Czech Technical University in Prague
Faculty of Electrical Engineering

Department of Computer Science and Engineering

DIPLOMA THESIS ASSIGNMENT

Student: **Bc. Pavol Žilecký**

Study programme: Open Informatics
Specialisation: Artificial Intelligence

Title of Diploma Thesis: **Junction-aware Multicriteria Bicycle Route Planning**

Guidelines:

1. Survey existing methods and speedups for multicriteria route planning and bicycle route planning.
2. Formalise the multicriteria bicycle trip planning problem and a representation of the cycleway network with advanced representation of junctions.
3. Propose a multicriteria algorithm and speedups to solve the bicycle trip planning problem.
4. Implement the proposed algorithm capable of working with real-world network of roads and cycleways.
5. Evaluate the implemented algorithm on a set of test scenarios based on real-world cycleway network and travel demand data.

Bibliography/Sources:

[1] E. Q. V. Martins. On a multicriteria shortest path problem. European Journal of Operational Research, 16(2):236-245, 1984.
[2] Jan Hrncir and Qing Song and Pavol Zilecky and Marcel Nemet and Michal Jakob: Bicycle Route Planning with Route Choice Preferences. In Prestigious Applications of Artificial Intelligence (PAIS). 2014.
[3] J. Broach, J. Dill, and J. Gliebe,'Where do cyclists ride? A route choice model developed with revealed preference GPS data', Transportation Research Part A: Policy and Practice, 46(10), 1730-1740, (2012).

Diploma Thesis Supervisor: Mgr. Jan Hrnčíř

Valid until the end of the summer semester of academic year 2015/2016

[redacted]

doc. Ing. Filip Železný, Ph.D.
Head of Department

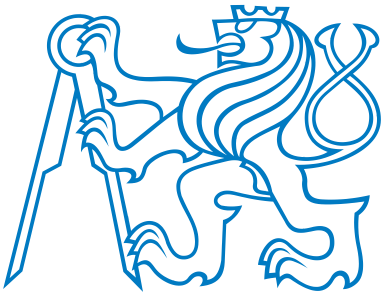
L.S.

[redacted]

prof. Ing. Pavel Ripka, CSc.
Dean

Prague, March 26, 2015

Czech Technical University in Prague
Faculty of Electrical Engineering
Department of Computer Science and Engineering



Master's Thesis

Junction-aware Multicriteria Bicycle Route Planning

Pavol Žilecký

Supervisor: Jan Hrnčíř

Study Programme: Open Informatics

Field of Study: Artificial Intelligence

May 11, 2015

Aknowledgements

At first, I would like to thank my supervisors Jan Hrnčíř and Michal Jakob for their guidance and support. Also, I wish to thank Jan Nykl and Sebastián García for their feedback. Last but not least, a huge "Thank you!" goes to my parents and Zuzka Baronová for their support and encouragement.

Also, I greatly appreciate the access to computing and storage facilities owned by parties and projects contributing to the National Grid Infrastructure MetaCentrum, provided under the programme "Projects of Large Infrastructure for Research, Development, and Innovations" (LM2010005).

Declaration

I hereby declare that I have completed this thesis independently and that I have listed all the literature and publications used.
I have no objection to usage of this work in compliance with the act §60 Zákon č. 121/2000Sb. (copyright law), and with the rights connected with the copyright act including the changes in the act.

In Prague on May 11, 2015 .....

Abstract

Navigating the cyclists in larger cities is often challenging due to cities' fragmented cycling infrastructure and/or complex terrain topology. Cyclists would thus benefit from intelligent route planning that would help them discover routes that best suit their transport needs and preferences. The studies of cyclists' route choice reveal that the cyclists are sensitive to difficult manoeuvres at junctions. Therefore, to address this issue, we proposed a method to extend a graph structure with an advanced junction representation. Also, to account other cyclists' route-choice criteria, we employ multi-criteria route search. Direct application of optimal multi-criteria route search algorithms is, however, not feasible due to their prohibitive computational complexity. In this thesis, we formalise a multi-criteria bicycle route planning problem and propose several solution methods together with pruning heuristics for speeding up the multi-criteria route search. We evaluate our methods on a real-world cycleway network of the city of Prague and discuss the difference between the bicycle routes planned on the extended graph and the standard graph. The evaluation shows that speedups of up to three orders of magnitude over the multi-criteria Dijkstra's algorithm are possible with a reasonable loss of solution quality.

Abstrakt

Navigace pro cyklisty ve větších městech je často náročná vzhledem k roztříštěné cyklistické infrastruktuře nebo složité topologii terénu. Inteligentní plánovač cyklistických tras pomůže cyklistům objevit trasy, které nejlépe vyhovují jejich potřebám a preferencím. Studie volby tras cyklistů ukazují, že cyklisté jsou citliví na obtížné manévry na křižovatkách. Pro řešení tohoto problému jsme navrhli grafovou strukturu s rozšířenou reprezentací křižovatek. Pro zohlednění více kritérií při volbě trasy, jsme použili vícekriteriální algoritmy na hledání nejkratší cesty v grafu. Přímá aplikace optimálních vícekriteriálních vyhledávacích algoritmů je vzhledem k jejich velké výpočetní složitosti nemožná. V této práci jsme proto formalizovali vícekriteriální plánovací problém pro hledání cyklotras a navrhli několik způsobů řešení spolu s heuristikami pro urychlení vícekriteriálního vyhledávání. Navrhované metody jsme otestovali na cyklistické síti města Prahy a porovnali jsme rozdíly mezi trasami plánovanými na standardním grafu a na grafu s rozšířenou reprezentací křižovatek. Vyhodnocení ukazuje, že je možné urychlení o tři řády vzhledem k vícekriteriálnímu Dijkstrovu algoritmu s rozumnou ztrátou kvality řešení.

Contents

"When I see an adult on a bicycle,
I do not despair for the future of human race."
H.G. Wells, English author

Chapter 1

Introduction

1.1 Motivation

People have been using a bicycle since 19th century [1]. Since then, it has become an important part of life of millions of people around the world. They use the bicycle for commuting purposes like travel from home to work, to school, and also for non-commute trips, e.g., shopping, errands. At last, cycling has also turned into a way of exercise and recreation.

In today's society where the number of civilization diseases is enormously increasing every year and worldwide obesity has doubled since 1980 according to World Health Organization [2], cycling presents a reasonable and easy solution. It is a workout which has a great impact on the cardiovascular system. In developed countries, the bicycle is favoured as a "green" machine that can help reduce cyclist's weight and cities' traffic congestion. It is an affordable mode of transport that can significantly reduce the cost of travel. Moreover, cycling does not require oil-based fuels. As such, harmful emissions are reduced with the presence of fewer motor vehicles. Cycling also reduces a noise pollution in urban areas [3]. Besides, bicycle commuters often find they save time because they can breeze past the traffic and do not need to hunt for a parking space for their vehicle. In fact, commuters also save a small fortune in gasoline, car's maintenance and repairs. To summarize, bicycling is a way to sharply lower emissions and increase the quality of life.

To promote cycling in cities, we decided to develop a *cycle planner* application. A user will use the cycle planner to decide which path to take in order to successfully travel from an origin to a destination. Such a planner would be particularly useful for inexperienced cyclists with a limited knowledge of their surroundings. It could help them to discover routes that best suite their preferences. Furthermore, experienced cyclists would benefit from an automated bicycle route planning software to fine-tune their everyday paths [4]. In contrast to car drivers, cyclists consider a significantly broader range of factors when deciding about their routes. To properly cover cyclists' needs and preferences, and to model cycleway transport network features, multiple planning criteria and rich world representation are required. Therefore, the bicycle route planning is a challenging Artificial Intelligence problem.

1.2 Aim of the Thesis

The aim of the thesis is to build a cycle planner application capable of planning bicycle routes while considering multiple criteria which affect the bicycle ride and taking into account an complexity of junctions.

To successfully provide such a outcome, we did the following steps. First, we analyse existing methods for solving the multi-criteria shortest path problem and describe the solution methods widely used in a transportation domain. Particularly we focus on a bicycle route planning problem.

Second, we model a network of ways allowed for cyclists (i.e., a cycleway network) as a graph structure and then extend it with a advance representation of junctions. The advance representation is needed to cover factors that have a significant impact on a ride through the junction. After a proper definition of the cycleway network, a mathematical formalisation of the multi-criteria bicycle route planning problem is introduced.

We aim to use *multi-criteria algorithms* which take all criteria into account equally. A group of these algorithms return multiple alternative bicycle routes between an origin and a destination location. Each one of the alternative paths provides different properties based on specified criteria. Providing multiple alternatives is an advantage, because the relative importance of criteria vary widely among cyclists. On the other hand, as described in following Section 2.3, running time of these algorithms is much worse than algorithms considering only one criterion. Also, a number of alternative routes may grow exponentially with respect to a number of criteria and a distance between the origin and the destination.

We require to compute the solution routes in a reasonable time for real-time services. To fulfil this condition, we implement the multi-criteria algorithm and introduce several pruning techniques.

Eventually, we evaluate the techniques with respect to the running time and a route quality. In the end, we take the best implemented method and shows all alternative routes in a web application for the city of Prague.

This work builds on the Marcel Němet's bachelor's project [5] developed in 2013. We extend his work in the following directions. First, the cycleway graph structure is extended by the advanced representation of junctions. Second, Němet used for solving the multi-criteria shortest path problem a single criteria algorithm by taking into account weighted sum of the criteria values. We use the *proper* multi-criteria algorithms which provides multiple alternative bicycle routes. Third, we focus to enhance a plan quality by improving the data processing and the graph data structure construction steps.

1.3 Structure of the Thesis

To begin with, the background of the multi-criteria shortest path problem together with the related work is given in Chapter 2. Then we proceed to the specification of the multi-criteria bicycle route planning problem as the graph search problem in Chapter 3. Chapter 4 proposes a multi-criteria algorithm and speedups to solve the bicycle route planning problem and is followed by the implementation of the algorithm and the map data processing in

Chapter 5. The performance of the implemented solution methods is evaluated in Chapter 6. Finally, Chapter 7 presents the conclusion deduced from the obtained results.

Chapter 2

Related Work

This thesis deals with a multi-criteria bicycle route planning problem in an urban environment. In general, the route planning problem is an application of the shortest path problem in a road network. Graph theory defines the shortest path problem as the problem of finding the shortest path between two nodes in a graph. The graph consists of nodes and edges that connect them. Furthermore, each edge is associated with a cost, which is usually a real number depending on the problem and the algorithm used to solve the problem. Thus, the path, i.e., the sequence of edges, is the shortest one if it minimizes the sum of costs of its edges. In the route planning problem, the graph is the road network, the nodes are intersections, the edges are individual roads between these intersections, and the distance between them could be the cost of each edge.

The cost does not have to be only the distance between intersections. For example in public transport domain, it could be the number of transfers. We refer to the cost as the value of a criterion for what we search the shortest path. Notably, in a car route planning problem, the criteria could be a distance, a travel time or a fuel consumption. In the bicycle route planning problem, we are going to optimise a much wider set of criteria, because cyclists consider a broader range of factors when deciding about their routes.

Therefore, next section will introduce criteria considered by cyclists while selecting routes. After that, the shortest path problem considering only the single criterion is described, followed by the description of the multi-criteria problem which searches for the shortest path optimising multiple criteria at the same time. Lastly, the methods of turns penalisation on junctions in the route planning domain are presented.

2.1 Route Choice Criteria for Cyclists

To find out what criteria are crucial for cyclists in an urban environment, we get inspired by studies [6, 7]. These employed questionnaires and GPS tracking and have found that cyclists are sensitive to trip distance, turn frequency, slope, junction control, surface quality, noise, pollution, scenery, and traffic volumes.

