{i^+} + d_{i^+} + t_{i^+j^+} + d_{j^+} + t_{j^+i^-} + d_{i^-} + t_{i^-j^-} + w_{j^+} + d_{j^-} \\ D^{\Pi_3} = e_{i^+} + d_{i^+} + t_{i^+j^+} + d_{j^+} + t_{j^+j^-} + d_{j^-} + t_{j^-i^-} + w_{j^+} + d_{i^-} \end{cases} \tag{7.18}$$

If for a sequence Π time or load constraints at a node are not respected, then we set $D^{\Pi} = \infty$. Note that the vehicle might have to only wait in node j^+ (w_{j^+}), since the sequence starts in i^+ and $e_{i^-} = e_{i^+} + d_{i^+} + t_{i^+i^-}$. Now we define the *average departure time* of the departure times at the last nodes of the sequences Π_1, Π_2, and Π_3 of a vehicle when serving two request i and j as:

$$\bar{D}_{ij} = \sum_{\Pi \in \{\Pi_1,\Pi_2,\Pi_3\}} \frac{D^{\Pi} \cdot k^{\Pi}}{k^{\Pi}}, \tag{7.19}$$

where $k^{\Pi} = 1$, if $D^{\Pi} \neq \infty$, $k^{\Pi} = 0$ otherwise. Note that if $k^{\Pi_1} + k^{\Pi_2} + k^{\Pi_3} = 0$ and thus $\bar{D}_{ij} = \infty$, it is not feasible to serve request i and request j with the same vehicle (by starting from i^+). Note also that in general $\bar{D}_{ij} \neq \bar{D}_{ji}$.

The assignment problem Our goal is to construct clusters of requests which are close in respect to $\bar{D}_{ij}$. We first define an auxiliary graph $\hat{G} = (\hat{R}, \hat{A})$ with a set of nodes $\hat{R} = R \cup \{n+1, ..., n+m\}$ consisting of n nodes representing requests and m nodes representing the available vehicles. $\hat{A} = \{(i,j) : i,j \in \hat{R}, i \neq j\}$ represents the set of arcs. Arc weights $\hat{D}_{ij}$ are assigned as follows:

$$\hat{D}_{ij} = \begin{cases} \bar{D}_{ij} & \forall i,j \in R \\ \infty & \forall i,j \in V \\ e_0 + t_{0j^+} + w_{j^+} + d_{j^+} & \forall (i,j) \in V \times R \\ l_{2n+1} - e_{i^-} - t_{i^-,2n+1} - d_{i^-} & \forall (i,j) \in R \times V \end{cases} \tag{7.20}$$

with $w_{j^+} = \max(e_{j^+} - e_0 - t_{0j^+}, 0)$.

Next we define and solve an assignment problem on $\hat{G}$. The time window constraints (7.7), (7.8), (7.9), and (7.10) are relaxed. However, they are partially enforced when calculating $\bar{D}_{ij}$. Constraints (7.2), (7.3), and (7.4) have to be taken into account. Constraints (7.11), (7.12), (7.5), and (7.6) are relaxed. Constraints (7.3), (7.4), and (7.14) are replaced by the classical assignment constraints. The problem to be solved becomes a standard assignment problem of size $n + m$:

$$\min \sum_{(i,j)\in \hat{A}} \hat{D}_{ij} x_{ij} \tag{7.21}$$

$$\sum_{j\in \hat{R}} x_{ji} = 1 \qquad \forall i \in \hat{R} \tag{7.22}$$

$$\sum_{i\in \hat{R}} x_{ji} = 1 \qquad \forall j \in \hat{R} \tag{7.23}$$

$$x_{ij} \in \{0,1\} \qquad \forall (i,j) \in \hat{A} \tag{7.24}$$

The solution of the assignment problem is a set of clusters which contain a sequence of requests. Some of these clusters contain a vehicle (maximal one) while the rest of them, called *sub-tours*, are composed by requests only. The solution of the assignment problem can be exploited in two ways. First, it can be used to quickly construct an initial solution, as shown in [129]. Second, the assignment problem provides the reduced cost value for each arc (i,j) calculated as $\bar{c}_{ij} = \hat{D}_{ij} - u_i - v_j$ where u_i and v_j are the dual variables of constraints (7.22) and (7.23). The reduced cost indicates the impact on the cost of the solution when replacing edges which are part of the current solution with edges outside the solution.

Note that several possible solutions with similar cost exist, because there are at least $n+m-1$ arcs with reduced cost equal to 0, but which are not used in the optimal solution. In other words, the solution of the assignment problem proposes the way in which requests in the same cluster may be served by the same vehicle. Nevertheless, there are pairs of requests which in the current solution are placed in different clusters, but which could be served by the same vehicle without increasing the cost of the solution too much.

Let us consider a small example instance consisting of four requests $\{1,2,3,4\}$ which can be served by two vehicles. A possible feasible solution is the following: requests 1 and 2 are served by vehicle (A) and requests 3 and 4 are served by vehicle (B) (see Figure 7.2). The graph $\hat{G} = (\hat{R}, \hat{A})$ associated with this solution is reported in Figure 7.3.

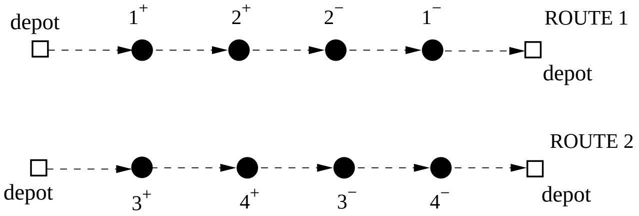


Figure 7.2: The complete initial solution.

Each arc of this graph represents two possible moves, e.g, arc $(3,2)$ suggests to move requests 3 to the vehicle serving request 2 or to move request 2 to the vehicle serving 3. The resulting two new solutions would be $(d,4,d),(d,1,3,2,d)$ and $(d,1,d)$, $(d,3,2,4,d)$. In a similar way, by applying the moves represented by arc $(2,3)$, the two new solutions would be $(d,4,d)$, $(d,1,2,3,d)$ and $(d,1,d)$, $(d,2,3,4,d)$. Note that the difference between considering arc $(3,2)$ and $(2,3)$ is the position of the pick-up nodes of requests 3 and 2.

Now let us suppose that the table in Figure 7.4 represents the reduced cost matrix given by the solution of the assignment problem. We are interested in low reduced costs, as we suppose that they indicate the most promising moves. In order to limit the number of moves,

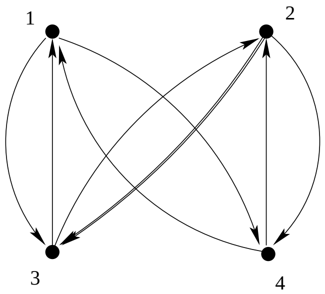


Figure 7.3: The graph $\hat{G} = (\hat{R}, \hat{A})$ associated with the solution $(d, 1, 2, d)$, $(d, 3, 4, d)$, without nodes representing vehicles.

we define a granular threshold T_{Gran} and ignore moves with reduced cost $\bar{c}_{ij} > T_{\mathsf{Gran}}$. See Figure 7.5.

	Bus A	Bus B	1	2	3	4
Bus A	╲	╲	▨ 0	0	0	1097
Bus B	╲	╲	0	0	▨ 0	1097
1	902	902	╲	▨ 0	17	374
2	▨ 0	0	233	╲	0	104
3	0	0	╲	╲	╲	▨ 0
4	0	▨ 0	╲	╲	╲	╲

▨ Arcs used in the solution

╲ Values close to infinity

Figure 7.4: Matrix of reduced costs given by the assignment solution.

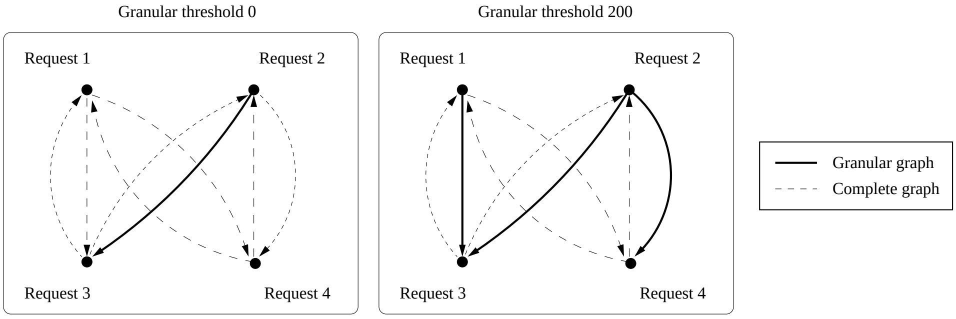


Figure 7.5: Allowed moves by varying the granular threshold (bold).

7.3.2 Preprocessing

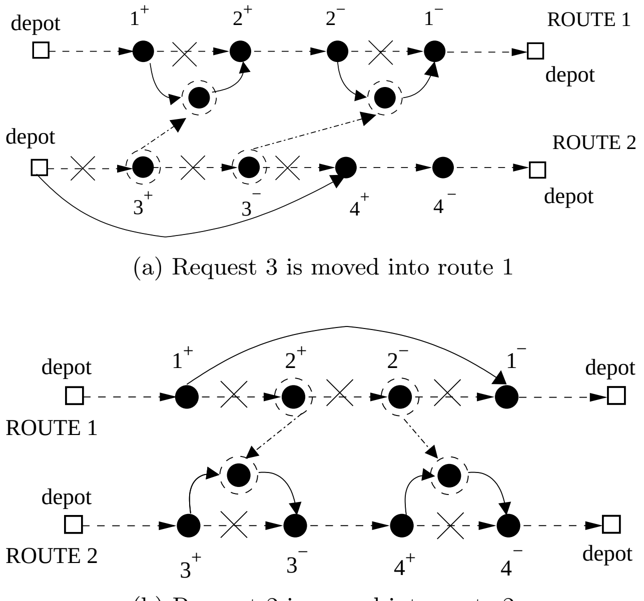
We apply graph pruning and time window tightening techniques as described in [29] before the optimization procedure is started. First, time windows are tightened. In case of an outbound request, the time window at the pick-up node i^+ is set to $e_{i^+} = \max(0, e_{i^-} - T^{\mathsf{trip}} - d_{i^+})$ and $l_{i^+} = \min(l_{i^-} - t_{i^+,i^-} - d_{i^+}, H)$, where H is the end of the planning horizon. In case of an inbound request, the time window at delivery node i^- is set to $e_{i^-} = e_{i^+} + d_{i^+} + t_{i^+i^-}$ and $l_{i^-} = \min(l_{i^+} + d_{i^+} + T^{\mathsf{trip}}, H)$. Then we assign a very high value in the distance matrix to all the arcs which cannot be part of a feasible solution: arcs which connect start and end depots with delivery and pick-up nodes, respectively, arcs which connect delivery nodes with its pick-up nodes, and arcs connecting nodes which are incompatible regarding their time-windows and travel time.

7.3.3 Initial Solution

The solution of the assignment problem can be used to construct a feasible initial solution for the Granular Tabu Search as first discussed in [129]. The assignment problem produces clusters of close requests. Some clusters include a vehicle, others do not. We first set all requests which belong to clusters which have no vehicle as unserved. For each cluster which includes a vehicle, we apply a simple step-by-step procedure to produce a feasible route. Initially the route consists only of the depot. Then, for each request of the cluster, the procedure attempts to insert the request into the route. If this fails, the request is set as unserved. Lastly, the procedure tries to insert the unserved requests into any of the feasible routes that have been produced.

7.3.4 Local search

Each iteration of the Granular Tabu Search optimization process requires the evaluation of the whole granular neighborhood. The local search algorithm executes exactly one simple move and explores all the feasible positions in a route where the request can be inserted. In the worst case the complexity is $O(n)$. See Figure 7.6 for an example of generated solutions when considering arc $(3, 2)$ in the reduced cost matrix.