Results from [6] also reveal interesting facts about cyclists' attitude to intersections. On a 1 mile commute trip, a cyclist would be willing to travel about 5.9% and 9.1% farther to avoid a crossing and a left turn, respectively, at an unsignalized intersection with a cross

traffic averaged 10,000–20,000 vehicles per day. This percentage is even larger for non-commute trips. Such an outcome is essential in a bicycle route planning problem.

2.2 Single Criterion Shortest Path Problem

The single criterion shortest path problem finds the path between two nodes in the graph minimising only one cost value at a time. The cost determines a value for the criterion. For example, to solve the single criterion shortest path problem for cars, only the travel time but not the fuel consumption nor the distance can be considered.

The standard solution to the single criterion shortest path problem in route planning domain is Dijkstra's algorithm [8]. In each iteration, the algorithm sets a label to a node with its current cost from the origin node. The algorithm maintains a priority queue of labels ordered by the cost. One node may have multiple labels assigned to it, and each label represents different path to the node, but only the best label is expanded from the priority queue while the others are discarded. Once a label is expanded from the priority queue, the algorithm has found the shortest path from the origin to the node to which the label belongs.

In last decades there have been many studies improving the Dijkstra's algorithm, particularly in the route planning domain [9, 10]. These improvements mostly aim to decrease the size of the algorithm's search space. For example, a *bidirectional search* simultaneously runs forward search from the origin and backward search from the destination. The A* algorithm [11] is a representative of a goal directed search technique that guides the search toward the destination. For each node, the A* algorithm employs a *heuristic* function providing a lower bound of the cost of a path from the node to the destination. This causes that the nodes closer to the destination are going to be expanded first. In some literature, the Dijkstra's algorithm and its variations are also referred as *label-setting* algorithms [12].

Contraction Hierarchies [13] is an advanced algorithm and a proof of the progress in route planning domain. The idea behind the algorithm is to skip over "unimportant" nodes. It employs the hierarchical aspect of the road network by augmenting the graph with *shortcuts* while preserving the shortest path relation between nodes. The speed-up against the standard goal directed solutions for the route planning queries is over a magnitude.

An alternative method for computing the shortest path is the Bellman-Ford [14, 15] algorithm. Each round in the Bellman-Ford algorithm browses all edges and looks for an improvement in nodes' path from the origin. It belongs to a group of *label-correcting* algorithms, because at each round it checks and eventually corrects those labels whose cost have been improved. Therefore, it may scan one node multiple times. In contrast to the Dijkstra's algorithm, the Bellman-Ford algorithm works also with negative costs.

2.3 Multi-Criteria Shortest Path Problem

All mentioned methods so far only consider the single cost value, i.e., algorithms are able to find the shortest path with respect to only one criterion. In the bicycle routing problem, we instead aim to consider multiple criteria. A cyclist may consider at the same time a trip

distance, a traffic volume and a quality of surface, e.g., she is willing to travel a little further in order to avoid busy roads with cobblestone surface.

To solve the multi-criteria planning problem, authors in [16] proposed a general label-setting multi-criteria extension of Dijkstra's algorithm. The algorithm is operating on labels that have a cost vector $\overrightarrow{c}$ instead of a single cost value as in the original Dijkstra's algorithm. The cost vectors contain one value for each considered criterion.

To compare the cost vectors we define a partial order relation $\prec$ called *dominance*,

$$\forall \overrightarrow{c}, \overrightarrow{c}' \quad \overrightarrow{c} \prec \overrightarrow{c}' \quad \text{iff} \quad c_i \leq c'_i \quad \forall i \; 1 \leq i \leq k \quad \wedge \quad c_j < c'_j \quad \exists j \; 1 \leq j \leq k$$

The cost vector $\overrightarrow{c}$ dominates other cost vector $\overrightarrow{c}'$, if and only if it is strictly better in at least one criteria value and not worse in any of criteria values. In particular, imagine a bi-criteria problem, a cost vector (2,3) dominates (2,4), but no dominance relation exists between cost vectors (2,3) and (3,2) [17]. Similarly, dominance holds for labels, a label dominates another label if and only if the same holds for cost vectors assigned to them. Note that, two vectors $\overrightarrow{c}, \overrightarrow{c}'$ are *non-dominated*, if and only if $\overrightarrow{c} \not\prec \overrightarrow{c}'$ and $\overrightarrow{c}' \not\prec \overrightarrow{c}$.

The multi-criteria Dijkstra's algorithm stores for each node a bag of non-dominated labels. In contrary to the original Dijkstra's algorithm, one node may have been expanded multiple times, because it is not guaranteed that there will not be any other label dominating currently expanded one. At the end, the algorithm terminates with a Pareto set [18] of solutions, i.e., all routes to the destination whose cost vectors are non-dominated to each other.

The multi-criteria search is computationally more expensive than the single criterion one. The number of label expansions can grow exponentially with the distance from the origin, even for the two objective case [19]. This makes the problem unusable in real-time applications. Therefore, the problem has attracted recent attention of researchers that aim to find the set of routes similar to the optimal one in a reasonable time. To prune the algorithm's search space, authors in [20] used an elliptical bounding box around the origin and the destination. In [21], they introduced a near admissible multi-criteria search algorithm by introducing an ϵ parameter that weakens the relation between labels. The proposed algorithm guarantees to find a solutions within a factor $(1 + \epsilon)$ from the best solutions. Note that the Pareto set of solutions in the optimal multi-criteria solution method grows exponentially but with the introduced ϵ parameter the growth become polynomial. At last, in [22], authors developed several heuristic speedups, e.g., they put criteria into buckets and instead of comparing the criteria they compare the buckets.

An alternative approach is represented by MOA* [23], NAMOA* [17], and Tung-Chew [24] algorithms that are multi-criteria extensions of the standard A* algorithm. The extension lies in using a vector of heuristics that contains, for each criterion, an estimate value to the destination. MOA* and NAMOA* algorithms define a data structure for storing unexpanded labels called an OPEN. During the expansion of a label, the algorithm first finds a subset of non-dominated labels in OPEN and then from the subset it expands a random label or the smallest label in a lexicographical order within criteria. It is proven [17] that the algorithm does not expand such a label that would be dominated by any label from the destination node's bag of labels. This is an interesting property from a perspective of the search space size.

The NAMOA* algorithm recently achieved reasonable results for a bi-criteria route planning [25] on continent size networks. For the computation of heuristic values, authors used

a modified method that was proposed in [24]. Basically, before the multi-criteria search, they compute single costs values from destination to each node and then use this value as heuristic estimate. Also, there have been proposed an improvement of NAMOA* using parallel search [26]. In [27], authors proposed to decrease a number of iterations of the NAMOA* algorithm by sorting the subset of non-dominated labels in OPEN with respect to labels' crowding distance. A non-dominated label with the lowest crowding distance is expanded, i.e., the algorithm expands a label which has a lot of other labels around; therefore, it must be in a decent region to explore. The results showed that in average, the sort helps to decrease a number of expanded nodes to half in contrary to NAMOA*.

Existing bicycle route planning approaches transform multi-criteria search to single-criterion search either by optimising each criteria separately [28, 29] or by using a weighted combination of all criteria [30, 31]. Unfortunately, the scalarisation of multi-criteria problems using a linear combination of criteria may miss many optimal routes [32, 33] and consequently reduce the quality and relevance of suggested routes. It also requires the user to weight the importance of individual route criteria a priori.

To avoid the scalarisation, authors in [34] proposed speedups for a bi-criteria Best Compromise A* (BCA*) method and effectively applied it to the bicycle route planning problem. Two of the considered criteria were distance and insecurity, and with the best speedup they achieved a ten times less number of expanded labels than with the standard BCA*. Recently, in [35], authors explored the use of optimal multi-criteria shortest path algorithms for multi-criteria bicycle routing. They proposed a route selection method within the Pareto set of solutions.

According to wiki pages of an open source multi modal trip planner called OpenTripPlanner [36], it uses the MOA* algorithm with the ϵ parameter and the Tung-Chew [24] heuristic to plan only walk-transit and bike-transit trips. Therefore, to our best knowledge, there is no openly available bicycle route planner which uses multi-criteria algorithms such as the multi-criteria Dijkstra's algorithm, MOA*, or NAMOA*.

2.4 Turns Handling

Certain manoeuvres on intersections take less time than the others, e.g., a left turn is more challenging than a right turn due to the traffic in an opposite direction. Also, in case of the bicycle route planning, turning left or handle u-turn is also much dangerous than going straight or turning right. Also, junctions may have restricted certain turns.

To declare turn penalties and disallow turns, a few researchers [37, 38] used a model called an *edge-based* graph. Researchers took the standard graph representing the road network (described above) and they modelled road segments as nodes and edges as possible connections between a pair of successive road segments. Example of such a transformation is shown in Figure 2.1. The disadvantage of the edge-based graph is that a naive implementation takes much more space than the standard graph. In [9], authors avoid this problem by introducing an interface which produces on-the-fly conversion from the standard to the edge-based representation during the bidirectional Dijkstra search.

Other approach dealing with the problem of the edge-based graph is described in [39]. Authors proposed to use a compact representation in which each intersection is a single

node associated with a *turn table*, where each row represents node's incoming edge and each column represents node's outgoing edge. Then, an element of the table contains the cost for the turn to get from the incoming edge to the outgoing one.

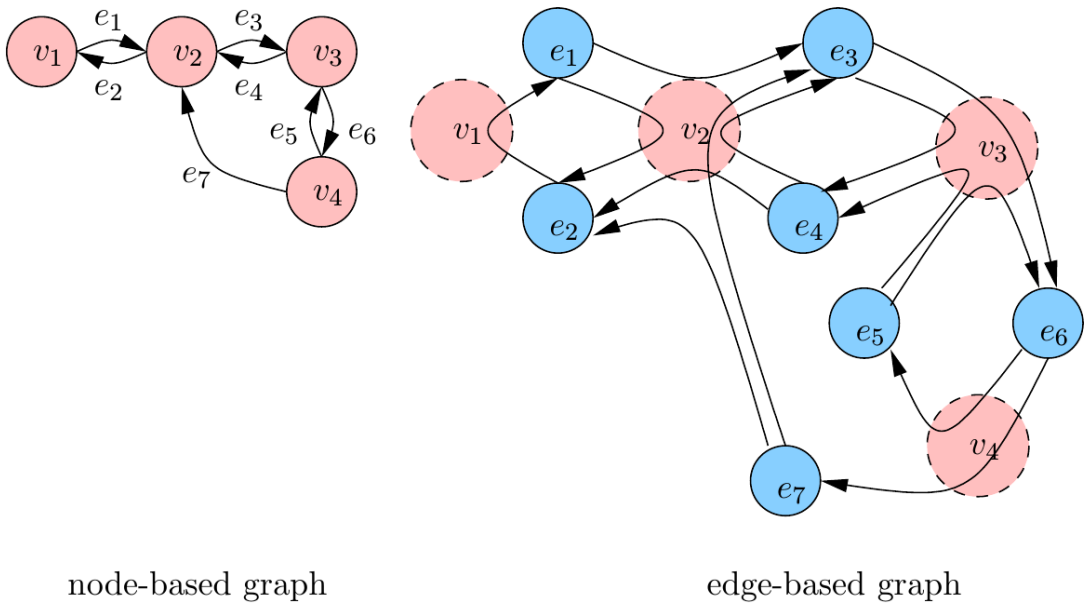


Figure 2.1: Example of node-base graph and its transformation to edge-based graph, source [37]

Chapter 3

Problem Specification

In this chapter we specify the multi-criteria bicycle route planning problem in an urban environment as the shortest path problem in the graph data structure. In order to do that, we first present the formal model of cycleway network, i.e., cycleway graph, originally developed in [5, 30]. Then, we extend this model in order to address the issue with sensitiveness of cyclists to difficult manoeuvres at intersections. As a consequence, we will be able to penalize actions such as left turns on frequented junctions. Furthermore, we define multi-criteria bicycle routing problem as a k-criteria graph search problem and specify criteria with appropriate cost functions. Criteria have been designed to cover important factors affecting a bike ride based on studies [6, 7] of real-world cycle route choice behavior.

3.1 Cycleway Graph

The cycleway network is represented by a directed weighted *cycleway graph* $G = (V, E, g, h, l, F, f, \overrightarrow{c})$, where

- V is the set of nodes representing start and end points (e.g., cycleway nodes) of cycleway segments,
- $E = \{(u, v) | (u, v \in V) \wedge (u \neq v)\}$ is the set of directed edges representing cycleway segments,
- the function $g : V \to \mathbb{R}^2$ assigns a latitude and a longitude values to each node $v \in V$,
- an altitude value is assigned to each node by the function $h : V \to \mathbb{R}$,
- the horizontal length of each edge $(u, v) \in E$ is given by the function $l : E \to \mathbb{R}_0^+$,
- F is the set of all possible *features* of edges reflecting certain properties (e.g., a surface of a cycleway segment, a road type)
- the function $f : E \to F^n$ returns the *features* associated with each edge $(u, v) \in E$. Note that the edge (u, v) can have multiple features assigned to it, thus $f((u, v)) \subseteq F$ with the number of elements in interval $1 \leq |f((u, v))| \leq |F|$.

- the cost of each edge $\overrightarrow{c} = (c_1, c_2, \ldots, c_k)$ is a k-dimensional vector of cost functions, where each cost function returns value for a criterion it represents

For the given edge $(u, v) \in E$, the cost vector $\overrightarrow{c} = (c_1, c_2, \ldots, c_k)$ contains only the non-negative values. Each element c_i of the vector $\overrightarrow{c}$ belongs to a criterion value and it is computed by a particular cost function $c_i : E \times F^n \to \mathbb{R}_0^+$. The function c_i integrates the influence of map features $f((u, v))$ together with length, elevation and other properties of the edge (u, v), with respect to i-th criterion. For example, the features indicating that the edge is a residential street with bad surface quality will have a negative influence on the speed of cyclists when travel time is considered as a criterion. The cycleway graph is directed due to the fact that some cycleway segments in the map are one-way only.

3.2 Extended Cycleway Graph

This section defines extension of the cycleway graph G to cover the structure of junctions. We present different approach than used in the literature (Section 2.4), e.g. an turn table. Contrary to turn table method, we aim to not change the execution of the standard graph search algorithms.

Therefore, let us define a set of junction nodes $V_{\text{junction}} \subseteq V$. We say that a node $v \in V_{\text{junction}}$ is a junction, if all of its incoming and outgoing edges represent a motor vehicle infrastructure and the number of its neighbours is greater than two. Each junction v is modelled as a small subgraph composed of a junction entrance node set $V_{\text{in}}(v)$, a junction exit node set $V_{\text{out}}(v)$, and an inner edge set $E_{\text{inner}}(v) = \{(v_i^{\text{in}}, v_j^{\text{out}}) | (v_i^{\text{in}} \in V_{\text{in}}) \wedge (v_j^{\text{out}}(v) \in V_{\text{out}}(v))\}$.

As shown in Figure 3.1, junction node v is extended to an eight node subgraph with the entrance node set $V_{in}(v) = \{v_1^{in}, v_2^{in}, v_3^{\text{in}}, v_4^{\text{in}}\}$ and exit node set $V_{\text{out}}(v) = \{v_1^{\text{out}}, v_2^{\text{out}}, v_3^{\text{out}}, v_4^{\text{out}}\}$, and for every node $v_i^{in} \in V_{\text{in}}(v)$, there exists a directed edge from v_i^{in} to $v_j^{\text{out}} \in V_{\text{out}}(v)$. We connect the subgraph to the graph G by replacing all v's incoming edges (x_i, v) , where $x_i \in V$, by the new incoming edges (x_i, v_i^{in}) and by replacing all v's outgoing edges (v, y_i), where $y_i \in V$ by the new outgoing edges (v_i^{out}, y_i) and the original junction node v is replaced by all created entrance nodes v_i^{in} and exit nodes v_i^{out}.

The extended cycleway graph $G' = (V', E', g, h, l, F', f, \overrightarrow{c})$ in contrast to G contains

- $V' = (V \setminus V_{\text{junction}}) \cup \{V_{\text{in}}(v) \cup V_{\text{out}}(v) \mid \forall v \in V_{\text{junction}}\}$
- $E' = (E \setminus \{\forall (x, v) \wedge (v, y) \mid x, y \in V, v \in V_{junction}\}) \cup E_{\text{inner}}$
- $F' = F \cup F_{\text{junction}}$, i.e. F' consists of all features F in the original graph G and features influencing ride through a junction. Junction features are furthermore described in Section 5.2.2

3.3 Multi-criteria Bicycle Planning Problem

The *multi-criteria bicycle planning problem* is defined as a triple $C = (G', o, d)$:

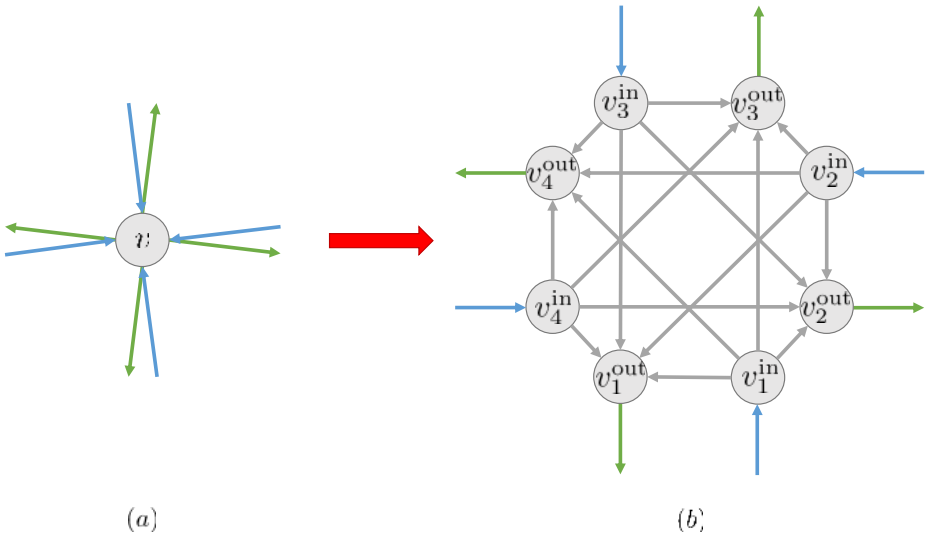


Figure 3.1: (a) junction node $v \in V_{\text{junction}}$, (b) extension of junction node v, in the graph incoming edges are coloured in *green*, outgoing edges in *blue* and inner edges in *gray*

- $G' = (V', E', g, h, l, F', f, \overrightarrow{c})$ is the *cycleway graph*
- $o \in V'$ is the route origin
- $d \in V'$ is the route destination

In order to determine a solution to the multi-criteria bicycle planning problem, we must define several terms first. A *bicycle route* $\pi(u, v)$ is a finite sequence of cycleway segments between a node $u \in V'$ and a node $v \in V'$ in the extended cycleway graph G'. A set of all such possible routes $\Pi(u, v)$ is then define as

$$\Pi(u, v) = \{\pi(u, v) \mid \pi(u, v) = \langle (u_1, u_2), \ldots, (u_{n-1}, u_n) \rangle\} \qquad u_i \in V',\ u_1 = u,\ u_n = v$$

In general π will indicate a bicycle route between arbitrary cycleway nodes and individual criteria values of the cost vector $\overrightarrow{c}(\pi)$ are equal to the sum of criteria values of its component cycleway segments:

$$\overrightarrow{c}(\pi) = \left(\sum_{j=1}^{|\pi|} c_1((u_j, u_{j+1})), \ldots, \sum_{j=1}^{|\pi|} c_k((u_j, u_{j+1})) \right)$$

Any bicycle route from the origin $o \in V'$ to the destination $d \in V'$ will be denoted as a *solution* bicycle route and referred as $\pi' = \pi(o, d)$. Then, a set of all possible solution bicycle routes will be characterize as

$$\Pi' = \{\pi' \mid \pi' = \langle (u_1, u_2), \ldots, (u_{n-1}, u_n) \rangle\} \qquad u_i \in V',\ u_1 = o,\ u_n = d$$

In Section 2.3 we define the *dominance* relation among vectors expressed by the symbol $\prec$. In particular, the cost vector $\overrightarrow{c}$ dominates other cost vector $\overrightarrow{c}'$, if and only if it is strictly better in at least one criteria value and not worse in any of criteria values. Thus, in multi-criteria cycle planning problem, a bicycle route π_p dominates another bicycle route π_q if and only if $\overrightarrow{c}(\pi_p) \prec \overrightarrow{c}(\pi_p)$.

In particular, imagine a bi-criteria problem, a cost vector (2,3) dominates (2,4), but no dominance relation exists between cost vectors (2,3) and (3,2) [17].

The solution of the multi-criteria cycle planning problem is a set of *non-dominated* bicycle routes Π^* defined in the following way,

$$\Pi^* = \{\pi_p^* \in \Pi' \mid \nexists \pi_q' \in \Pi' \wedge \overrightarrow{c}(\pi_q') \prec \overrightarrow{c}(\pi_p^*)\}, \quad \Pi^* \subseteq \Pi'$$

We will also refer to the set of non-dominated solutions Π^* as a *Pareto set*. Note that some algorithms may not find optimal Pareto set, i.e. there might exist such a sulution $\pi_q' \in \Pi'$, but the algorithm is not be able to find it.

3.3.1 Scalarised Multi-criteria Bicycle Planning Problem

Alternatively, we can specify the *scalarised* multi-criteria bicycle planning problem as a quadruple $C = (G', o, d, \overrightarrow{w})$. Additional vector $\overrightarrow{w} = (w_1, \ldots, w_k)$ represents the weights of the criteria (profile). This vector determines the importance of the individual criterion value $c_i \in \overrightarrow{c}$. A bicycle route π^* is then a solution to the multi-criteria cycle planning problem if and only if the solution π^* minimizes the total cost

$$c(\pi^*) = \sum_{j=1}^{|\pi^*|} \overrightarrow{w} \cdot \overrightarrow{c}((u_j, u_{j+1})), \qquad u_j, u_{j+1} \in V'$$

Note that the vector $\overrightarrow{w}$ must have the same dimension k as a cost vector $\overrightarrow{c}$. In other words, it must hold $|\overrightarrow{c}| = |\overrightarrow{w}| = k$.

3.4 Criteria and Cost Functions Definition

This section provides definitions of criteria suitable for bicycle route planning problem and their appropriate cost functions. We define the criteria based on the study of cyclists' route choice behaviour [6, 7].

3.4.1 Distance

The study [6] reveals the fact that commuting cyclists are relatively more sensitive to distance and less sensitive to most other variables. The *distance* criterion represents a cycleway segment's length in a real-world. It describes, how far cyclists will travel in order to get from one a cycleway node $u \in V'$ to another cycleway node $v \in V'$. It is the simplest possible criterion, but it preserves the fact that cyclists will mostly choose the shortest route among two equally comfortable ones. The cost function for the distance criterion is defined in the following way:

$$c_{\text{dist}}((u,v)) = l((u,v))$$

3.4.2 Travel time

The *travel time* criterion captures the preference towards routes that can be travelled in a short time. Travel time is a sensitive factor in cyclists' route planning especially for commuting purposes. In order to express how much will a certain set of features $f((u,v))$ decreases a cyclist's speed on a given edge $(u,v) \in E$, we develop function $r_{\text{tt}} : F^n \to \mathbb{R}_0^+$. A road covered by cobblestones is a typical example of a feature that slows down cyclists. Further, to model the slow down caused by obstacle features such as stairs or crossings (e.g., an additional constant time is needed before each crossing of a road or for a waiting time at traffic lights), we define the slowdown function $q : F^n \to \mathbb{R}_0^+$ which returns the slowdown in seconds on the given edge $(u,v) \in E$ with a set of features $f((u,v))$.