(a) Request 3 is moved into route 1

(b) Request 2 is moved into route 2

Figure 7.6: Two possible solutions induced by arc $(3, 2)$.

To evaluate a route, we use an adapted version of the eight-step evaluation scheme introduced by [29], see Table 7.2. The difference lies in the fact that their algorithm also considers non-feasible solutions by penalizing constraint violations. In our Tabu Search algorithm we only consider feasible solutions. The eight-step evaluation scheme uses the forward time slack F_i for a node $i \in N$ defined by [115], adapted to the DARP:

```
1. Set $D_0 := e_0$
2. Compute $A_i$, $w_i$, $B_i$, $D_i$, $y_i$ for each node $i$ along the route
   if some $B_i > l_i$ or $y_i > Q$, return false
3. Compute $F_0$
4. Set $D_0 := e_0 + \min\{F_0, \sum_{0<p<q} w_p\}$
5. Update $A_i$, $w_i$, $B_i$ and $D_i$
6. Compute all $T_i^{\mathsf{ride}}$
   if all $T_i^{\mathsf{ride}} < T^{\mathsf{ride}}$ return true
7. for every node $j$ that is an origin
   a) Compute $F_j$
   b) set $w_j := w_j + \min(F_j, \sum_{j<p<q} w_p)$; $B_j := A_j + w_j$; $D_j := B_j + d_j$
   c) Update $A_i$, $w_i$, $B_i$ and $D_i$ for each node $i$ that comes after $j$ in the route
   d) Update $T_i^{\mathsf{ride}}$ for each request $i$ whose destination is after $j$
   e) If all $T_i^{\mathsf{ride}} \leq T^{\mathsf{ride}}$ of requests whose destinations lie after $j$, return true
8. return false
```

Table 7.2: Eight-step evaluation scheme.

$$F_i = \min_{i \leq j \leq q} \Big(\sum_{i<p\leq j} w_p + (\min(l_j - B_j, T^{\mathsf{trip}} - P_j))^+ \Big).$$

w_p denotes the waiting time at node p, q is the last node on the route, and P_j the ride time of the passenger whose destination is $j^- \in N^-$ given that j^- is visited before i on the route; $P_j = 0$ for all other j. F_i gives the maximum amount of time by which the departure from a node i can be delayed without violating time windows and passenger ride time.

7.3.5 Tabu list and aspiration criteria

In a TS framework, the search is guided by a short term memory called tabu list. It is used to avoid cycling and to help the search process to escape local minima. At each iteration the executed move is declared *tabu* for a given number of iterations (*tabu tenure Tt*), which forbids its reversal (see [63]). Moves involving arcs which are tabu can be executed only if they lead to a solution whose objective function value is better than the best solution found so far: this is called *aspiration criterion.* Our algorithm declares all those edges as tabu which are involved when moving a request. For example, Figure 7.7 shows which arcs become tabu after the removal of request j from a route. Note that the number of arcs to be inserted in the tabu list depends on the position of the pickup and delivery node in the route.

7.3.6 Diversification and intensification

Diversification and intensification strategies serve to improve the effectiveness of the Tabu Search method [63]. During intensification the search focuses on promising portions of the solutions space, while diversification moves the algorithm to other unexplored regions. Three diversification strategies have been used in this work: dynamic variation of tabu tenure, frequency-based penalization, and the variation of the granularity threshold.

Dynamic variation of Tabu tenure The *Tabu tenure Tt* can remain constant or vary according to several strategies. In our algorithm the variation is based on the objective

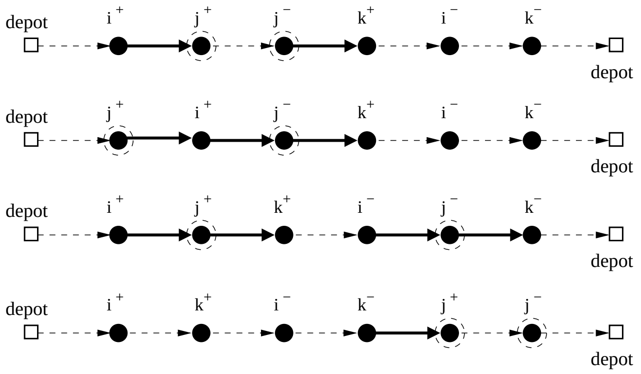


Figure 7.7: The figure shows four routes. From each route, request j is removed. After their removal, the arcs in bold are declared *tabu* and are inserted into the tabu list.

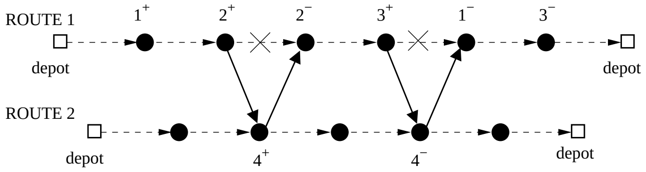


Figure 7.8: Arcs to be saved when a move is executed.

function evolution. If the value of the objective function has improved at least once in Nit consecutive number of iterations, then the search process is probably exploring an interesting portion of the solutions space, thus intensification is required and the Tabu tenure value is reduced. On the other hand, if the objective function value did not improve for Nit iterations, the search process may have reached a local minimum, thus diversification is required and the Tabu tenure value is increased. The function used to increase or to decrease Tt is the following: $Tt = Tt \pm \varepsilon Tt$. The maximal value and the minimal value for Tt are bounded by $Tt = Tt + \vartheta Tt$ and $Tt = Tt - \vartheta Tt$, respectively.

Frequency-based penalization Frequency-based penalization uses a long-term memory for recording the number of times an arc appeared in the incumbent solution. Let s be the current solution then each solution $\bar{s} \in N(s)$ such that $f(\bar{s}) > f(s)$ is penalized by a factor $p(s) = \lambda\rho\sqrt{n \cdot m} f(\bar{s})$. ρ gives the mean value of the number of times the considered arc has been added to the current solution, λ is a parameter used to control the intensity of the diversification, and $\sqrt{n \cdot m}$ is a scaling factor required to adjust the penalties with respect to the problem size. This strategy has been proposed by [124] and successfully applied in many other Tabu Search algorithms for vehicle routing problems, see [29].

Variation of Granularity threshold The granularization process intensifies the search by reducing the neighborhood search space. Several diversification strategies can be defined by ways how to change dynamically granularity threshold T_{Gran} and thus the neighborhood size. We use the following strategy. At the beginning $T_{\mathsf{Gran}} = 0$. T_{Gran} is increased, i.e., $T_{\mathsf{Gran}} = T_{\mathsf{Gran}} + \delta$, when at least one of the following conditions are verified: (a) all the feasible solutions in the current neighborhood are tabu or (b) the algorithm is unable to improve the best solution found so far for a fixed number of iterations (FIt). T_{Gran} is set

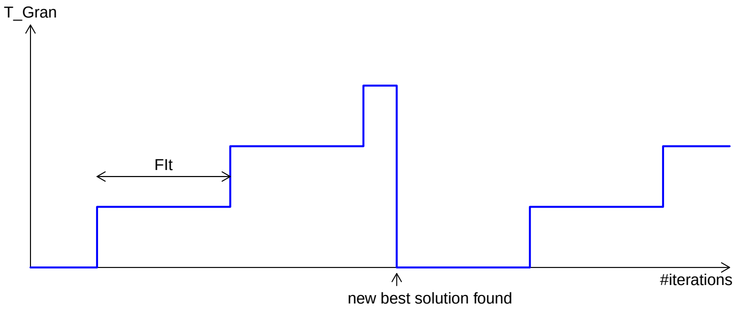


Figure 7.9: Variation of granularity threshold T_{Gran}.

to 0 again when the algorithm improves the best solution found so far. δ is defined as $\delta = \gamma \left(\max_{(ij)\in\hat{A}} \bar{c}_{ij} - \min_{(ij)\in\hat{A}} \bar{c}_{ij} \right)$ (see Figure 7.9).

7.3.7 Stopping criterion

We applied two simple stopping criteria. We set a run time limit and a maximum number of consecutive iterations (4000) during which the best global optimal solution did not improve.

7.4 Experimental Results

The Granular Tabu Search was implemented in C++. All experiments were conducted on a Bi AMD Opteron Dual Core computer with 3.2 GHz. Following guidelines of [29] and results of preliminary testing we used $\lambda = 0.01$, $Tt = 7.5 \log_{10} n$, $Nit = 3$, $\epsilon = 20\%$, and $\vartheta = 70\%$. For the granular threshold we used $\gamma = 10\%$ and $FIt = 40$.

7.4.1 Test instances

[29] produced a data set of 20 randomly generated DARP instances. They contain between 24 and 144 requests; $n/2$ requests were defined as inbound requests while the remaining $n/2$ requests were defined as outbound requests. The service time at pickup and delivery nodes was set to $d_i = 10$. At each node exactly one passenger mounts or leaves the vehicle ($q_i = 1$). Travel times t_{ij} are equal to the Euclidean distance between nodes i and j. Pickup nodes of outbound requests and arrival nodes of inbound requests were assigned a large time window $[0, 1440]$ equal to the length of the planning horizon. The data set was split into two groups. Group (a) was assigned narrow time windows whereas the time windows of group (b) were wider. Furthermore, for instances R1a-R6a and R1b-R6b the number of vehicles was set such that routes are only moderately full; instances R7a-R10a and R7b-R10b might be unfeasible with fewer vehicles. Maximum route duration for all instances was set to $T^{\mathsf{route}} = 480$, vehicle capacity to $Q = 6$, and maximum passenger ride time to $T^{\mathsf{ride}} = 90$.

7.4.2 Evaluation of Granular Tabu Search

A direct comparison with the results of [29] was not possible, as they only minimize routing cost. Therefore, we compared our results with the results of tests conducted on the instances of [29] of a Variable Neighborhood Search algorithm (VNS) reported in [103][1] and a Genetic Algorithm (GA) presented in [78], detailed results can be found in [23]. Note that they only provide results for 13 test instances. Of the remaining seven, two instances were used for parameter tuning and the others were excluded because detailed information about the solution (number of vehicles used, sequence of nodes served by each vehicle, etc.) is not available. Therefore, we also restrict our tests to these 13 test instances. We used the same weights for f' as proposed in [78] and [103]: $\omega_1 = 8$, $\omega_2 = 3$, $\omega_3 = 1$, $\omega_4 = 1$, and $\omega_4 = n$. The coefficient α was set to 10 000. We applied the same CPU time for the optimization process as [103], 3min to 50min depending on the test instance. The main objective of a Granular Tabu Search is to find good solutions in a short amount of time by concentrating on potential good moves ([126]). Therefore, we recorded intermediate values of the objective function every 10s during the optimization process. We compared these values to intermediate values of the objective function of the VNS algorithm presented in [103][2]. Note that all values reported for the VNS and the GA are *average values* over 5 runs. Our algorithm does not include a random component. We tested our Granular Tabu Search which uses the reduced costs (called GTS_RC, hereafter), as well as a second version of the Granular Tabu Search which applies the value $\bar{D}_{ij}$ which we call GTS_P. As discussed above, $\bar{D}_{ij}$ gives a measure of the temporal and spatial closeness of two requests. GTS_P treats moves involving close requests first, i.e., moves with low $\bar{D}_{ij}$. Finally, we also ran a classical Tabu Search (TS) which investigates the entire neighborhood at each step.

7.4.3 Comparison on $f'(s)$

In Figures 7.10 and 7.11 we present the evolution of the values of f' during the first 3 minutes of the computation time. The red, green, violet, and dashed blue lines represent GTS_RC, GTS_P, TS, and VNS, respectively. The continuous blue line marks best known results for each instance which have all been calculated with the VNS. These are reported in [103] and in Table 7.3 (column VNS). See Table 7.4 for the exact values of the objective functions after 30s, 60s, and 180s. Table 7.3 reports the results of VNS, GA, and our Tabu Search variants after long CPU times, depending on the instance between 3 to 50 minutes.