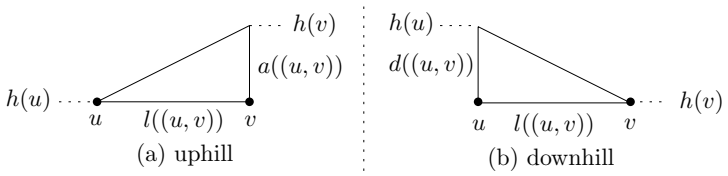


Figure 3.2: (a) Positive vertical ascend a. (b) Positive vertical descend d. Source [30]

Besides, changes in the elevation may affect the cyclist's speed and hence affect travel times. Going uphill reduces speed and require additional energy. On the other hand, riding downhill usually means a speed-up, the possibility to stop pedaling and rest. For the case of uphill rides, we define the positive vertical ascend $a : E \to \mathbb{R}_0^+$ (cf. Figure 3.2) and the positive ascend grade $a' : E \to \mathbb{R}_0^+$ for a given edge $(u,v) \in E$ as follows:

$$a((u,v)) := \begin{cases} h(v) - h(u) & \text{if } h(v) > h(u) \\ 0 & \text{otherwise} \end{cases}$$

$$a'((u,v)) := \frac{a((u,v))}{l((u,v))}$$

Similarly, for the case of downhill rides, we define the positive vertical descend $d : E \to \mathbb{R}_0^+$ (cf. Figure 3.2) and the positive descend grade $d' : E \to \mathbb{R}_0^+$ for a given edge $(u,v) \in E$ as follows:

$$d((u,v)) := \begin{cases} h(u) - h(v) & \text{if } h(u) > h(v) \\ 0 & \text{otherwise} \end{cases}$$

$$d'((u,v)) := \frac{d((u,v))}{l((u,v))}$$

To model the speed acceleration caused by vertical descend for a given edge $(u,v) \in E$, we define the downhill speed multiplier $s_d : E \times \mathbb{R}^+ \to \mathbb{R}^+$ as:

$$s_d((u,v), s_{\text{dmax}}) := \begin{cases} s_{\text{dmax}} & \text{if } d'((u,v)) > d'_{\text{c}}, \\ \frac{(s_{\text{dmax}}-1)d'((u,v))}{d'_{\text{c}}} + 1 & \text{otherwise} \end{cases}$$

where $s_{\text{dmax}} \in \mathbb{R}^+$ is the maximum downhill speed multiplier, and $d'_{\text{c}} \in \mathbb{R}^+$ is the critical d' value over which a downhill ride would use the multiplier of s_{dmax}. This reflects the fact that the speed acceleration is remarkable for the ride on a steep downhill (compared to a mild one), however, it is limited due to safety concerns, bicycle physical limits and air drag.

Considering the integrated effect of edge length, the change in elevation and its associated features, the travel time criterion is defined as:

$$c_{\text{tt}}((u,v)) = \frac{l((u,v)) + a_l \cdot a((u,v))}{s \cdot s_d((u,v), s_{\text{dmax}}) \cdot r_{\text{tt}}(f((u,v)))} + q((u,v)),$$

where s is the average cruising speed of a cyclist, and a_l is the penalty coefficient for uphill rides. Intuitively, $c_{\text{tt}}((u,v))$ can model the travel time of flat rides, uphill rides, and downhill rides with $s_d((u,v), s_{\text{dmax}}) = 1$ for uphill and flat scenarios and $a((u,v)) = 0$ for downhill and flat scenarios.

3.4.3 Comfort

The *comfort criterion* aims to provide a comfortable route from an origin to a destination. This criterion penalizes bad road surface, hard turns (especially left turn from main road to side road), obstacles such as steps, and places where the cyclist needs to dismount his or her bicycle.

To identify how much a given edge $(u,v) \in E$ is or is not comfortable, we define function $r_{\text{co}} : F^n \to \mathbb{R}_0^+$. The purpose of the function r_{co} is to penalize edges containing features disturbing comfortability of a ride. Edges with small values of r_{co} indicates great comfort for riding. It is based on the recognition that cyclists feel their travel time is shortened as they relish travel on comfortable cycleways (r_{co} is lower than 1). After all, the r_{co} value is weighted by the travel time c_{tt} spent on traversing the edge.

To cover hardness of manoeuvres on a particular junction, we define function $p : F^n_{\text{junction}} \to \mathbb{R}_0^+$ returning what distance are cyclists willing to travel additionally, in order to avoid an intersection. To keep a value of the criterion in seconds, we divide a value of the function $p(f(u,v))$ by cyclist's average cruising speed.

The comfort cost function is therefore defined as:

$$c_{\text{co}}((u,v)) = c_{\text{tt}}((u,v)) \cdot r_{\text{co}}(f((u,v))) + \frac{p(f((u,v)))}{s}$$

3.4.4 Quietness

The goal of the *quietness criterion* is to find a quiet route with low traffic or an absence of traffic. The function $r_{\text{qu}}(f((u,v))) : F^n \to \mathbb{R}_0^+$ assigns small values to edges representing an infrastructure dedicated for cyclists and large values to motor roads and crossings. The value of r_{qu} is also weighted by the travel time c_{tt} spend on traversing the edge. Such function for quietness is defined as:

$$c_{\text{qu}}((u,v)) = c_{\text{tt}}((u,v)) \cdot r_{\text{qu}}(f((u,v)))$$

3.4.5 Elevation Gain

The *elevation gain* criterion captures the preference towards flat routes with minimum uphill segments. The criteria function for elevation gain takes into account the positive vertical ascend a and it penalizes uphill rides by the equivalent flat distance $a_l \cdot a((u,v))$ of the segment with a vertical ascend of $a((u,v))$, and with uphill penalty coefficient a_l similarly defined in Section 3.4.3. The cost function for the elevation gain is defined as:

$$c_{\text{eg}}((u,v)) = \frac{a_l \cdot a((u,v))}{s}$$

In case of flat or downhill cycleway segments $c_{\text{eg}}((u,v)) = 0$.

3.4.6 Bike Friendliness

The *bike friendliness* criterion captures the preference towards routes suitable for cycling, i.e., a good-quality surface, a low traffic and a dedicated infrastructure.

We use functions $r_{\text{co}}((u,v))$, $p(f((u,v)))$ and $r_{\text{qu}}((u,v))$ defined in Section 3.4.3 and Section 3.4.4. The comfort criteria feature function $r_{\text{co}}((u,v))$ is employed to penalize bad road surfaces, obstacles such as steps, and places where the cyclist needs to dismount his/her bicycle, with small values indicating cycling-friendly surfaces. The quietness criteria feature function $r_{\text{qu}}((u,v))$ measures traffic volumes by considering the infrastructure for cyclists (e.g., dedicated cycleways), the type of roads, and the junctions, where low-traffic cycleways are assigned a small coefficient value. The maximum value returned by $r_{\text{co}}((u,v))$ and $r_{\text{qu}}((u,v))$ is used here to avoid the cycleway segments that negatively affect suitability for cyclists. The maximum is then weighted by edge length $l((u,v))$, i.e., 500 m of cobblestones is worse than 100 m of cobblestones. To capture complexity of ride through a junction, we add the value returned by function $p(f((u,v)))$ defined in Section 3.4.3. Therefore, the final cost function for bike friendliness criterion is defined as:

$$c_{\text{suit}}((u,v)) = \max\Big(r_{\text{co}}((u,v)), r_{\text{qu}}((u,v))\Big) \cdot l((u,v)) + p(f((u,v)))$$

Chapter 4

Solution Method

This chapter illustrates how the multi-criteria bicycle routing problem is solved. First, we present a solution approach using a single cost shortest path algorithm and then we introduce a multi-criteria search algorithm in order to obtain the full Pareto set of bicycle routes.

4.1 Scalarised Multi-Criteria Solution Method

First of all, we solve scalarised multi-criteria cycle planning problem $C = (G', o, d, \overrightarrow{w})$ (defined in Section 3.3) by employing single cost shortest path Dijkstra algorithm.

A single cost value is defined as a dot product of a cost vector of criteria values $\overrightarrow{c}$ and a profile $\overrightarrow{w}$

$$c((u,v)) = \overrightarrow{w}((u,v)) \cdot \overrightarrow{c}((u,v)), \qquad u, v \in V'$$

When searching for a bicycle route, users may have various preferences as it is summarized in section 2.1. In order to the cover the majority of cyclists' preferences, we decide to use the following four criteria presented in section 3.4: Travel Time, Comfort, Quietness and Elevation Gain. These criteria are the best candidates to be used in a single cost shortest path algorithm, because they all returned values with the same unit (seconds). Therefore, they can be incorporated together in a weighted sum.

4.1.1 Profiles

A *profile*, i.e., a vector $\overrightarrow{w}(u,v) = (w_1, \ldots, w_k)$ of criteria weights $w_i \in \mathbb{R}_0^+$ for an edge $(u,v) \in E'$, reflects the fact that for various users, certain factors of the path have different importance. Therefore, we define four profiles, each providing a bicycle route relevant for different types of cyclists. In our case, the profile consists from four elements, where each w_i belongs to one of the selected criteria: travel time, comfort, quietness, and elevation gain.

The *Fast* profile $\overrightarrow{w} = (1, 0, 0, 0)$ uses only travel time criterion to provide a route with the shortest possible duration. A solution to this profile might have contain bad road surface, steps or steep cycleway segments. Though, users of this profile should be experienced riders equipped with a bicycle that can handle worse conditions.

The *Commuting* profile $\overrightarrow{w} = (3, 5, 1, 1)$ is designed for people who use a bicycle for daily traveling to work or school. This profile attempts to find a comfortable and quick, but also reasonably quiet and flat route. A solution to this profile is a quick route prioritizing good surface and avoiding frequented roads and steep ascends where possible.

The *Bike friendly* profile $\overrightarrow{w} = (0, 1, 3, 1)$ is proposing a path that is primarily quiet (prefers designated bicycle routes and avoids high traffic roads) and secondary comfortable and without steep climbs. In contrast with the commuting profile, this profile is designed for non-commuting people who usually do not tolerate sharing a road with motor vehicles.

The *Flat* profile $\overrightarrow{w} = (0, 1, 1, 8)$ is designed for cyclists that want to avoid going uphill as much as possible. The other comfort and quietness criteria are also considered to provide a route suitable for cyclists.

4.2 Multi-Criteria Solution Method

To solve the multi-criteria cycle planning problem $C = (G', o, d)$, we employ the optimal multiobjective version of Dijkstra's algorithm, i.e. the multi-criteria Dijkstra (MCD) algorithm. It uses several data structures: a label L defined as triple $(u, \overrightarrow{c}(\pi(o, u)), L_{\text{pred}})$, where u denotes a node to which L is assigned to, the cost vector $\overrightarrow{c}(\pi(o, u))$ describes criteria values of bicycle path from origin o to node u and L_{pred} is L's predecessor label. We store the previous label to be able to reconstruct a solution, i.e., a bicycle route $\pi(o, d)$ from origin o to destination d. Moreover, we define the multi-set data structure $Bag(u)$ for each node $u \in V'$ to maintain the non-dominated labels pointing to u. For managing all unexplored labels created during the search, we define a priority queue Q. Labels are in a lexicographical order by criteria values in the cost vector.

The pseudocode of the MCD algorithm is given in Algorithm 1 except for the line 13 which will be introduced in a following subsection. First, we initialize priority queue Q and $Bag(u)$ for each node u from the set of all nodes V'. Then we create an initial label L_{origin} assigned to an origin node o, with a zero cost vector and no predecessor label. We insert this label to the priority queue Q and the multi-set $Bag(o)$.