7.4.4 Comparison on $f''(s)$

[103] only reported values for the objective function f', but they gained these values by optimizing a reduced objective function f'' (see Equation 7.25) which includes all factors of $f'(s)$, except excess ride time $e(s)$. They obtained f' at the end of the optimization process by calculating $e(s)$ of the final solution and by adding it to f''. For this reason, we ran a second test set and compared VNS with the Tabu Search variants on objective function f''. We calculated the values for $f''(s)$ of the VNS using data reported in [103]. Unfortunately, this could not be done for the GA, as the necessary data was not available.

$$f''(s) = \omega_1 \cdot c(s) + \omega_2 \cdot r(s) + \omega_3 \cdot l(s) + \omega_4 \cdot g(s) + \alpha \cdot k(s) \tag{7.25}$$

[1] We use updated results which differ slightly from the results originally published in [103]. They are available from http://prolog.univie.ac.at/research/DARP.

[2] Results provided to us by the authors of [103], not included in their paper.

Table 7.3: Results for GA [78], VNS [103], Tabu Search (TS) and Granular Tabu Search (GTS_RC) for objective function f'. CPU indicates CPU time in minutes of the optimization process. CPU times for VNS, TS, GTS_P, and GTS_RC are the same.

	n	m	GA^c	CPU	VNS^a	TS^b	GTS_P^b	GTS_RC^b	CPU
R1a	24	3	4694	5.57	**3152.22**	3167.40	3236.14	3167.40	2.70
R2a	48	5	19426	11.42	14622.40	**6027.29**	6202.98	6130.38	5.16
R3a	72	6	65306	21.58	15985.90	9965.29	**9952.00**	9962.26	6.38
R5a	120	11	213420	58.23	25478.78	13054.80	13716.60	**13050.30**	13.93
R9a	108	8	333283	40.78	**13912.96**	16475.90	16599.90	19449.50	33.53
R10a	144	10	740890	65.98	25791.02	18260.20	**18259.40**	18487.50	40.27
R1b	24	3	4762	5.46	**2875.37**	2908.66	2908.66	2907.18	3.78
R2b	48	5	13580	11.72	5003.11	**4969.28**	5037.06	5004.26	8.29
R5b	120	11	98111	58.93	12373.00	**11747.20**	13241.90	11969.90	23.19
R6b	144	13	185169	81.23	16486.78	**15000.50**	18438.30	15063.50	26.39
R7b	36	4	9169	8.29	4592.52	**4365.98**	4587.97	4401.78	4.49
R9b	108	8	167709	44.66	13433.32	**12265.50**	12487.70	12274.80	30.32
R10b	144	10	474758	66.41	16478.16	**16391.00**	18572.10	16730.80	51.81

[a] Intel Pentium D computer 3.2 GHz, [b] Bi AMD Opteron Dual Core computer 3.2 GHz, [c] Intel Celeron 2 GHz

Again, we present the evolution of the objective function, this time $f''(s)$ (see Figures 7.12 and 7.13, and Table 7.5), and the results after long CPU times, depending on the instance, between 3 to 50 minutes (see Table 7.6). We compare the results of VNS, GTS_RC, GTS_P, and TS.

7.4.5 Discussion

Comparing the results of the Tabu Search variants, it can be observed that almost always GTS_RC produces a good solution in a short amount of CPU time and dominates TS. Therefore, it can be said that the choice of using the reduced cost as an indicator for good moves proves to be efficient. In the long run, as expected, TS performs better, as it explores a larger search space. On the other hand, GTS_P performs poorly. See Figures 7.10, 7.11, 7.12, and 7.13 for detailed results.

By looking at the results gained after 60s of CPU time, we observe that GTS_RC for f' produces better results in most instances than VNS (9 out of 13 instances, Table 7.4). Also when optimizing f'', GTS_RC performs strongly and provides better results than VNS in 6 out of 13 instances (Table 7.5).

Although not the main goal of this work, we provide also results of the algorithms for long CPU times (see Tables 7.3 and 7.6). We observe that when optimizing f' it is important to include excess ride time in the optimisation process and that the results of GA are not competitive. More in detail, for f', TS dominates GTS_RC, which dominates VNS for most instances. TS provides better solutions than VNS for 10 out of 13 instances. The GA is outperformed by all other algorithms. When comparing f'', VNS performs very strongly. Still, TS and GTS_RC provide better solutions than VNS for 2 and 1 instances, respectively, and the results of TS are close to the results of VNS (which are average values over 5 runs).

Table 7.4: Results for VNS [103], Tabu Search (TS) and Granular Tabu Search (GTS_RC) for objective function f', after 30s, 60s, and 180s of CPU time.

			30s				60s				180s			
	n	m	VNS[a]	TS	GTS_P	GTS_RC	VNS[a]	TS	GTS_P	GTS_RC	VNS[a]	TS	GTS_P	GTS_RC
R1a	24	3	**3156.31**	3167.40	3236.14	3184.33	**3156.31**	3167.40	3236.14	3178.78	**3153.48**	3167.40	3236.40	3167.40
R2a	48	5	15159.46	7822.93	**6333.01**	6480.51	15309.90	6733.59	**6333.01**	6346.82	13024.52	6235.93	6230.41	**6130.38**
R3a	72	7	14953.32	**10096.50**	11655.80	10125.20	15570.66	**10014.00**	10734.70	10074.70	14949.00	9965.29	10457.60	**9962.26**
R5a	120	11	29181.62	21868.40	35131.80	**14139.30**	25212.58	16218.30	30012.00	**13844.90**	22042.38	13443.10	19620.40	**13280.50**
R9a	108	8	-[b]	26620.00	37416.00	**20982.10**	**16433.16**	20327.90	25350.30	20982.10	**15635.14**	17949.30	16971.70	20713.80
R10a	144	10	-	24453.20	36708.10	**20987.00**	-	21839.10	35509.00	**20072.40**	24771.76	20376.40	32056.90	**19325.70**
R1b	24	3	**2874.74**	2916.25	2907.18	2955.14	**2874.74**	2916.25	2907.18	2908.66	**2874.74**	2908.66	2907.18	2908.66
R2b	48	5	5141.81	**4986.50**	5861.48	5067.53	5052.65	**4986.50**	5207.70	5049.97	**4945.30**	4986.50	5139.69	5033.23
R5b	120	11	15660.02	22402.80	24552.50	**12755.80**	14914.36	18686.90	23301.60	**12558.30**	13579.22	13372.20	17726.70	**12210.90**
R6b	144	13	22736.96	30036.90	34153.10	**16292.00**	21880.40	26794.60	32110.90	**15865.70**	16563.44	20482.90	30695.90	**15385.50**
R7b	36	4	4615.86	**4387.16**	4587.97	4412.31	4599.47	**4365.98**	4587.97	4412.31	4575.33	**4365.98**	4587.97	4412.31
R9b	108	8	15977.08	17946.80	22661.70	**12839.40**	16408.28	14426.50	21443.70	**12500.60**	13178.64	12585.20	20284.20	**12461.20**
R10b	144	10	**21520.88**	47677.20	54526.70	21972.10	**19625.40**	27785.60	53184.50	20746.10	19646.40	18030.60	47751.30	**17393.90**

[a] Results provided to us by the authors of [103], not included in the original paper. Average values over 5 runs by using a Xeon computer with 2.67Ghz.

[b] A hyphen (-) indicates that no feasible solution has been found yet.

Table 7.5: Results for VNS [103], Tabu Search (TS) and Granular Tabu Search (GTS_RC) for objective functions f'', after 30s, 60s, and 180s of CPU time.

			30s				60s				180s			
	n	m	VNS[a]	TS	GTS_P	GTS_RC	VNS[a]	TS	GTS_P	GTS_RC	VNS[a]	TS	GTS_P	GTS_RC
R1a	24	3	**3122.64**	3146.67	3130.16	3156.57	3122.64	**3118.33**	3119.04	3156.57	3119.81	**3118.33**	**3118.33**	3124.91
R2a	48	5	**5622.86**	5729.25	5811.48	5715.84	**5601.51**	5729.25	5745.08	5663.73	**5511.98**	5629.64	5678.76	5656.67
R3a	72	7	**9913.00**	9984.80	11930.20	10009.20	**9807.72**	9836.21	10460.70	9931.28	**9670.04**	9811.92	9921.22	9931.28
R5a	120	11	14092.86	20122.60	20784.20	**13642.40**	13783.50	15023.20	19646.10	**13400.90**	13328.56	**12990.10**	16200.70	13096.60
R9a	108	8	-[b]	24347.70	26847.30	**20056.50**	**15730.54**	24130.70	20138.10	17955.90	**14399.28**	17171.70	19652.10	17022.30
R10a	144	10	-	24049.10	35899.20	**19569.20**	-	19760.30	34212.20	**19251.60**	19646.14	18775.70	31734.00	**18419.00**
R1b	24	3	**2874.74**	2913.68	2951.45	2974.87	**2874.74**	2913.68	2951.45	2974.87	**2874.74**	2905.51	2951.45	2916.82
R2b	48	5	**4994.35**	5059.61	5724.77	5058.71	**4979.24**	5059.61	5147.37	5058.71	**4945.30**	4975.35	5017.05	5026.13
R5b	120	11	13011.70	21346.40	19325.70	**12523.60**	12732.84	17921.40	18574.80	**12335.30**	12238.24	12869.00	17595.30	**11980.00**
R6b	144	13	16717.76	28620.80	25259.50	**16427.80**	16462.50	25707.00	24549.40	**15817.10**	15992.08	19677.60	23036.00	**15503.20**
R7b	36	4	4325.91	4368.21	4412.07	**4310.21**	4309.52	4368.21	4412.07	**4306.44**	**4283.03**	4299.17	4381.86	4306.44
R9b	108	8	13247.00	17854.80	21491.50	**12910.20**	13030.08	14378.20	21207.50	**12476.50**	12612.86	12530.80	19452.90	**12429.90**
R10b	144	10	**19525.88**	40281.50	54522.30	23679.60	**18848.48**	27147.90	50012.40	23650.40	17968.44	**17631.40**	47617.00	17995.50

[a] Results provided to us by the authors of [103], not included in the original paper. Average values over 5 runs by using a Xeon computer with 2.67Ghz.

[b] A hyphen (-) indicates that no feasible solution has been found yet.