Algorithm terminates when there is no label left in Q. It returns content of multi-set $Bag(d)$ which is equal to $sol(\Pi')$.

On the other hand, if queue Q contains some label L_{current}, we expand it by taking a node u to which is assigned and for each edge (u, v), we compute new cost vector $\overrightarrow{c}(\pi(o, v))$ by adding criteria values of the edge $\overrightarrow{c}((u, v))$ to the current label cost vector $\overrightarrow{c}(\pi(o, u))$. Using the node v, the cost vector $\overrightarrow{c}(\pi(o, v))$, and the predecessor label L_{current} we create a new label L_{next}.

Function `checkDominance` called in Algorithm 1 at line 14 and described in Algorithm 2, controls the dominance relation $\prec$ between the cost vector of label L_{next} and all labels inside $Bag(v)$. If cost vector of L_{next} is not dominated by any of the cost vectors of labels pointing to node v, the algorithm inserts it into $Bag(v)$ and Q. Also, if some label L inside $Bag(v)$ is dominated by L_{next}, it will be eliminated from the multi-set data structure and not considered in future search.

Algorithm 1: Multi-criteria Dijkstra algorithm

Input: extended cycleway graph $G' = (V', E', g, h, l, F', f, \overrightarrow{c})$,
origin node $o \in V'$, destination node $d \in V'$
Output: full Pareto set of labels

$Q :=$ *empty* **priority queue**
$Bag(\forall v \in V') :=$ *empty* **set**
$L_{\text{origin}} := (o,\ (0, 0, \ldots, 0),$ **null**$)$
$Q.insert(L_{\text{origin}})$
$Bag(o).insert(L_{\text{origin}})$
while *Q is not empty* **do**
  $L_{\text{current}} := Q.pop()$
  $u := L_{\text{current}}.getNode()$
  $\overrightarrow{c}(\pi(o,u)) := L_{\text{current}}.getCost()$
  foreach *outgoing edge of* $u,\ (u,v) \in E'$ **do**
    $\overrightarrow{c}(\pi(o,v)) := (\ c_1(\pi(o,u)) + c_1((u,v)), \ldots, c_k(\pi(o,u)) + c_k((u,v))\ )$
    $L_{\text{next}} := (v, \overrightarrow{c}(\pi(o,v)), L_{\text{current}})$
    if *skip*(L_{next}) **then continue**
    if *checkDominance*(L_{next}) **then**
      $Bag(v).insert(L_{\text{next}})$
      $Q.insert(L_{\text{next}})$
    end
  end
end
return *Bag(d)*

Algorithm 2: *function* checkDominance(label L_{next})

$v := L_{\text{next}}.getNode()$
$\overrightarrow{c}'(\pi(o,v)) := L_{\text{next}}.getCost()$
foreach *label* $L \in Bag(v)$ **do**
  $\overrightarrow{c}(\pi(o,v)) := L.getCost()$
  if $\overrightarrow{c}(\pi(o,v)) \prec \overrightarrow{c}'(\pi(o,v))$ **then return** *false*
  if $\overrightarrow{c}'(\pi(o,v)) \prec \overrightarrow{c}(\pi(o,v))$ **then** *remove L from $Bag(v)$ and Q*
end
return *true*

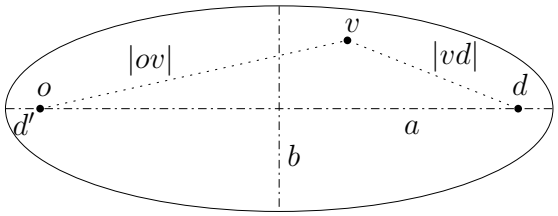


Figure 4.1: Geometry of the ellipse pruning condition.

4.2.1 Pruning Enabled Multi-criteria Dijkstra algorithm

As mentioned in Section 2.3, the multiple criteria search problem is computationally much harder than the single criterion one. Therefore, in order to decrease computational complexity of the MCD algorithm, we introduce a *pruning enabled* multi-criteria label setting algorithm. We add an invocation of `skip` function in Algorithm 1 at line 13 (highlighted by green colour). The pruning enabled algorithm contains only one additional line in comparison to the original algorithm as shown in Algorithm 1. Modification of functions `checkDominance` and `skip` let us to speed up the search process. In the next subsections, we describe these modifications and its potential advantages and disadvantages.

4.2.1.1 Ellipse

The first pruning heuristic, inspired by [20], prevents multi-criteria Dijkstra algorithm from searching the whole cycleway graph. This is very crucial, because MCD always expands all cycleway nodes at least once (often multiple times, due to more non-dominant path) no matter the distance between an origin and a destination node. In contrast to single criterion Dijkstra algorithm, MCD cannot stop search when it reaches the destination. There is no guarantee that the solution will not be dominated by any other solution found later. The heuristic permits visiting only the nodes that are within a predefined ellipse. Unfortunately, the method may miss some of the solution's from an optimal Pareto set of bicycle routes Π^* and yield to $sub(\Pi^*)$. In the worst case, in some unusual cycleway network topology it might not find a solution at all.

As shown at Figure 4.1, the focal points of the ellipse correspond to the origin o and the destination d of the bicycle route. We derive a length of a semi-major axis a and a semi-minor axis b from a constant ratio $\frac{a}{b}$, obtained as an input parameter for the ellipse method. The distance between origin o and a peripheral point on the main axis of the ellipse is represented by d'. In addition, to improve the ellipse's performance for short origin-destination distances, we keep a minimum value $d'_{\min}$ for the length of d'. During the search, the function `skip` checks whether an edge (u, v) has its target node v inside the ellipse. This is done by checking the inequality $|ov|+|vd| \leq 2a$, in other words, node v lies inside the ellipse if sum of distances between v and o and v and d is lower or equal to two times the length of a semi-major axis a.

4.2.1.2 ϵ-dominance

We use the method initially proposed in [21], to approximate the set of Pareto optimal bicycle routes. The approach guarantees to find a solution that is within a factor of $(1 + \epsilon)$ from the optimal solution. Approximation of Pareto optimality relies on the use of ϵ-dominance relation between cost vectors defined as

$$\overrightarrow{c}(\pi_p) \prec_\epsilon \overrightarrow{c}(\pi_q) \iff \overrightarrow{c}(\pi_p) \prec (1+\epsilon) \cdot \overrightarrow{c}(\pi_q)$$

Difference between dominance and ϵ-dominance in a bi-criteria problem is illustrated in Fig. 4.2. All points filled in *grey* colour at (a) are dominated by the *black* point, respectively, all *grey* points at (b) are ϵ-dominated by the *black* point.

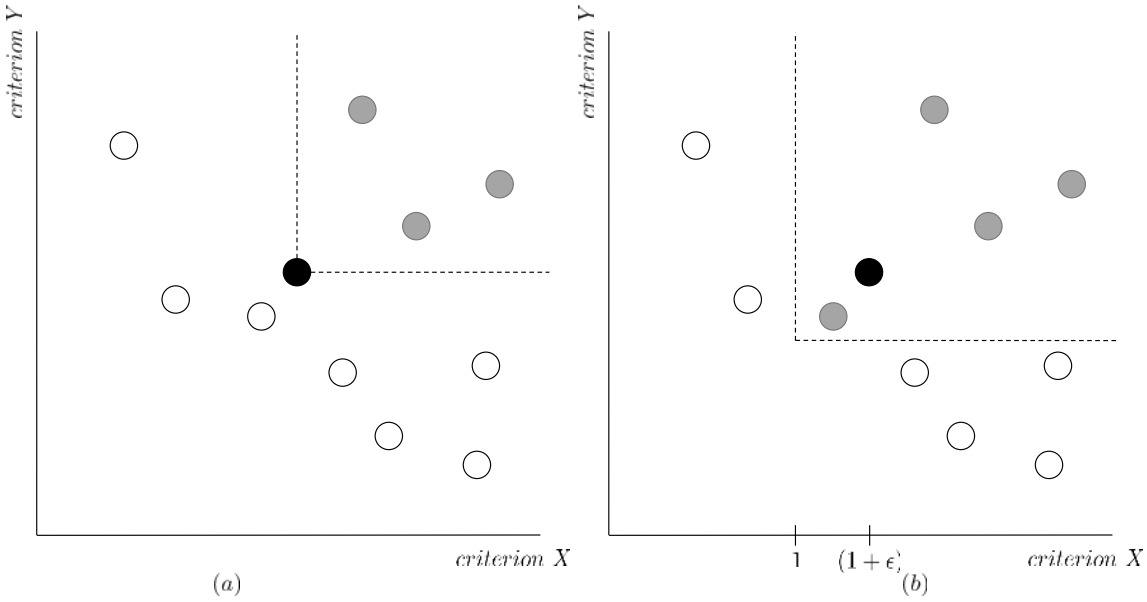


Figure 4.2: *a*) dominance relation, *b*) ϵ-dominance relation

To integrate this method to MCD algorithm we exchange $\prec$ relation to $\prec_\epsilon$ relation in the `checkDominance` function.

4.2.1.3 Buckets

Buckets method originally defined in [22] discretizes the cost space using buckets for the criteria values. Figure 4.3 indicates how cost vectors are mapped into buckets. In epsilon dominance method, we set only one ratio for all criteria, but in buckets method, we are able adjust the bucket size for individual criteria.

The speed up is executed in the `checkDominance` function (cf. pseudocode in Algorithm 3). The function `bucketValue` assigns a bucket to each criteria value based on defined bucket size for a particular criterion.

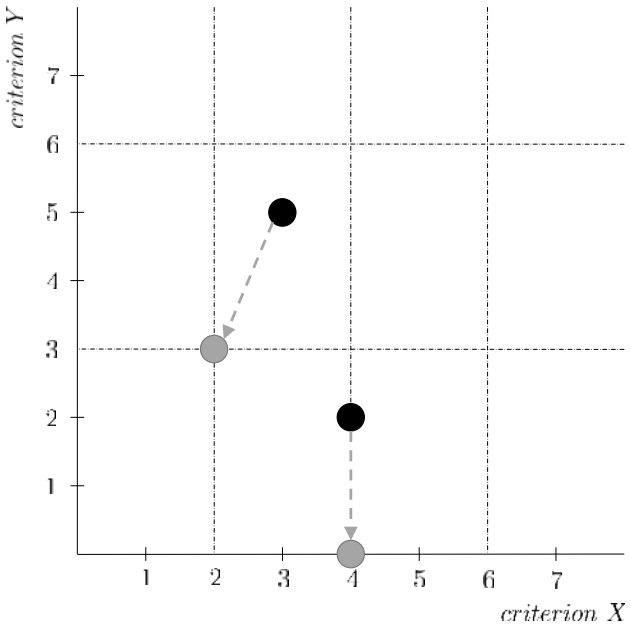


Figure 4.3: Mapping a point in cost space to buckets. Bucket size for criterion X is equal to two and for criterion Y is equal to three.

Algorithm 3: Buckets: *function* checkDominance(label *next*)

$v := L_{\text{next}}.getNode()$
$\overrightarrow{c}'(\pi(o,v)) := L_{\text{next}}.getCost()$
foreach *label* $L \in Bag(v)$ **do**
  $\overrightarrow{c}(\pi(o,v)) := L.getCost()$
  if $bucketValue(\overrightarrow{c}(\pi(o,v))) \prec bucketValue(\overrightarrow{c}'(\pi(o,v)))$ **then**
    return *false*
  end
  if $bucketValue(\overrightarrow{c}'(\pi(o,v))) \prec bucketValue(\overrightarrow{c}(\pi(o,v)))$ **then**
    remove L *from* $Bag(v)$ *and* Q
  end
end
return *true*

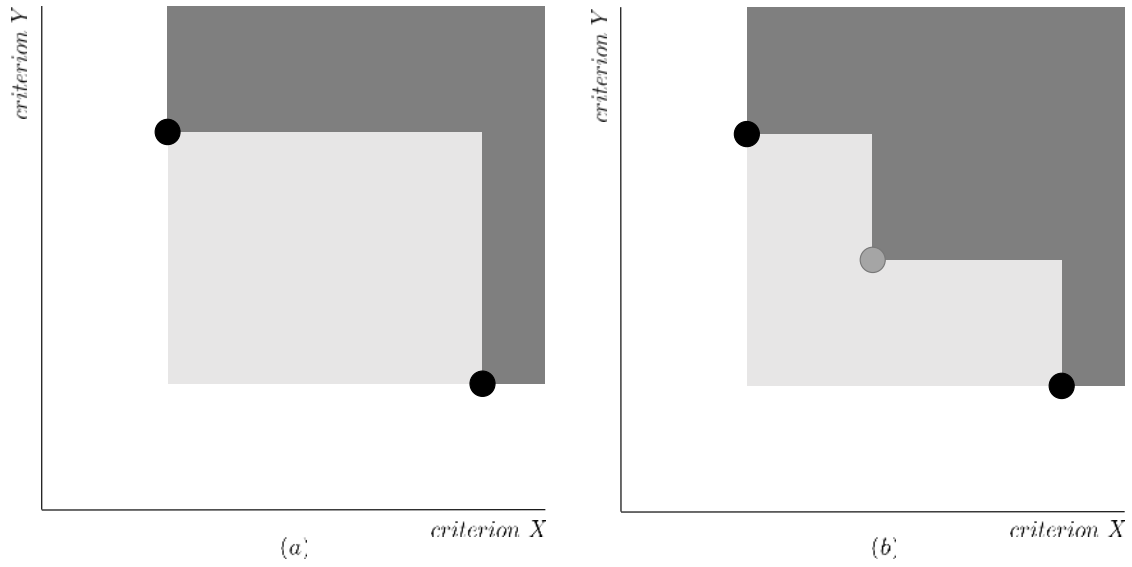


Figure 4.4: *black* points represent single cost solutions, *light grey* area is a available cost space for solutions, and *dark grey* represents area where no label will be expanded and is defined by (a) single cost solutions, and by (b) a found solution (*grey* point)

4.2.1.4 Bounded search

The last pruning method bounds cost space, in order to not let MCD expands labels with no chance to reach Pareto set of bicycle routes Π^*. We introduce a set of cost vectors $COSTS$, similarly as in MOA* [23] or NAMOA* [17], that records all cost vectors found for destination node. Once a solution is known, its cost vector can be used to prune labels. To bound a search space from start of the MCD, we solve a single criteria bicycle routing problem for each criterion individually and insert resulted cost vectors to the $COSTS$ set. The computational cost for solving the single criteria solutions is small opposed to the benefits it brings to MCD.

Illustration of the bounded cost space for bicycle routes is shown in Fig. 4.4.

We integrate the method in function `skip`, where the algorithm checks whether a cost vector of label L_{next} is not dominated by any of vectors from $COSTS$ set. If there exists such a solution cost vector, the algorithm will no longer consider L_{next} for future search.

Chapter 5

Implementation

This chapter provides an inside of important parts of the *cycle planner*. We show what kind of data we use, and how they are processed in order to build the cycleway graph G' defined in Section 3.2. We also describe what values are returned by functions defined for cost functions in Section 3.4, based on features obtained from the map data. The insight into main part of the cycle planner application is presented in Section 5.3.

5.1 Data

OpenStreetMap To build the cycleway graph G', we need data containing information about a real world route network. To fulfil this requirement, we are going to work with an OpenStreetMap (OSM) project that creates and provides *free* geographical data of Earth. It is being built by volunteers and is released with the open-content license.

OSM data are distributed in various formats. The basic one, *osm* is an XML format and due to its large size is mainly distributed compressed. The most popular format is *pbf*, which is a binary format and in general takes half of the size of the compressed *osm* file. When we combine advantages of *osm* and *pbf* we obtain *o5m* format which provides high-speed processing that uses *pbf* coding and has the same structure as *osm*. Moreover, there exists other formats, e.g., *OSM JSON* or *Level0L*, but they are not so widely used and not all OSM tools support them.

To obtain OSM data, we can either download *planet.osm* file containing all data of Earth (approximately 39GB in compressed *osm* format) or smaller files called *extracts*, containing data for individual continents, countries, and metropolitan areas[1].

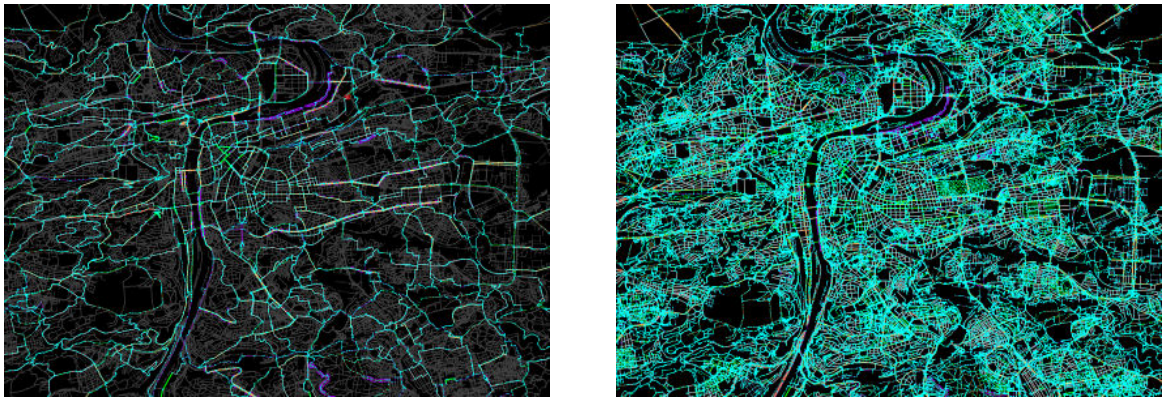
OSM data consists of three basic elements. *Node* represents a point in space, *way* defines a linear feature (road or boundary) or area (river, square or building). Third element, *relation* consists of an ordered list of one or more members of the relation (nodes, ways, and/or other relations) and describes logical or geographic relations between its members. A typical example of relation is a bus route, official bicycle route, or multipolygon denoting an island or a lake. All mentioned OSM elements can contain various *tags*. The *tag* describes the features of the element to which are attached and consists of two text attributes key and

[1] http://wiki.openstreetmap.org/wiki/Planet.osm

value. We denote the tags using triple `element::key::value`, e.g., `way::highway::primary` represents a primary road that often links larger towns.

OpenStreetMap project groups together a massive number of people, whether users, contributors, or developers. The community develops a variety of tools for creating, updating, rendering and processing OSM data.

iD and Potlach 2 are online map editors written in JavaScript and Flash, respectively. Both are available from OSM website[2] under menu item "Edit". For analysing OSM, we use desktop map data editor *JOSM*[3], where we are able to inspect extracts of OSM data created by ourselves. JOSM also extensively supports filters that give a user option to disable unnecessary OSM elements. In Figure 5.1 a left part a shows only official bicycle routes and b shows all road segments where is possible to ride a bicycle. Both pictures are showing a filtered OSM file in JOSM application for the Prague city region.



(a) (b)

Figure 5.1: (a) a official bicycle routes in Prague in OSM (b) cycleway network in Prague in OSM

OSM data preprocessing A majority of OSM data is not essential for constructing the cycleway network (e.g., buildings, region boundaries, rivers, etc.). The main intention is to preserve only part of a physical world suitable for a bicycle ride. In other words, we are looking for a subset of OSM data, that forms a cycleway network.

After detailed study of all OSM map features[4] represented by *tags* and analyzing map data with JOSM, we specify guidelines for extracting cycleway network from OSM:

[2] http://www.openstreetmap.org/
[3] https://josm.openstreetmap.de/
[4] http://wiki.openstreetmap.org/wiki/Map_Features

- Keep only objects attached with tag `node::highway::*` or `way::highway::*` except those with tags `way::highway::motorway`, `way::highway::trunk` where bikes are disallowed by legislation and with `way::highway::proposed`, `way::highway::construction` representing roads in plan or under construction. Also, we exclude ways associated with `way::highway::bus_guideway` and `way::highway::raceway`, these roads are not suited for regular traffic by design.
- Keep all elements with tags `way::highway::*_link`. For navigation purposes, we want to preserve correct number of exits on a roundabout or a junction.
- Elements containing a tag `way::highway::pedestrian` are allowed only in combination with tag `way::bicycle::yes`
- Exclude elements with a tag `way::motorroad::yes`. This tag is used to describe roads having motorway-like access restrictions but are not motorways.
- Exclude elements with a tag `way::bicycle::no`
- Exclude elements with a tag `way::area::yes`, these are declaring boundaries of closed space e.g. square. Therefore, they are not relevant for routing purposes.
- Exclude `way::route::ferry` or `relation::route::ferry`. For effective planning it would require to integrate a timetable.
- Keep ways containing one of the following tags `way::access::no`, `way::access::private` or `way::access::customers` only in mix with `way::bicycle::yes`, `way::bicycle::permissive` or `way::bicycle::dismount`.
- Avoid triple of tags `way::highway::steps`, `way::tunnel::yes` and `way::layer::*`, where value for key=`layer` is lower than zero. Objects attached with this triple describe steps to an underground station.
- Exclude elements representing underground parking garages. These can be detected by `way::highway::service` and `way::layer::*`, where value for key=`layer` is lower than zero.
- Keep all members of relation with a tag `relation::route::bicycle`, which represents either official or proposed cycle routes (proposed by cycling communities, e.g., Auto*Mat, and maintained by local authorities).
- Exclude all other relations than those with tags `relation::route::bicycle` or `relation::type::restriction`.
- Remove ways not recommended for bicycle ride by professionals or experienced cyclists (e.g., roads such as Sokolovská, Legerova in Prague). We enumerate such ways by hand and filter them from the final OSM by their identifiers.

Another tool that we use to process OSM data files is command line Java application *Osmosis*[5]. We focus on routing in city environment, therefore we primarily use the Osmosis

[5] http://wiki.openstreetmap.org/wiki/Osmosis

to crop country size OSM extracts to contain only elements for supported cities. In example below, we show how to use the Osmosis application for cropping the city of Prague area from the Czech Republic OSM extract.

```
$ ./osmosis --rx file=czech-republic.osm --bounding-box left=14.333400
right=14.550400 top=50.136200 bottom=50.022500 completeWays=yes
--used-node --wx prague.osm
```

Parameter `completeWays=yes` ensures that for ways that have at least one node in the bounding box, all their nodes will be included as well. Also, we employ the Osmosis in the version 0.43.1 as a library within cycle planner Java application to help manipulate OSM objects in Java code.

Osmfilter[6] is a free command line application created to filter OSM data files by specific tags. Application lets the user to define various criteria to select OSM objects, which are currently in his/her area of interest and either to keep only selected objects or to remove them from an output OSM file. As input and output format, it supports *osm* and *o5m* file formats. The best practise to fast data processing is to use the compact *o5m* format at least as an input file. In order to convert our data file to *o5m* format we use another lightweight tool called *osmconvert*.

We use osmfilter to apply guidelines defined above to obtain OSM data representing only the cycleway network. Filtered OSM data file in the *osm* format has a size of 86% of the original OSM, which makes it important for loading to cycle planner Java application. The lower the size of the input file, the faster it becomes to build a planning graph. The size will be reduced even more, if we delete redundant information such as author, current version, and time of the last modification from all OSM elements.

An example below shows a usage of osmfilter tool. First argument is the input file, in this case *europe.o5m*. Then, we declare that filter should keep all nodes/ways/relations containing tag with the key equals to `highway` without restriction to the value. Next, from all preserved objects, osmfilter must drop those containing tag with key=`access` and value=`no`. At last, we have to specify output file *goodways.osm*.

```
$ ./osmfilter europe.o5m --keep="highway=" --drop="access=no" -o=goodways.osm
```

To summarize, this filter example preserves all OSM objects representing road network on Europe continent with allowed access rights.