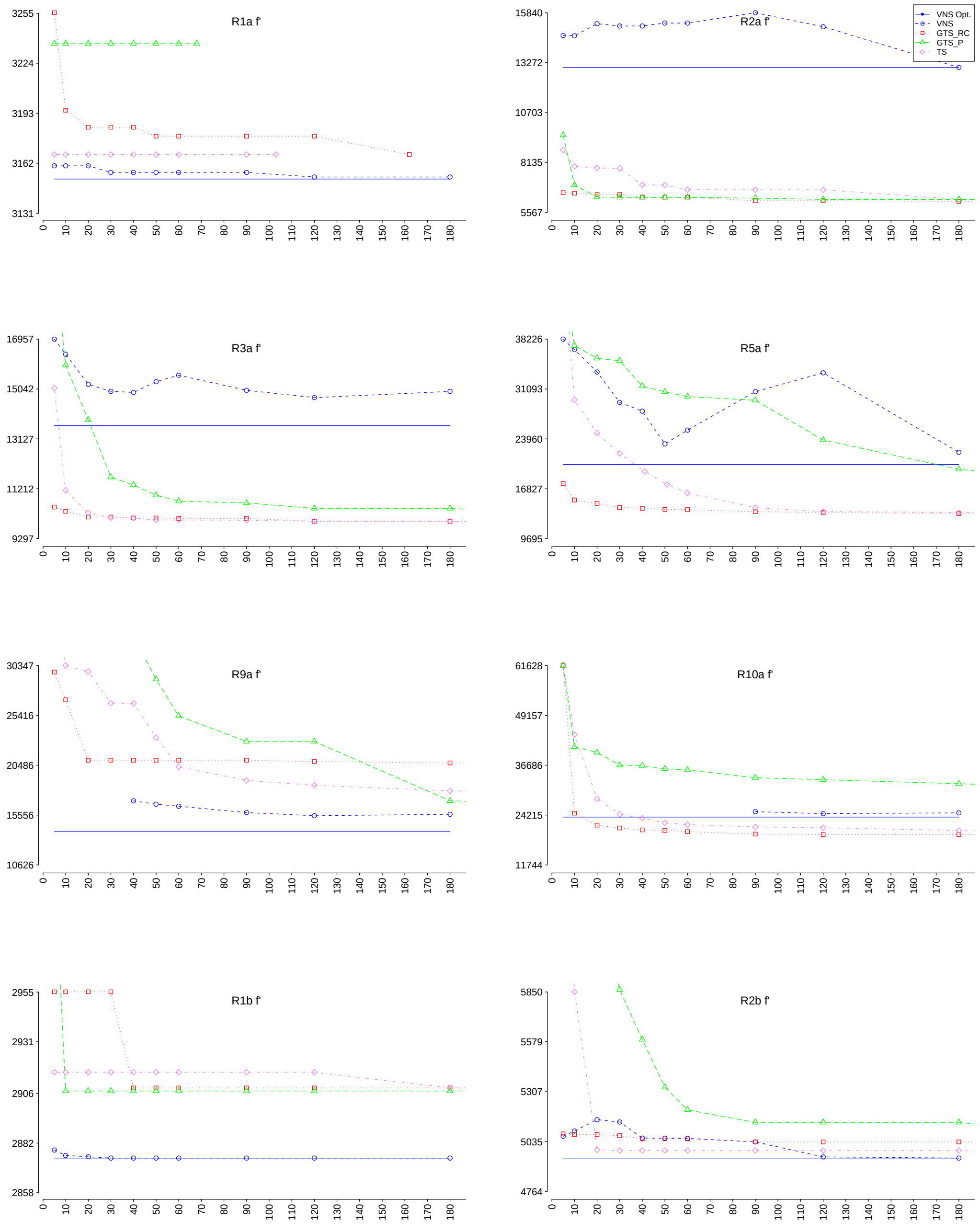


Figure 7.10: Evolution of objective function f' over time (x-axis: CPU time, y-axis: value of f').

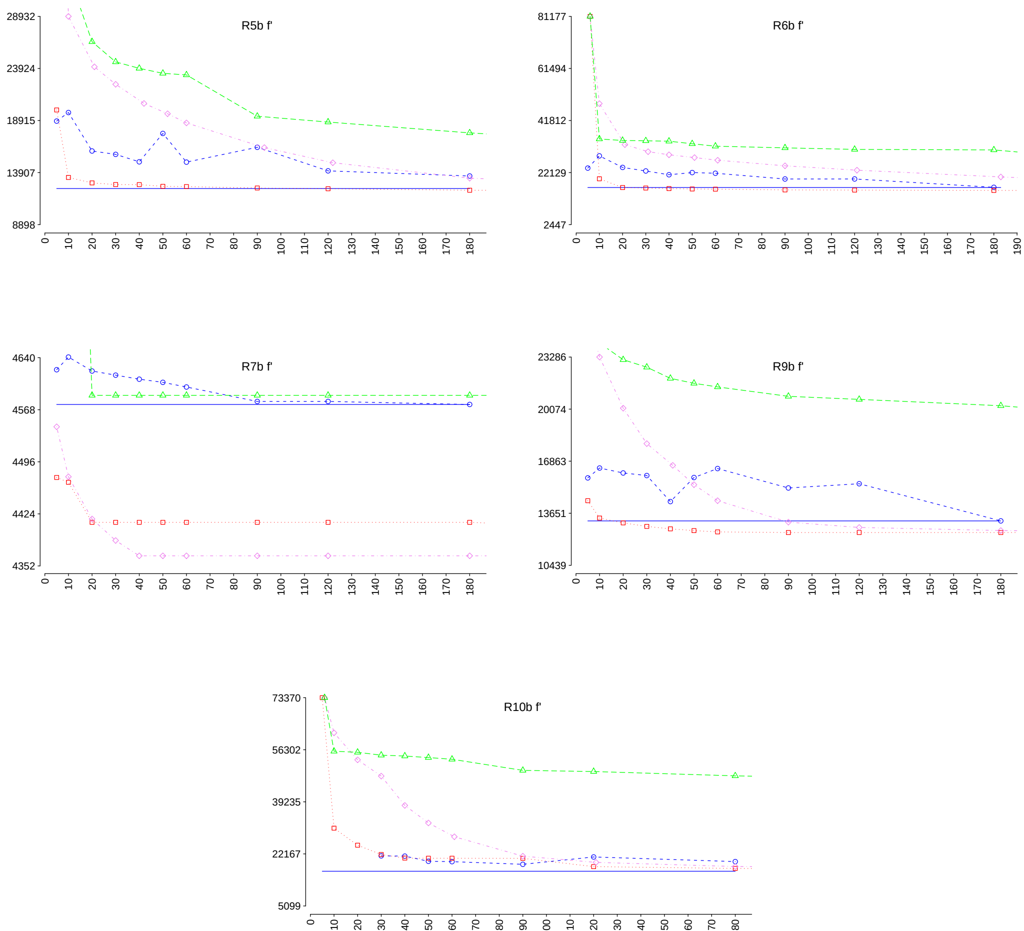


Figure 7.11: Evolution of objective function f' over time (x-axis: CPU time, y-axis: value of f').

Table 7.6: Results for VNS [103], Tabu Search (TS) and Granular Tabu Search (GTS_RC) for objective function f'', CPU indicates CPU time in minutes of the optimization process. CPU times for VNS, TS, GTS_P, and GTS_RC are the same.

	n	m	VNS^a	TS^b	GTS_P^b	GTS_RC^b	CPU
R1a	24	3	3118.56	**3118.33**	**3118.33**	3124.91	2.70
R2a	48	5	**5546.55**	5623.06	5677.85	5656.67	5.16
R3a	72	6	**9632.47**	9775.78	9921.22	9859.95	6.38
R5a	120	11	**12642.77**	12770.50	13338.00	13002.80	13.93
R9a	108	8	**13301.65**	15419.70	19652.10	16789.50	33.53
R10a	144	10	**17459.16**	18295.50	17986.00	18069.20	40.27
R1b	24	3	**2875.36**	2905.51	2932.89	2916.82	3.78
R2b	48	5	**4929.08**	4975.35	5001.14	5020.86	8.29
R5b	120	11	**11635.79**	11731.60	14045.30	11896.10	23.19
R6b	144	13	**14927.10**	14963.80	18138.40	15127.50	26.39
R7b	36	4	4297.89	4299.17	4360.23	**4290.11**	4.49
R9b	108	8	12067.00	**12021.80**	12429.70	12243.10	30.32
R10b	144	10	**16238.27**	16641.80	17786.30	16753.30	51.81

^{a} Pentium D computer 3.2 GHz, ^{b} Bi AMD Opteron Dual Core computer with 3.2 GHz

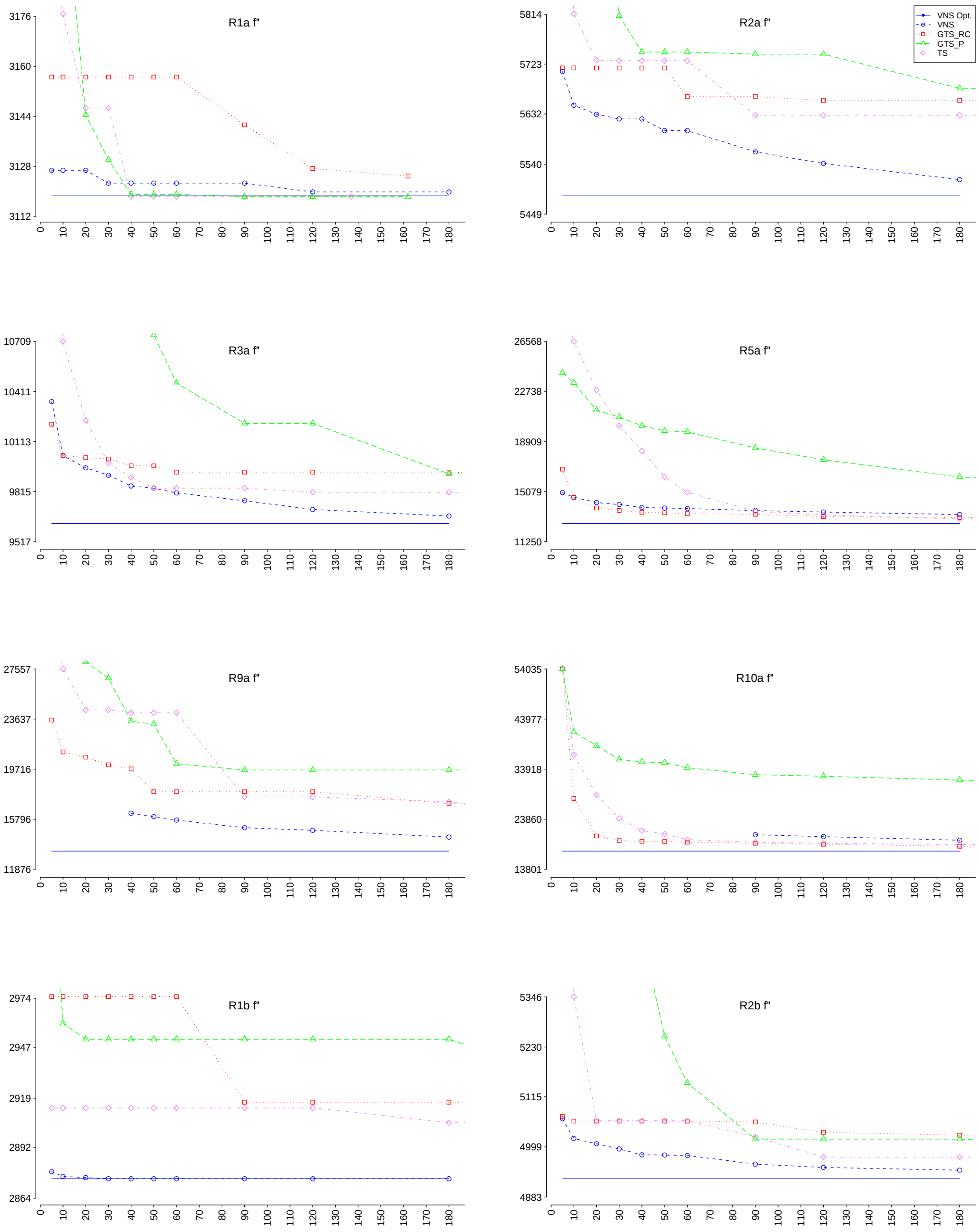


Figure 7.12: Evolution of objective function f'' over time (x-axis: CPU time, y-axis: value of f'').