Osmfilter supports supplying parameters through a separate file. This useful feature allows to specify multiple parameters in a more complex way and still preserves readability and comprehensibility of the filter. File containing parameters to select cycleway network from OSM data is shown in Figure 5.2 and it is structured in a following way. First, we specify all object types (nodes, ways, and relations) which will be kept if they meet the filter criteria based on tags and then all objects which will be dropped regardless of meeting the keep filter criteria. All specified criteria apply to dependent objects, e.g., nodes in ways, ways in relations, relations in other relations. In the end, we determine output file format. We use *osm*, because the main cycle planner Java application uses the Osmosis library for

[6] http://wiki.openstreetmap.org/wiki/Osmfilter

parsing OSM data files which does not support *o5m* format. On the other hand, *osm* format is human readable, which is a great advantage. To run osmfilter with a parameter file from Figure 5.2, we use following command

```
$ ./osmfilter prague.osm --parameter-file=cycle-osmfilter.params
-o=cycle-prague.osm
```

Shuttle Radar Topography Mission (SRTM) SRTM is an international research effort that obtained digital elevation models on a near-global scale to generate the most complete high-resolution digital topographic database of Earth. Currently, available elevation model of Europe is sampled at 3 arc-seconds which is 1/1200th of a degree of latitude and longitude, or about 90 meters. In most cases such a resolution is sufficient for routing purposes. To import SRTM data to OSM file we use osmosis application which provides a plugin for attaching elevation to OSM nodes.

```
$ ./osmosis --rx file=cycle-prague.osm --write-srtm --wx
srtm-cycle-prague.osm
```

5.2 Graph construction

The goal of this section is to demonstrate the construction of the graph G' using OSM data in the cycle planner Java application. First, we construct a *general graph* which contains all necessary data in a raw format and is designed to easily manipulate with nodes and edges (optimised for `add` and `remove` operations). Also, we present a couple of methods on top of the general graph in order to prepare the data for the bicycle planning problem. Then from the general graph, we construct a *bicycle planning graph* which contains only processed data, i.e., values for individual criteria (e.g., the travel time, the comfort, etc.). This graph is later used by planning algorithms.

5.2.1 General Graph

We define two objects: `GeneralNode` and `GeneralEdge`, where the `GeneralNode` is an equivalent to a cycleway node $u \in V'$ and the `GeneralEdge` is an equivalent to a cycleway segment $(u, v) \in E'$ in the extended cycleway graph G'.

Geographical location of the `GeneralNode` is represented by latitude and longitude values and stored in two coordinates systems. The first is the World Geodetic System (version WGS 84), a standard geographic coordinate system type. The second is a projected coordinate system, in particular for the Czech Republic we use the spatial reference system "S-JTSK (Ferro) / Krovak". Storing coordinates' projected values is beneficial in case of computing distance between two nodes. Calculation of *Euclidean* distance is over a magnitude faster than computing *Harvesine* distance, which gives exact *great circle* distance between two points represented by latitude and longitude values and takes into account curvature and average radius of the Earth. To be able to compute an uphill or a downhill slope (Section 3.4.2), we save the elevation along with location information. Moreover, the `GeneralNode` stores tags

```
--keep=
( highway=primary or
highway=secondary or
highway=tertiary or
highway=unclassified or
highway=residential or
highway=service or
highway=*_link or
highway=living_street or
( highway=pedestrian and bicycle=yes ) or
highway=track or
highway=road or
highway=footway or
highway=cycleway or
highway=bridleway or
highway=steps or
highway=path ) and
( ( access!=private and access!=no ) or
( ( access=private or access=no ) and
( bicycle=yes or bicycle=permissive or bicycle=dismount ) ) )

--keep-relations=
route=bicycle

--drop=
( motorroad=yes and highway!=*_link ) or
route=ferry or
bicycle=no or
area=yes or
( highway=steps and tunnel=yes and layer<0 ) or
( highway=service and layer<0 ) or
access=customers

--drop-relations=
type= and
type!=route and
route!=bicycle

--out-osm
```

Figure 5.2: *cycle-osmfilter.params* – parameters file for osmfilter tool

of a corresponding OSM node and tags of relations of which is the node member. The same applies to the `GeneralEdge`, but instead of node it stores tags of corresponding ways and relations.

In the following paragraphs, we present processing and optimising methods for the graph consisting of `GeneralNode` and `GeneralEdge` objects in order to obtain a graph suited for planning purposes.

Delete loops It is really common that OSM data contains faults which are not visible on the rendered maps, but are crucial for routing purposes. Such a case are edges with the same start node and end node. We declare, that nodes are the same if they have either equal OSM identifier or geographical coordinates. This kind of edges are redundant in cycle planning problem, because they have a zero length.

Traffic lights positioned out of junction We penalize traffic signals and pedestrian crossings by the static slowdown constant (Section 3.4.2). In OSM, there exist cases where traffic signals for motor vehicles are represented by a node before a pedestrian crossing node. In other words, there may occur a sequence of connected nodes in the following order: a traffic light node, a pedestrian crossing node (*zebra*), and a junction node. This causes a phenomena that for cyclists would be more profitable to use a pavement and then enter a junction from a zebra as shown in Figure 5.3. Consequently, cyclists would avoid the traffic light node containing the slowdown constant.

To solve this issue, we decide to move the traffic light node closer to a junction. Specifically, if there exists an edge between the traffic light node associated with tag `node::highway::traffic_signals` and with only two neighbours, and the zebra node, then we move all tags related to traffic signals from the traffic light node to the zebra node.

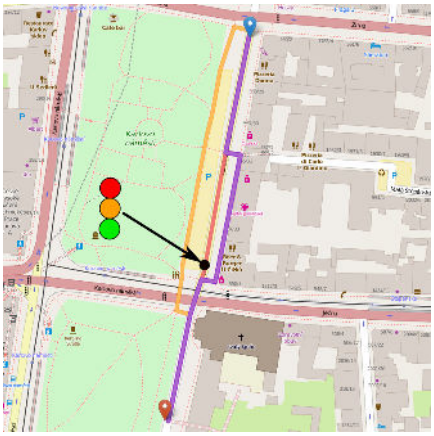


Figure 5.3: Traffic light positioned out of junction

Node tags Tags in OSM are not only assigned to ways, but also to nodes and relations. However, the function f (defined in Section 3.1) only returns features associated to the cycleway segment.

Therefore, we have to ensure that relevant tags saved into the `GeneralNode`, e.g. `node::crossing::traffic_signals`, `node::highway::stop`, `node::barrier::gate` etc., are migrated to the proper `GeneralEdge`.

We distinguish the node's tags to three categories: whether they belong to a motor vehicle infrastructure, a pedestrian or bicycle infrastructure, or to whole cycleway network. It is important to correctly select the category, in order to move the tag to an appropriate set of edges. For example, the tag `node::crossing::traffic_signals` belongs to pedestrian network, but node to which the tag is assigned may be also part of edges of the motor vehicle infrastructure.

Interpolating From a user experience view, a utility for finding a nearest node in the graph for a selected geographic location of an origin and an destination is a critical factor. However, the utility is highly dependent on nodes' density. Especially, some edges might be really long, e.g., maximal length of an edge in the Prague city graph is approximately 700 meters.

To improve this utility, we propose to interpolate edges with nodes separated by a distance greater or equal than $d_{\min}$. Thus, we define a vector of the edge $(u, v) \in E'$ and its normalized form $norm(\overrightarrow{(u,v)})$ as

$$\overrightarrow{(u,v)} = (v^{\text{lon}}, v^{\text{lat}}, v^{\text{elev}}) - (u^{\text{lon}}, u^{\text{lat}}, u^{\text{elev}})$$

$$norm(\overrightarrow{(u,v)}) = \frac{\overrightarrow{(u,v)}}{\|\overrightarrow{(u,v)}\|}$$

then the i-th interpolated node w_i of the edge (u, v) is defined as

$$(w_i^{\text{lon}}, w_i^{\text{lat}}, w_i^{\text{elev}}) = (u^{\text{lon}}, u^{\text{lat}}, u^{\text{elev}}) + (i \cdot d) \cdot norm(\overrightarrow{(u,v)}),$$

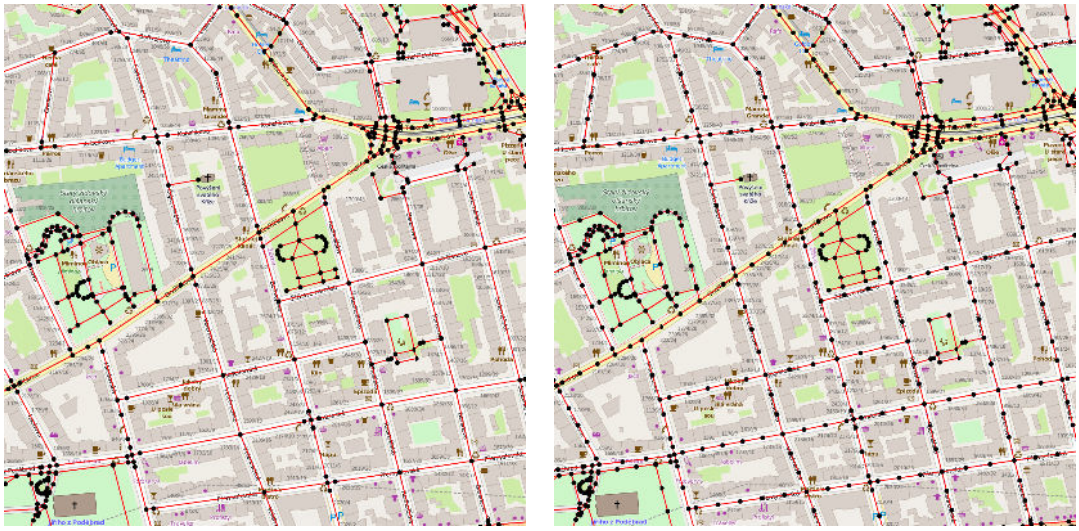
$$\text{where } i = 1, 2 \ldots n, \quad n = \lfloor \frac{l(u,v)}{d_{\min}} \rfloor - 1, \quad \text{and} \quad d = \frac{l(u,v)}{n}$$

For each such a node w_i, we create a `GeneralNode` and instead of the original edge (u, v) we create two `GeneralEdge` objects equivalent to edges (u, w_i) and (w_i, v).

In order to not create unnecessarily short edges, we are going to interpolate only edges longer than threshold $t = 2d_{\min}$. Figure 5.4a shows a graph without interpolated edges and Figure 5.4b shows a graph with interpolated edges.

Simplification If a node u has only one incoming edge (v, u) and only one outgoing edge (u, w) , where $u \neq v \neq w$ and $f((v, u)) = f((u, w))$, i.e., all tags/features are the same, then we can *contract* such a node u and create a new edge (v, w).