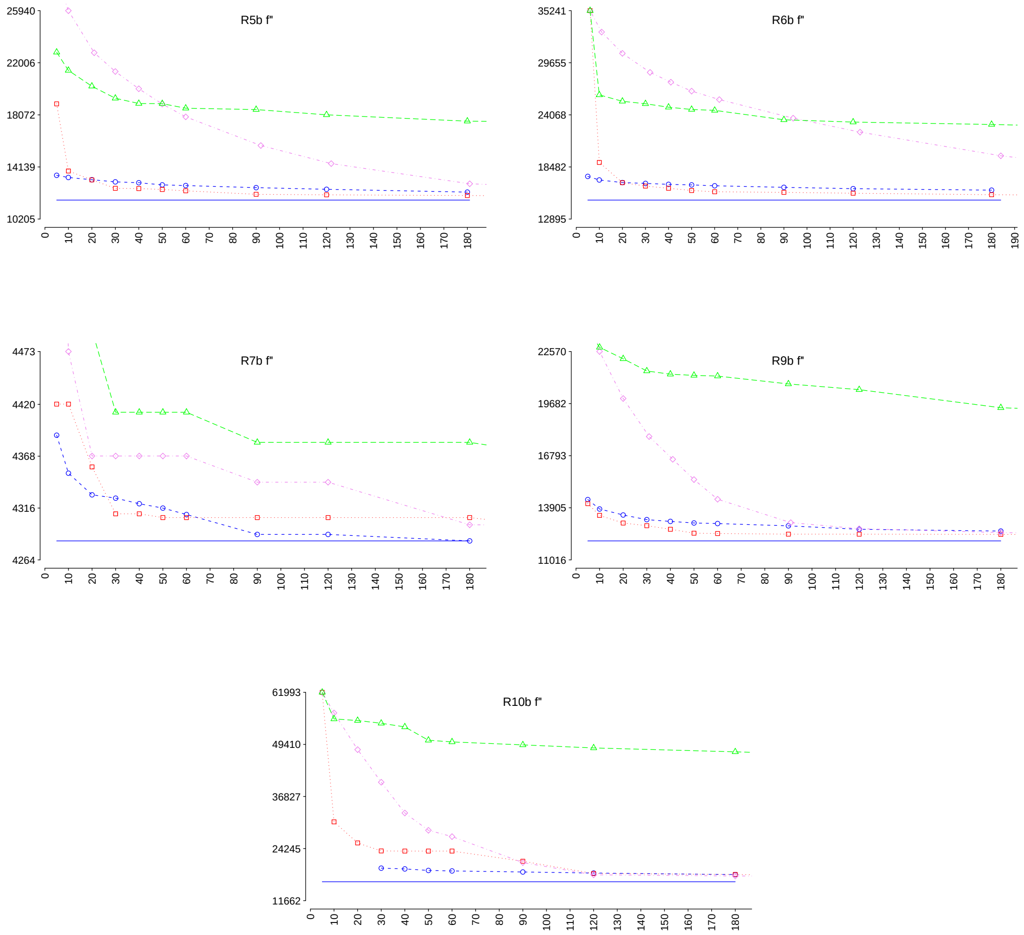


Figure 7.13: Evolution of objective function f'' over time (x-axis: CPU time, y-axis: value of f'').

7.5 Summary

In this paper, a fast algorithm for solving the static *Dial-a-Ride Problem* (DARP) has been proposed. The *Granular Tabu Search* method has been applied for the first time to solve this kind of problem. A major characteristic of the proposed algorithm is how the granularity has been produced: we reduced the original problem to an assignment problem and we exploited the reduced costs to build clusters of close requests. The computational results prove that our algorithm performs well in comparison to other solution methods and that it is able to provide good solutions in the first three minutes of CPU time of the optimization process. Directions for future research include: study of the applicability of the technique presented in this paper to other routing problems and the on-line scenario, as well as the analysis of the time-dependent case (traffic information).

Chapter 8

Conclusions

In this thesis, we discussed multi-modal routing and the Dial-a-Ride system. Both respond to the need of innovative routing services and a more efficient utilization of the available transportation infrastructure, which is an important component of sustainable development. The major contributions of this thesis are:

Network modeling. We showed how to produce a model of a complete transportation network of an inter-urban region by means of a labeled graph. It includes all major modes of transportation, i.e., car and bicycle, walking and public transportation. Furthermore, it includes traffic conditions, road types, and timetable information. We used time-dependent arc costs to include traffic information and timetable information. We produced graphs of the multi-modal transportation networks of the French region Ile-de-France (including the city of Paris) and New York City. To our knowledge, this is the first work to consider a multi-modal network in this configuration and on this scale.

SDALT. We introduced a new algorithm `SDALT` which solves the regular language constrained shortest path problem on a multi-modal network. A generalization of Dijkstra's algorithms may be used to solve this kind of routing problems. However, its performance may not be sufficient for real world applications. For this reason, this thesis introduced `SDALT` which is an adaption of the speed-up technique `ALT`. The experiments show that `SDALT` performs well, with speed-ups of a factor 1.5 to 40 (up to a factor of 60 with approximation) with respect to the basic algorithm.

2-Way Multi-Modal Shortest Path Problem (`2WMMSP`). We introduced the 2-way multi-modal shortest path problem. When using a private vehicle for parts of the outgoing path, it has to be picked up during the incoming path so that it can be taken to the starting location. For this reason, the parking location must be chosen in order to optimize the combined travel times of the outgoing and incoming path. We proposed an efficient algorithm which solves this problem and various improvements including the application of `SDALT`.

Dial-A-Ride problem. We presented a new algorithm to produce in a short time good solutions for the Dial-A-Ride problem. The objective of the Dial-A-Ride problem is to maximize the number of passengers served and the quality of service, as well as to minimize overall system cost. The main contribution here is the development of an efficient and fast heuristic to produce good solutions in a short amount of time (less than 3 minutes). We proposed a new Granular Tabu Search which uses information provided by the solution of a simple and useful sub-problem to guide the local search process. This sub-problem provides distance information and clusters of close requests. The idea is that passengers who are close both spatially (in terms of the distance between pick-up and delivery points) and temporally (with respect to time windows) are probably best served by the same vehicle in order to produce good solutions. Our algorithm produces better results for more than half of the test instances after 60s of optimization time in comparison with other methods.

Future Work

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 and by either limiting the constraints which can be imposed on the shortest paths [109] or by identifying homogeneous regions (arcs with the same label) of the network and by applying contractions only to those regions [50]. Our algorithm SDALT can solve more general routing problems and proves to work well even when considering more difficult constraints than the one considered in [109] and on a highly dis-homogeneous graph. Also, time-dependent cost functions on arcs can be easily incorporated. SDALT provides speed-ups of a factor 1.5 to 60 (up to a factor of 60 with approximation and depending on the constraints) with respect to the basic algorithm $\mathtt{D}_{\mathtt{RegLC}}$. Nevertheless, the objective for future research is to further increase speed-ups. The application of contraction is a viable option, although handling time-dependency and considering the labels on arcs during the contraction is not straightforward. A further area of future research is to study the multi-criteria scenario, where not only travel time but also travel cost, the number of changes, etc., are minimized.

Regarding the algorithm 2-WAY-PATH-SEARCH to solve the 2-way multi-modal shortest path problem (Chapter 6), stronger stopping conditions and techniques to further decrease the number of parking nodes which have to be re-evaluated should be studied. This is necessary to also solve difficult instances in a reasonable time. The application of clustering could assist in the grouping of parking nodes which are close to each other and prevent all these nodes from having to be re-evaluated singularly. A further preprocessing phase which precalculates a limited set of *good* parking nodes for queries with similar source and target nodes could also be useful.

The solutions quality of the Dial-A-Ride algorithm proposed in Chapter 7 could be improved by introducing more complex moves during the local search process. Instead of just evaluating all the positions where one request can be inserted, moves could involve two or more requests. Furthermore, allowing unfeasible solutions (by penalizing the violation of constraints) would probably result in a better exploration of the search space. As the techniques presented in this thesis proved to be quite efficient, it would be interesting to study the applicability to other similar routing problems such as the vehicle routing problem with time windows (VRPTW). The extension of our algorithm to the real-time scenario, where requests arrive during operation time, as well as the analysis of the time-dependent case, where traffic information is considered, are other possible future research directions.

Bibliography

[1] I. Abraham, D. Delling, A. V. Goldberg, and R. F. Werneck. A Hub-Based Labeling Algorithm for Shortest Paths on Road Networks. In R. Klasing, editor, *Proceedings of the 11th International Symposium on Experimental Algorithms (SEA'11)*, volume 6630 of *Lecture Notes in Computer Science*, pages 376–387. Springer, Berlin, 2010.

[2] I. Abraham, D. Delling, A. V. Goldberg, and R. F. F. Werneck. Hierarchical Hub Labelings for Shortest Paths. In L. Epstein and P. Ferragina, editors, *Proceedings of the 20th Annual European Symposium of Algorithms (ESA'12)*, volume 7501 of *Lecture Notes in Computer Science*, pages 24–35. Springer, Berlin, 2012.

[3] I. Abraham, A. Fiat, A. V. Goldberg, and R. Werneck. Highway Dimension, Shortest Paths, and Provably Efficient Algorithms. In M. Charikar, editor, *Proceedings of the 21st Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'10)*, SODA, pages 782–793. SIAM, Philadelphia, 2010.

[4] R. Ahuja, J. Orlin, S. Pallottino, and M. G. Scutellà. Dynamic Shortest Paths Minimizing Travel Times and Costs. *Networks*, 41(4):197–205, 2003.

[5] C. Artigues, M.-J. Huguet, F. Gueye, F. Schettini, and L. Dezou. State-based accelerations and bidirectional search for bi-objective multi modal shortest paths. *Transportation research Part C*, 27:233–259, 2013.

[6] C. L. Barrett, K. R. Bisset, M. Holzer, G. Konjevod, M. Marathe, and D. Wagner. Engineering label-constrained shortest-path algorithms. *The Shortest Path Problem: Ninth Dimacs Implementation Challenge*, 74:309–327, 2009.

[7] C. L. Barrett, K. R. 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. In R. H. Mohring and R. Raman, editors, *European Symposium on Algorithms (ESA)*, volume 2461 of *Lecture Notes in Computer Science*, pages 126–138. Springer, Berlin, 2002.

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

[9] H. Bast. Car or public transport - two worlds. *Efficient Algorithms*, 5760:355–367, 2009.

[10] H. Bast, E. Carlsson, A. Eigenwillig, R. Geisberger, C. Harrelson, V. Raychev, and F. Viger. Fast Routing in Very Large Public Transportation Networks Using

Transfer Patterns. In *Proceedings of the 18th Annual European Symposium on Algorithms (ESA'10)*, volume 6346 of *Lecture Notes in Computer Science*, pages 290–301. Springer, Berlin, 2010.

[11] H. Bast, S. Funke, D. Matijevic, P. Sanders, and D. Schultes. In transit to constant time shortest-path queries in road networks. In *Proceedings of the 9th Workshop on Algorithm Engineering and Experiments (Alenex'07)*, pages 46–59. SIAM, Philadelphia, 2007.

[12] H. Bast, S. Funke, P. Sanders, and D. Schultes. Fast routing in road networks with transit nodes. *Science*, 316(5824):566, 2007.

[13] G. Batz, R. Geisberger, S. Neubauer, and P. Sanders. Time-Dependent Contraction Hierarchies and Approximation. In P. Festa, editor, *Proceedings of the 9th International Symposium on Experimental Algorithms (SEA'10)*, volume 6049 of *Lecture Notes in Computer Science*, pages 166–177. Springer, Berlin, 2010.

[14] G. V. Batz, D. Delling, P. Sanders, and C. Vetter. Time-dependent contraction hierarchies. In I. Finocchi and J. Hershberger, editors, *Proceedings of the 11th Workshop on Algorithm Engineering and Experiments (ALENEX'09)*, ALENEX, pages 97–105. SIAM, Philadelphia, 2009.

[15] R. Bauer and D. Delling. SHARC: Fast and robust unidirectional routing. *Journal of Experimental Algorithmics (JEA)*, 14:2.4–2.29, 2009.

[16] R. Bauer, D. Delling, P. Sanders, D. Schieferdecker, D. Schultes, and D. Wagner. Combining hierarchical and goal-directed speed-up techniques for Dijkstra's algorithm. *ACM Journal of Experimental Algorithmics*, 15:2.3:1–31, Mar. 2010.

[17] R. Bauer, D. Delling, and D. Wagner. Experimental study of speed up techniques for timetable information systems. *Networks*, 57(1):38–52, Jan. 2011.

[18] M. Beckmann, C. B. McGuire, and C. B. Winsten. Studies in the Economics of Transportation. *Technical Report RM-1488*, 1956.

[19] R. Bellman. On a routing problem. *Quarterly of Applied Mathematics*, Paper P-10, 1956.

[20] J. L. Bentley. Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18(9):509–517, Sept. 1975.

[21] G. Berbeglia, J.-F. Cordeau, and G. Laporte. Dynamic pickup and delivery problems. *European Journal of Operational Research*, 202(1):8–15, 2010.

[22] A. Berger, D. Delling, A. Gebhardt, and M. Müller-Hannemann. Accelerating time-dependent multi-criteria timetable information is harder than expected. In J. Clausen and G. D. Stefano, editors, *Proceedings of the 9th Workshop on Algorithmic Approaches for Transportation Modeling, Optimization, and Systems (ATMOS'09)*, volume 12 of *ATMOS*. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, Germany, 2009.

[23] K. B. Bergvinsdottir. *The genetic algorithm for solving the dial-a-ride problem.* Master thesis, Technical University of Denmark, 2004.

[24] A. Bousquet, S. Constans, and E. F. Nour-Eddin. On the adaptation of a label-setting shortest path algorithm for one-way and two-way routing in multimodal urban transport networks. In *International Network Optimisation Conference (INOC'09)*, 2009.

[25] L. S. Buriol, M. G. C. Resende, and M. Thorup. Speeding Up Dynamic Shortest-Path Algorithms. *INFORMS Journal on Computing*, 20(2):191–204, Sept. 2008.

[26] I. Chabini. Discrete dynamic shortest path problems in transportation applications : Complexity and algorithms with optimal run time. *Transportation research records*, 1645:170–175, 1998.

[27] K. Cooke and E. Halsey. The shortest route through a network with time-dependent internodal transit times. *Journal of Mathematical Analysis and Applications*, 14(3):493–498, 1966.

[28] J.-F. Cordeau. A Branch-and-Cut Algorithm for the Dial-a-Ride Problem. *Operations Research*, 54(3):573–586, 2006.

[29] J.-F. Cordeau and G. Laporte. A tabu search heuristic for the static multi-vehicle dial-a-ride problem. *Transportation Research Part B: Methodological*, 37(6):579–594, 2003.

[30] J.-F. Cordeau and G. Laporte. The dial-a-ride problem: models and algorithms. *Annals of Operations Research*, 153(1):29–46, 2007.

[31] G. B. Dantzig. *Linear Programming and Extensions*, volume 4. Princeton University Press, 1962.

[32] B. Dean. *Continuous-time dynamic shortest path algorithms.* Master thesis, Massachussets Institute of Technology, 1999.

[33] D. Delling. Time-dependent SHARC-routing. *Algorithmica*, 60(1):60–94, 2011.

[34] D. Delling, J. Dibbelt, T. Pajor, D. Wagner, and R. F. Werneck. Computing Multimodal Journeys in Practice. In *Proceedings of the 12th International Symposium on Experimental Algorithms (SEA'13)*, Lecture Nodes in Computer Science. Springer, Berlin, 2013.

[35] D. Delling, A. V. Goldberg, T. Pajor, and R. F. Werneck. Customizable Route Planning. In P. M. Pardalos and S. Rebennack, editors, *Proceedings of the 19th International Symposium on Experimental Algorithms (SEA'11)*, volume 6630 of *Lecture Notes in Computer Science*, pages 376–387. Springer, Berlin, 2011.

[36] D. Delling and G. Nannicini. Bidirectional Core-Based Routing in Dynamic Time-Dependent Road Networks. In S.-H. Hong, H. Nagamochi, and T. Fukunaga, editors, *Proceedings of the 19th International Symposium on Algorithms and Computation (ISAAC'08)*, volume 5369 of *Lecture Nodes in Computer Science*, pages 812–823. Springer, Berlin, 2008.

[37] D. Delling, T. Pajor, and D. Wagner. Accelerating multi-modal route planning by access-nodes. In A. Fiat and P. Sanders, editors, *Proceedings of the 17th Annual European Symposium on Algorithms (ESA'09)*, volume 5757 of *Lecture Notes in Computer Science*, pages 587–598. Springer, Berlin, 2009.

[38] D. Delling, T. Pajor, and D. Wagner. Engineering time-expanded graphs for faster timetable information. *Robust and Online Large-Scale Optimization: Models and Techniques for Transportation Systems*, 5868:182–206, 2009.

[39] D. Delling, T. Pajor, and R. Werneck. Round-based public transit routing. In D. A. Bader and P. Mutzel, editors, *Proceedings of the 2012 Meeting on Algorithm Engineering and Experiments (ALENEX'12)*, ALENEX, pages 130–140. SIAM, Philadelphia, 2012.

[40] D. Delling, P. Sanders, D. Schultes, and D. Wagner. Engineering route planning algorithms. In J. Lerner, D. Wagner, and K. A. Zweig, editors, *Algorithmics of Large and Complex Networks - Design, Analysis, and Simulation*, volume 5515 of *Lecture Nodes in Computer Science*, pages 117–139. Springer, Berlin, 2009.

[41] D. Delling, P. Sanders, D. Schultes, and D. Wagner. Highway hierarchies star. In C. Demetrescu, A. V. Goldberg, and D. S. Johnson, editors, *The Shortest Path Problem: Ninth DIMACS Implementation Challenge*, volume 2, pages 141–175. American Mathematical Society, 2009.

[42] D. Delling and D. Wagner. Landmark-based routing in dynamic graphs. *Experimental Algorithms*, 2:52–65, 2007.

[43] D. Delling and D. Wagner. Pareto Paths with SHARC. *Proceedings of the 8th International Symposium on Experimental Algorithms (SEA'09)*, 5526:125–136, 2009.

[44] D. Delling and D. Wagner. Time-Dependent Route Planning. In R. K. Ahuja, R. H. Mohring, and C. D. Zaroliagis, editors, *Robust and Online Large-Scale Optimization: Models and Techniques for Transportation Systems*, volume 5868 of *Lecture Notes in Computer Science*, pages 207–230. Springer, Berlin, 2009.

[45] C. Demetrescu, A. V. Goldberg, and D. S. Johnson, editors. *The Shortest Path Problem: Ninth Dimacs Implementation Challenge*, volume 74 of *DIMACS: Discrete Mathematics and Theoretical Computer Science Series.* American Mathematical Society, 2009.

[46] J. Desrosiers, Y. Dumas, and F. Soumis. A dynamic programming solution of the large-scale single-vehicle dial-a-ride problem with time windows. *American Journal of Mathematical and Management Sciences*, 6:301–325, 1986.

[47] M. Diana and M. M. Dessouky. A new regret insertion heuristic for solving large-scale dial-a-ride problems with time windows. *Transportation Research Part B: Methodological*, 38(6):539–557, 2004.

[48] M. Diana, M. M. Dessouky, and N. Xia. A model for the fleet sizing of demand responsive transportation services with time windows. *Transportation Research Part B: Methodological*, 40(8):651–666, 2006.

[49] J. Dibbelt, T. Pajor, B. Strasser, and D. Wagner. Intriguingly Simple and Fast Transit Routing. In *Proceedings of the 12th International Symposium on Experimental Algorithms (SEA'13)*, Lecture Nodes of Computer Science. Springer, Berlin, 2013.

[50] J. Dibbelt, T. Pajor, and D. Wagner. User-Constrained Multi-Modal Route Planning. In D. A. Bader and P. Mutzel, editors, *Proceedings of the 14th Meeting on Algorithm Engineering and Experiments (ALENEX'12)*, pages 118–129. SIAM, Philadelphia, 2012.

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

[52] Y. Disser, M. Müller-Hannemann, and M. Schnee. Multi-criteria shortest paths in time-dependent train networks. In C. McGeoch, editor, *Proceedings of the 7th Workshop on Experimental Algorithms (WEA'08)*, volume 5038 of *Lecture Notes in Computer Science*, pages 347–361. Springer, Berlin, 2008.

[53] S. E. Dreyfus. An Appraisal of Some Shortest-Path Algorithms. *Operations Research*, 17(3):395–412, May 1969.

[54] Y. Dumas, J. Desrosiers, and F. Soumis. The pickup and delivery problem with time windows. *European Journal Of Operational Research*, 54(1):7–22, 1991.

[55] L. R. Ford. Network Flow Theory. Paper P-92, 1956.

[56] L. R. Ford and D. R. Fulkerson. Modern Heuristic Techniques for Combinatorial Problems. 1962.

[57] M. Fredman and R. Tarjan. Fibonacci heaps and their uses in improved network optimization algorithms. *Journal of the ACM*, 34(3):596–615, 1987.

[58] R. Geisberger. *Contraction hierarchies: Faster and simpler hierarchical routing in road networks.* Master thesis, Universitat Karlsruhe (TH), 2008.

[59] R. Geisberger. Contraction of timetable networks with realistic transfers. In P. Festa, editor, *Proceedings of the 9th International Symposium on Experimental Algorithms (SEA'10)*, volume 6049 of *Lecture Notes in Computer Science*, pages 71–82. Springer, Berlin, 2010.

[60] R. Geisberger, M. Kobitzsch, and P. Sanders. Route planning with flexible objective functions. In G. E. Blelloch and D. Halperin, editors, *Proceedings of the 12th Workshop on Algorithm Engineering and Experiments (ALENEX'10)*, ALENEX, pages 124–137. SIAM, Philadelphia, 2010.

[61] R. Geisberger, P. Sanders, D. Schultes, and D. Contraction hierarchies: Faster and simpler hierarchical routing in road networks. In C. C. McGeoch, editor, *Proceedings of the 7th Workshop on Experimental Algorithms (WEA'08)*, volume 5038 of *Lecture Notes in Computer Science*, pages 319–333. Springer, Berlin, 2008.

[62] R. Geisberger, P. Sanders, D. Schultes, and C. Vetter. Exact Routing in Large Road Networks Using Contraction Hierarchies. In *Transportation Science*, volume 46, pages 388–404, 2012.

[63] M. Gendreau. An Introduction to Tabu Search. *Handbook of Metaheuristics*, 57:37–54, 2003.

[64] F. Glover. Future paths for integer programming and links to artificial intelligence. *Computers & Operations Research*, 13(5):533–549, 1986.

[65] A. V. Goldberg and C. Harrelson. Computing the shortest path: A* search meets graph theory. In *Proceedings of the Symposium on Discrete Algorithms (SODA'05)*, pages 156–165. SIAM, Philadelphia, 2005.

[66] A. V. Goldberg and R. F. Werneck. Computing point-to-point shortest paths from external memory. In C. Demetrescu, R. Sedgewick, and R. Tamassia, editors, *Proceedings of the 7th Workshop on Algorithm Engineering and Experiments and the 2th Workshop on Analytic Algorithmics and Combinatorics (ALENEX /ANALCO '05)*, ALENEX/ANALCO, pages 26–40. SIAM, Philadelphia, 2005.

[67] L. Häme and H. Hakula. Dynamic journeying under uncertainty. *European Journal of Operational Research*, 225(3):455–471, Mar. 2013.

[68] P. E. Hart, N. J. 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.

[69] M. Holzer. *Engineering Planar-Separator and Shortest-Path Algorithms.* Phd thesis, University of Karlsruhe, 2008.

[70] M. Holzer, F. Schulz, and D. Wagner. Engineering multi-level overlay graphs for shortest-path queries. *ACM Journal of Experimental Algorithmics*, 13(2.5):1–26, 2008.

[71] J. E. Hopcroft, R. Motwani, and J. D. Ullman. *Introduction to Automata Theory, Languages, and Computation.* Pearson/Addison Wesley, third edition, 2007.

[72] Y.-W. Huang, N. Jing, and E. a. Rundensteiner. Effective graph clustering for path queries in digital map databases. *Proceedings of the fifth International Conference on Information and knowledge management (CIKM'96)*, pages 215–222, 1996.

[73] M.-J. Huguet, D. Kirchler, P. Parent, and R. Wolfler Calvo. Efficient algorithms for the 2-Way Multi Modal Shortest Path Problem. *Electronic Notes in Discrete Mathematics*, 41:431–437, June 2013.

[74] T. Ikeda, M.-Y. Hsu, H. Imai, S. Nishimura, H. Shimoura, T. Hashimoto, K. Tenmoku, and K. Mitoh. A fast algorithm for finding better routes by AI search techniques. In *Proceedings of Vehicle Navigation and Information Systems Conference*, pages 291–296. IEEE, 1994.

[75] I. Ioachim, J. Desrosiers, Y. Dumas, M. M. Solomon, and D. Villeneuve. A Request Clustering Algorithm for Door-to-Door Handicapped Transportation. *Transportation Science*, 29(1):63–78, 1995.

[76] R. Jacob, M. V. Marathe, and K. Nagel. A computational study of routing algorithms for realistic transportation networks. *ACM Journal of Experimental Algorithmics*, 4, 1999.

[77] J.-J. Jaw, A. Odoni, H. N. Psaraftis, and N. H. Wilson. A heuristic algorithm for the multi-vehicle advance request dial-a-ride problem with time windows. *Transportation Research Part B: Methodological*, 2(3):243–257, 1986.

[78] R. M. Jorgensen, J. Larsen, and K. B. Bergvinsdottir. Solving the Dial-a-Ride problem using genetic algorithms. *Journal of the Operational Research Society*, 58(10):1321–1331, 2006.

[79] S. Jung and S. Pramanik. An efficient path computation model for hierarchically structured topographical road maps. *IEEE Transactions on Knowledge and Data Engineering*, 14(5):1029 – 1046, 2002.

[80] E. Kaufman and R. L. Smith. Fastest paths in time-dependent networks for intelligent vehicle-highway systems applications. *Journal of Intelligent Transportation Systems*, 1(1):1–11, 1993.

[81] M. B. Kennel. KDTREE 2: Fortran 95 and C++ software to efficiently search for near neighbors in a multi-dimensional Euclidean space. Technical report, 2004.

[82] B. S. Kerner. *Introduction to Modern Traffic Flow Theory and Control.* Springer, Berlin, 2009.

[83] S. C. Kleene. Representation of Events in Nerve Nets and Finite Automata. In C. E. Shannon and J. McCarthy, editors, *Automata Studies*, pages 3 – 42. Princeton University Press, 1956.

[84] N. Labadie, R. Mansini, J. Melechovský, and R. Wolfler Calvo. The Team Orienteering Problem with Time Windows: An LP-based Granular Variable Neighborhood Search. *European Journal of Operational Research*, 220(1):15–27, Jan. 2012.

[85] U. Lauther. An Extremely Fast, Exact Algorithm for Finding Shortest Paths in Static Networks with Geographical Background. *Geoinformation und Mobilität - von der Forschung zur praktischen Anwendung*, 22:219–230, 2004.

[86] A. Lozano and G. Storchi. Shortest viable path algorithm in multimodal networks. *Transportation Research Part A*, 35:225–241, Mar. 2001.

[87] Q. Lu and M. M. Dessouky. A new insertion-based construction heuristic for solving the pickup and delivery problem with time windows. *European Journal Of Operational Research*, 175(2):672–687, 2006.

[88] O. B. G. Madsen, H. F. Ravn, and J. M. Rygaard. A heuristic algorithm for a dial-a-ride problem with time windows, multiple capacities, and multiple objectives. *Annals of Operations Research*, 60(1):193–208, 1995.

[89] J. Maue, P. Sanders, and D. Matijevic. Goal-directed shortest-path queries using precomputed cluster distances. *ACM Journal of Experimental Algorithmics*, 14, 2009.

[90] A. O. Mendelzon and P. T. Wood. Finding Regular Simple Paths in Graph Databases. *SIAM Journal on Computing*, 24(6):1235–1258, 1995.

[91] C. E. Miller, A. W. Tucker, and R. A. Zemlin. Integer programming formulations and traveling salesman problems. *Journal of the ACM*, 7(4):326–329, 1960.

[92] A. Moore. An intoductory tutorial on kd-trees. *Technical Report*, (209), 1991.

[93] E. Moore. The Shortest Path Through a Maze. In *Proceedings of an International Symposium on the Theory of Switching*, pages 285–292. Harvard University Press, 1959.

[94] H. Moritz. Geodetic Reference System 1980. *Journal of Geodesy*, 66(2):187–192, 1992.

[95] M. Müller-Hannemann and M. Schnee. Finding all attractive train connections by multi-criteria pareto search. In F. Geraets, L. G. Kroon, A. Schöbel, D. Wagner, and C. D. Zaroliagis, editors, *International Dagstuhl Workshop on Algorithmic Methods for Railway Optimization*, volume 4359 of *Lecture Notes in Computer Science*, pages 246–263. Springer, Berlin, 2007.

[96] G. Nannicini. *Point-to-Point Shortest Paths on Dynamic Time-Dependent Road Networks.* Phd thesis, Ecole Polytechnique, 2009.

[97] G. Nannicini, D. Delling, L. Liberti, and D. Schultes. Bidirectional A* search for time-dependent fast paths. In C. C. McGeoch, editor, *Proceedings of the 7th Conference on Experimental Algorithms (WEA'08)*, volume 5038 of *Lecture Notes in Computer Science*, pages 334–346. Springer, Berlin, 2008.

[98] G. Nannicini, D. Delling, and D. Schultes. Bidirectional A* search on time-dependent road networks. *Networks*, 59(2):240–251, 2012.

[99] G. Nannicini and L. Liberti. Shortest paths in dynamic graphs. *International Transactions in Operational Research*, 15:551–563, 2008.

[100] 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, 1990.

[101] J. Paquette, J.-F. Cordeau, G. Laporte, and M. M. B. Pascoal. Combining Multicriteria Analysis and Tabu Search for Dial-a-Ride Problems. *Technical Report CIRRELT*, (4), 2012.

[102] S. N. Parragh, K. F. Doerner, and R. F. Hartl. A survey on pickup and delivery problems Part II : Transportation between pickup and delivery locations. *Journal für Betriebswirtschaft*, (58):81–117, 2008.

[103] S. N. Parragh, K. F. Doerner, and R. F. Hartl. Variable neighborhood search for the dial-a-ride problem. *Computers & Operations Research*, 37(6):1129–1138, 2010.

[104] S. N. Parragh and V. Schmid. Hybrid column generation and large neighborhood search for the dial-a-ride problem. *Computers & Operations Research*, 40(1):490–497, 2012.

[105] I. Pohl. Bi-directional Search. In B. Meltzer and D. Michie, editors, *Machine Intelligence 6*, volume 6, pages 127–140. Edinburgh University Press, Edinburgh, 1971.

[106] H. N. Psaraftis. An exact algorithm for the single vehicle many-to-many dial-a-ride problem with time windows. *Transportation Science*, 17(3):351–357, 1983.

[107] E. Pyrga, F. Schulz, D. Wagner, and C. Zaroliagis. Efficient models for timetable information in public transportation systems. *Journal of Experimental Algorithmics (JEA)*, 12:Article 2.4, June 2007.

[108] M. Rabin and D. Scott. Finite automata and their decision problems. *IBM Journal of Research and Development*, 3(April):114–125, 1959.

[109] M. Rice and V. J. Tsotras. Graph indexing of road networks for shortest path queries with label restrictions. *Proceedings of the VLDB endowment*, 4(2):69–80, 2010.

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

[111] P. Sanders and D. Schultes. Highway hierarchies hasten exact shortest path queries. In G. Stolting Brodal and S. Leonardi, editors, *Proceedings of the 13th Annual European Symposium on Algorithms (ESA'05)*, volume 3669 of *Lecture Notes in Computer Science*, pages 568–579. Springer, Berlin, 2005.

[112] P. Sanders and D. Schultes. Engineering highway hierarchies. In Y. Azar and T. Erlebach, editors, *Proceedings of the 14th Annual European Symposium on Algorithms (ESA'06)*, volume 4168 of *Lecture Notes in Computer Science*, pages 804–816. Springer, Berlin, 2006.

[113] P. Sanders and D. Schultes. Robust, almost constant time shortest-path queries in road networks. In C. Demetrescu, A. V. Goldberg, and J. E. Hopcroft, editors, *9th DIMACS Implementation Challenge - Shortest Paths*, volume 74, pages 193–218. American Mathematical Society, 2006.

[114] G. Sauvanet. *Recherche de chemins multiobjectifs pour la conception et la realisation d'une centrale de mobilité destinée aux cyclistes.* Phd thesis, University of Tours (France), 2011.

[115] M. W. P. Savelsbergh. Local search in routing problems with time windows. *Annals of Operations Research*, 4(1):285–305, 1985.

[116] D. Schultes. *Route planning in road networks.* Phd thesis, TH Karlsruhe, 2008.

[117] F. Schulz. *Timetable Information and Shortest Paths.* Phd thesis, University of Karlsruhe (TH), 2005.

[118] F. Schulz, D. Wagner, and C. Zaroliagis. Using multi-level graphs for timetable information in railway systems. In D. M. Mount and C. Stein, editors, *In Proceedings of the 4th International Workshop on Algorithm Engineering and Experiments (ALENEX'02)*, volume 2409 of *Lecture Notes in Computer Science*, pages 43–59. Springer, Berlin, 2002.

[119] R. Sedgewick and J. S. Vitter. Shortest Paths in Euclidean Graphs. *Algorithmica*, 1(1):31–48, 1986.

[120] T. R. Sexton and Y. M. Choi. Pickup and delivery of partial loads with ”soft” time windows. *American Journal of Mathematical and Management Sciences*, 6(3-4):369–398, 1986.

[121] H. D. Sherali, A. G. Hobeika, and S. Kangwalklai. Time-Dependent, Label-Constrained Shortest Path Problems with Applications. *Transportation Science*, 37(3):278–293, 2003.

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

[123] M. Sipser. *Introduction to the Theory of Computation.* Course Technology Ptr, third edition, 2012.

[124] E. Taillard. Parallel iterative search methods for vehicle routing problems. *Networks*, 23(8):661–673, 1993.

[125] P. Toth and D. Vigo. Heuristic algorithms for the handicapped persons transportation problem. *Transportation Science*, 31(1):60–71, 1997.

[126] P. Toth and D. Vigo. The Granular Tabu Search and Its Application to the Vehicle-Routing Problem. *INFORMS Journal on Computing*, 15(4):333–346, 2003.

[127] M. Treiber, A. Kesting, and C. Thiemann. *Traffic Flow Dynamics.* Springer, Berlin, 2013.

[128] D. Wagner and T. Willhalm. Geometric speed-up techniques for finding shortest paths in large sparse graphs. In G. Di Battista and U. Zwick, editors, *Proceedings of the 11th Annual European Symbosium on Algorithms (ESA'03)*, volume 2832 of *Lecture Notes in Computer Science*, pages 776–787. Springer, Berlin, 2003.

[129] R. Wolfler Calvo and A. Colorni. An effective and fast heuristic for the Dial-a-Ride problem. *4or*, 5(1):61–73, 2006.

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

Appendix

Appendix A

SDALT: Examples

A.1 Details for `IVa`

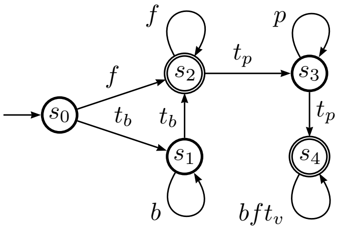


Figure A.1: Scenario `IVa`

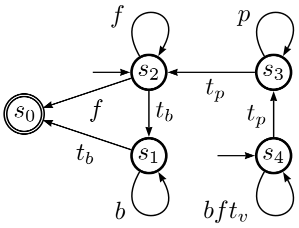


Figure A.2: Scenario `IVa`: backward automaton

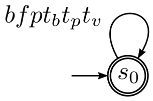
s_x	L_{bas}
s_0, s_1, s_2, s_3, s_4	

Table A.1: Scenario `IVa`: automaton for landmark distance calculation of `bas_ls` and `bas_bi`$_{vx}$.

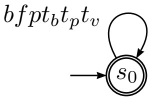
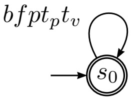
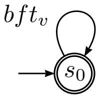
s_x	L_{adv,s_x}
s_0, s_1	
s_2, s_3	
s_4	

Table A.2: Scenario `IVa`: automata for landmark distance calculation of `adv_ls` and `adv_lc`

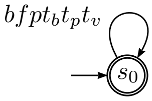
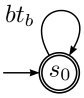
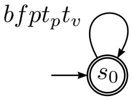
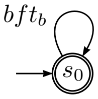
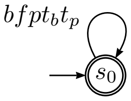
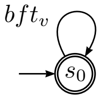
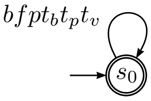
	forward search	backward search
s_x	L_{adv,s_x}	L_{adv,s_x}
s_0, s_1		
s_2		
s_3		
s_4		

Table A.3: Scenario `IVa`: automata for landmark distance calculation of $\texttt{adv\_bi}_{vx}$

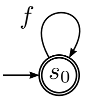
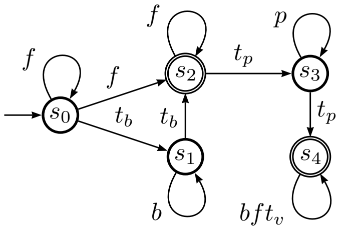
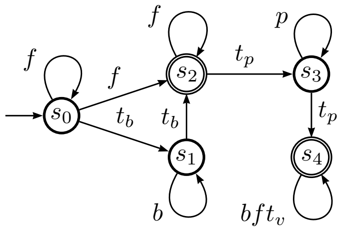
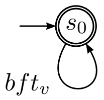
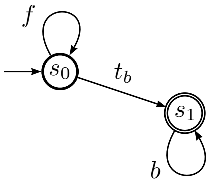
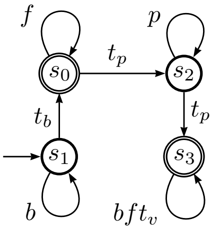
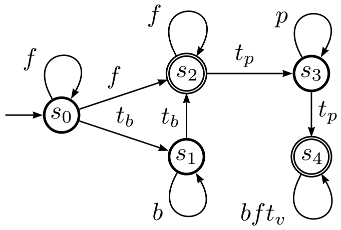
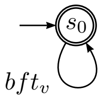
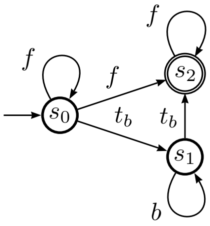
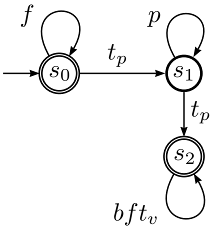
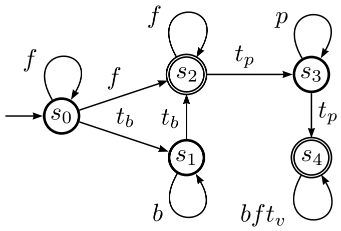
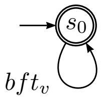
s_x	$L^{\ell\to v}_{s_x}$ $L^{\ell\to t}_{s_x}$	$L^{v\to\ell}_{s_x}$ $L^{t\to\ell}_{s_x}$
s_0		
		
s_1		
		
s_2		
		

Table A.4: Scenario `IVa`: automata for landmark distance calculation of `spe_ls` and forward search of `spe_bi`$_{vx}$, part 1

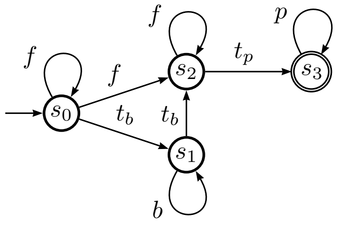
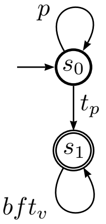
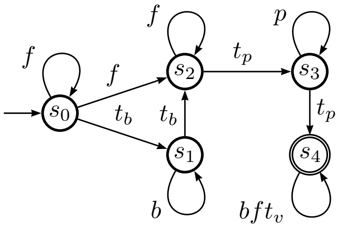
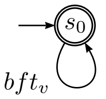
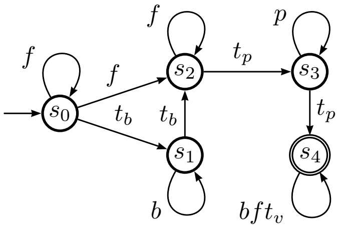
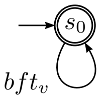
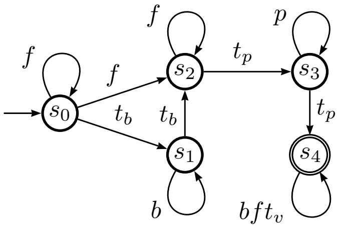
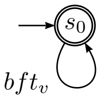
	$L_{s_x}^{\ell \to v}$	$L_{s_x}^{v \to \ell}$
s_x	$L_{s_x}^{\ell \to t}$	$L_{s_x}^{t \to \ell}$
s_3	 f s_0 f f s_2 t_p p s_3 t_b t_b s_1 b	
	 f s_0 f f s_2 t_p p s_3 t_b t_b s_1 t_p s_4 b bft_v	
s_4	 f s_0 f f s_2 t_p p s_3 t_b t_b s_1 t_p s_4 b bft_v	
	 f s_0 f f s_2 t_p p s_3 t_b t_b s_1 t_p s_4 b bft_v	

Table A.5: Scenario `IVa`: automata for landmark distance calculation of `spe_ls` and forward search of `spe_bi`$_{vx}$, part 2

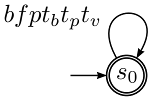
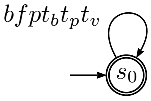
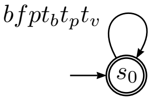
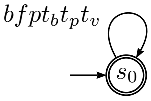
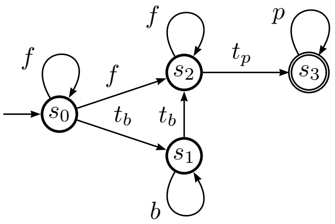
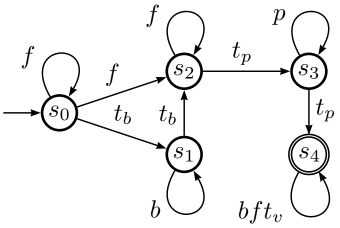
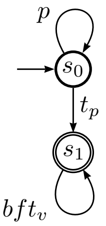
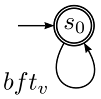
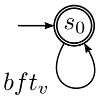
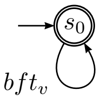
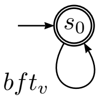
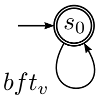
s_x	$L^{\ell\to v}_{s_x}$ $L^{\ell\to t}_{s_x}$	$L^{v\to\ell}_{s_x}$ $L^{t\to\ell}_{s_x}$
s_0, s_1, s_2	 	 
s_3	 f f f p t_p s_0 s_2 s_3 t_b t_b s_1 b  f f f p t_p s_0 s_2 s_3 t_b t_b t_p s_1 s_4 b bft_v	 
s_4	 	 

Table A.6: Scenario `IVa`: automata for landmark distance calculation of `spe_lc`

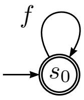
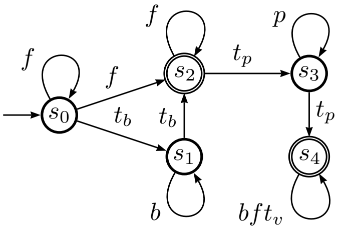
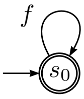
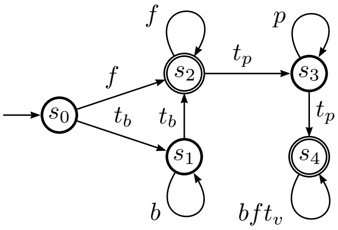
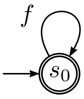
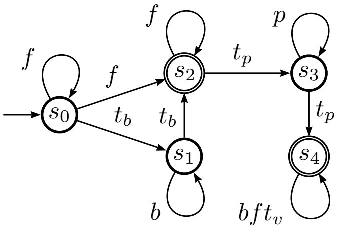
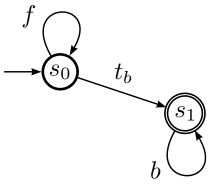
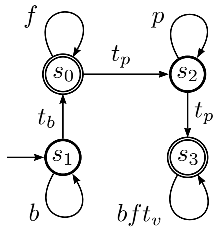
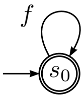
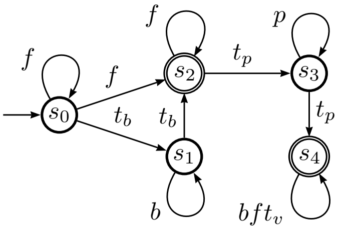
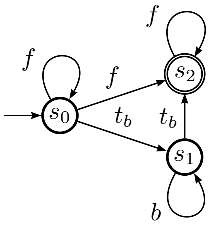
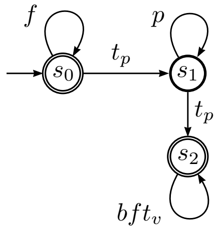
s_x	$L^{\ell\to r}_{s_x}$ $L^{\ell\to v}_{s_x}$	$L^{r\to\ell}_{s_x}$ $L^{v\to\ell}_{s_x}$
s_0		 f p f tp f s2 s3 s0 tb tb tp s1 s4 b bftv
		 f p tp f s2 s3 s0 tb tb tp s1 s4 b bftv
s_1		 f p f tp f s2 s3 s0 tb tb tp s1 s4 b bftv
		 f p tp s0 s2 tb tp s1 s3 b bftv
s_2		 f p f tp f s2 s3 s0 tb tb tp s1 s4 b bftv
		

Table A.7: Automata used for backward search of $\texttt{spe\_bi}_{vx}$, part 1

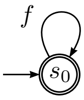
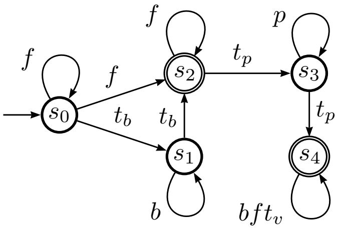
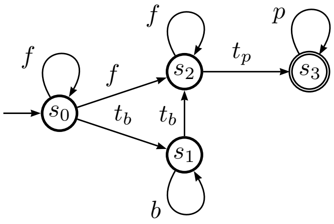
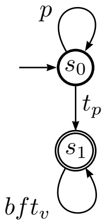
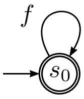
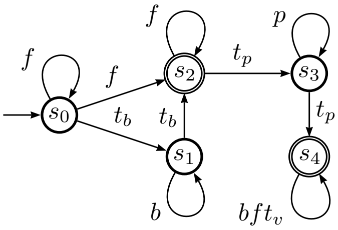
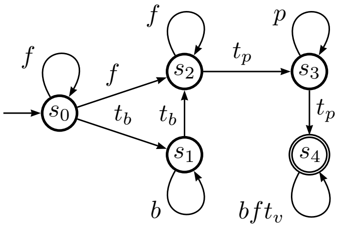
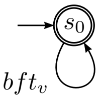
s_x	$L^{\ell\to r}_{s_x}$ $L^{\ell\to v}_{s_x}$	$L^{r\to\ell}_{s_x}$ $L^{v\to\ell}_{s_x}$
s_3		 f p f f t_p s_2 s_3 s_0 t_b t_b t_p s_1 s_4 b bft_v
	 f p f f t_p s_2 s_3 s_0 t_b t_b s_1 b	
s_4		 f p f f t_p s_2 s_3 s_0 t_b t_b t_p s_1 s_4 b bft_v
	 f p f f t_p s_2 s_3 s_0 t_b t_b t_p s_1 s_4 b bft_v	

Table A.8: Automata used for backward search of `spe_bi`$_{vx}$, part 2