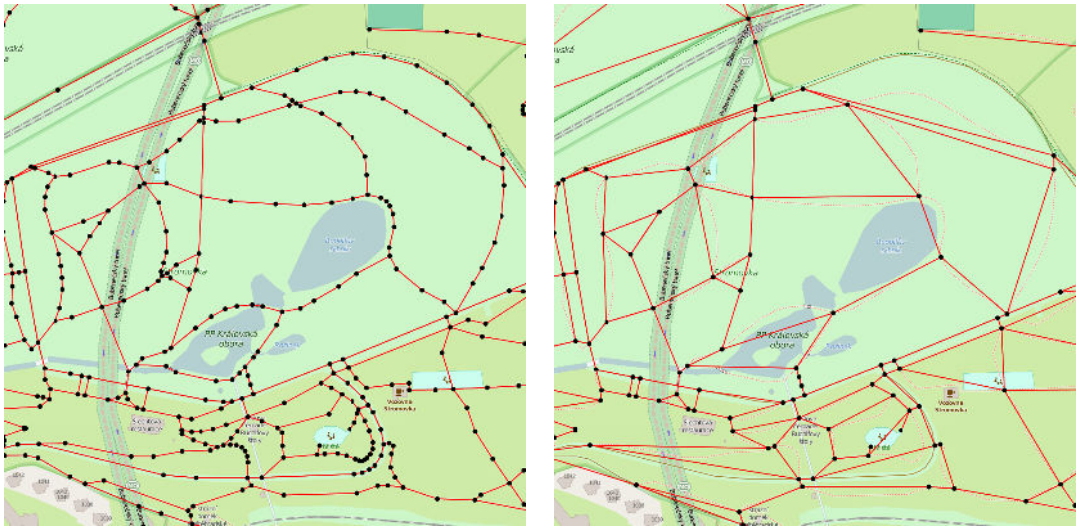
The imported graph from OSM elements contains more than a half of nodes meeting declared property. This is caused by the fact that even cycle network OSM data contain more nodes than are necessary for bicycle planning purposes. On other hand, these nodes are inevitable for rendering a bicycle route to map or computing edge's cost related to elevation. Therefore, we are not removing the nodes completely, we are just contracting them, i.e., we store them inside newly created edge and use them when they are demanded.



(a) without interpolated edges (b) with interpolated edges

Figure 5.4: Example of graphs

By contracting those nodes, we are able to lower the number of nodes and edges to half without any information loss. This has an impact on memory consumption and also on the running time of the shortest path algorithm, because the asymptotic complexity of Dijkstra's algorithm implemented with a priority queue depends on the number of both nodes and edges. Figure 5.5a shows an graph without contracting nodes and Figure 5.5a shows an graph after applying contraction.



(a) before simplification (b) after simplification

Figure 5.5: Example of graphs

Opposite direction During the OSM import phase we create only `GeneralEdge` objects with respect to the order of nodes within the way.

If the edge is one-way, it must contain a tag `way::oneway::yes` declaring that ride is only allowed in current direction or a tag `way::oneway::-1` declaring that ride is only allowed in opposite direction to current (in such case we must reverse the edge).

As long as no tag specifies one-way restriction, we assume that cyclists are allowed to ride on a way in both directions. Also, we create an opposite edge to each `GeneralEdge` if it contains one of the following tags, declaring that on a particular edge is allowed to ride a bicycle in opposite to direction defined by OSM way: `way::cycleway::opposite`, `way::cycleway::opposite_lane`, `way::cycleway::opposite_track`, and `way::bicycle:backward::yes`. The last mentioned tag allows cyclists to travel in both directions on a one-way street.

To allow ride in opposite direction on some calm one-way roads, we create reversed `GeneralEdge` objects for all one-way edges except those with tags `way::highway::primary`, `way::highway::primary_link` or `way::highway::secondary`, `way::highway::secondary_link` and assign them a tag `way::oneway::opposite`. However, the tag `way::oneway::opposite` will be strictly penalized.

Junction Extension To construct the extended graph G' defined in Section 3.2, we do the following actions for each junction node $v \in V_{junction}$.

Each incoming edge (u, v) of the junction node v is redirected to a new entrance node v^{in} whose coordinates are created in the following way.

$$v^{\text{in}}.lon = v.lon + \frac{(u.lon - v.lon) \cdot r \cdot cos(\theta)}{l(u,v)} + \frac{(u.lat - v.lat) \cdot r \cdot sin(\theta)}{l(u,v)}$$

$$v^{\text{in}}.lat = v.lat + \frac{(u.lat - v.lat) \cdot r \cdot cos(\theta)}{l(u,v)} - \frac{(u.lon - v.lon) \cdot r \cdot sin(\theta)}{l(u,v)}$$

Also each outgoing edge (v, w) of the junction node v is redirected from a new exit node v^{out} whose coordinates are created in following way.

$$v^{\text{out}}.lon = v.lon + \frac{(u.lon - v.lon) \cdot r \cdot cos(\theta)}{l(u,v)} - \frac{(u.lat - v.lat) \cdot r \cdot sin(\theta)}{l(u,v)}$$

$$v^{\text{out}}.lat = v.lat + \frac{(u.lat - v.lat) \cdot r \cdot cos(\theta)}{l(u,v)} + \frac{(u.lon - v.lon) \cdot r \cdot sin(\theta)}{l(u,v)}$$

After successful creation of all entrance nodes V_{in} and exit nodes V_{out} for a particular junction, we build junction inner edges from all nodes $v^{\text{in}} \in V_{\text{in}}$ to all nodes $v^{\text{out}} \in V_{\text{out}}$. To apply turn restriction defined by OSM relations, we can delete inner edges which represent restricted turn. The difference between the standard graph and the extended graph is shown in Figure 5.6.

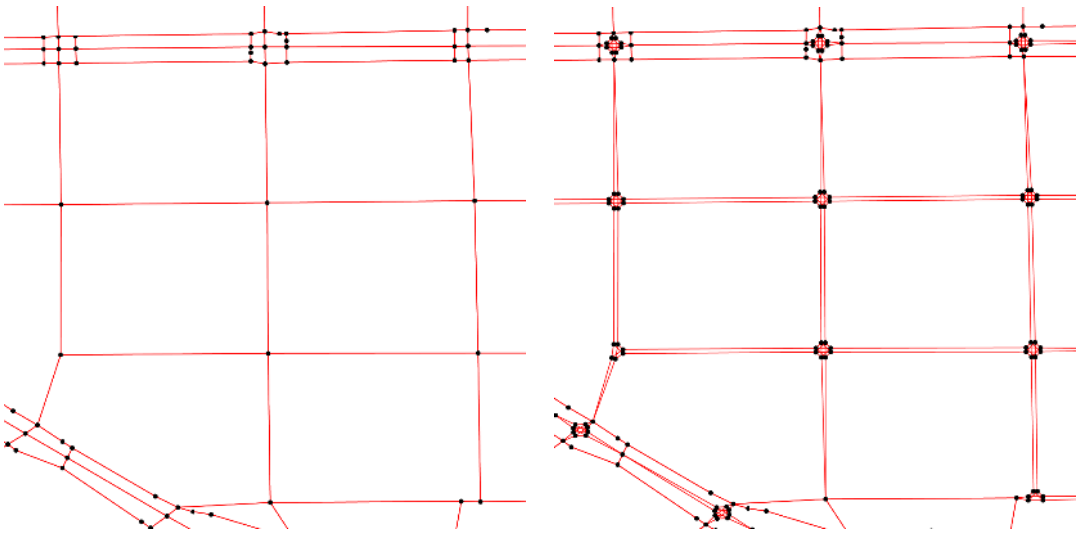
Furthermore, to be able to classify junctions, we define tags taking into account four aspects influencing a ride through a junction. First aspect is a presence of traffic signals. Even though, traffic lights cause delay and cyclists often tend to avoid them, it was shown in [6]

that if they are forced to cross an intersection with a high traffic volume, they will rather use one with a traffic control device. Consequently, we must consider possible actions to take on a junction: right turn, keep straight, left turn, and u-turn. Another important aspect is an amount of vehicles crossing a junction per day. We divide the traffic volume per day to four categories: 1, 5, 10, and 20, where the number is approximated amount of cars times one thousand per day. The last aspect is to distinguish a direction to a street with the same or lower traffic volume (**parallel**), or to a street with the higher traffic volume (**cross**). Following example shows two from 64 possible tags for junction inner edges:

- `way::turn::left_20parallel` describes left turn from high traffic street on junction without traffic lights and with traffic volume higher than 20k vehicles per day
- `way::turn::traffic_lights_straight_5cross` describes going straight through an controlled junction with traffic volume approximately 5k vehicles per day and where crossing street has higher traffic volume than incoming street to an intersection.

For the sake of clarity, we say that an cycleway segment belongs to a motor vehicle infrastructure (mentioned in the Section 5.2.1) if it contains at least one of the following tags

```
way::highway::living_street, way::highway::primary,
way::highway::primary_link, way::highway::residential,
way::highway::secondary, way::highway::secondary_link,
way::highway::tertiary, way::highway::tertiary_link,
way::highway::service, way::highway::unclassified, way::highway::road
```



(a) without junction extension (b) with junction extension

Figure 5.6

5.2.2 Bicycle Planning Graph

After successfully applying all techniques introduced above, we construct the bicycle planning graph from the general graph. The bicycle planning graph consists from `CycleNode` and `CycleEdge` objects. In contrast to the `GeneralEdge`, the `CycleEdge` contains instead of OSM tags just the cost vector of cost values for individual criteria. Next two paragraphs describe two issues we need to solve in order to construct the planning graph. First, we describe how we use OSM tags to compute criteria values. Second, we have to ensure that the final graph is a strongly connected component.

Map OSM Tags to Criteria In order to pre-compute criteria values and store them into `CycleEdge` objects, we have to map OSM tags, i.e., map features, with defined cost functions in Section 3.4. To remind, not all proposed criteria taking into account additional features describing e.g., surface or dedication of road. The distance and the elevation gain criteria considers only geographical factors as length and elevation. In following paragraphs, we describe how we integrate map features obtained from OSM into the travel time, the comfort, the quietness, and the bike friendliness criteria.

The cost function for the travel time criterion defines two functions r_{tt} and q both expressing factors which slows down cyclists. The value of functions r_{tt} and q depend on all features associated with an edge, therefore, we define additive functions depending only on one feature. All tags that affect cyclists' speed (e.g., road segments with a bad surface or segments where cyclists must dismount a bike) together with its penalization value for an additive function $r'_{\text{tt}} : F \rightarrow \mathbb{R}^+$ are shown in Appendix A in Table A.1. Table A.2 shows an additive function $q' : F \rightarrow \mathbb{R}^+$ which assigns a delay in seconds to each tag considered as a static slowdown factor (e.g., steps, crossing, traffic signals, etc.). Therefore, the total effect of all features of an edge $(u, v) \in E'$ given by function $f((u, v))$ for functions r_{tt} and q is defined as:

$$r_{\text{tt}}((u, v)) = \min\{r'_{\text{tt}}(x) \mid x \in f((u, v))\}$$
$$q((u, v)) = \max\{q'(x) \mid x \in f((u, v))\}$$

The comfort criterion and the bike friendliness criterion depend on two functions. First, the function r_{co} penalizes edges with features disturbing comfortability (e.g., surface covered by cobblestones). Second, the function p returns what additional distance are cyclists willing to travel to avoid a busy intersection. Similarly, as in previous case, both functions depends on all features returned by the function $f((u, v))$. Therefore Table A.3 shows map features associated with an additive function $r'_{\text{co}} : F \rightarrow \mathbb{R}^+$ for r_{co} together with values it obtains. Also, we introduce a function $p' : F \rightarrow \mathbb{R}^+$ which is an additive function for p and the Table A.5 shows features related to p' together with values. Then, the functions r_{co} and p are defined as:

$$r_{\text{co}}((u, v)) = \max\{r'_{\text{co}}(x) \mid x \in f((u, v))\}$$
$$p((u, v)) = \max\{p'(x) \mid x \in f((u, v))\}$$

The values for the function p' were derived from Table 5.1, which represents in percent how much are cyclists willing to travel farther on one mile commute trip to avoid the junction with certain properties. The values was derived by [6], where authors study cyclists' behaviour taken from GPS tracks.

	S	ST	L	LT	R	RT	U	UT
5P	0	2.1	6	2.1	0	0	6	5
10P	0	2.1	9.1	2.1	0	0	9.1	5
20P	0	2.1	23.1	2.1	0	0	23.1	5
1C	1	1	2	1	0	0	2	5
5C	4.1	2.1	4.1	2.1	2	0	4.1	5
10C	5.9	2.1	5.9	2.1	3.8	0	5.9	5
20C	32.2	2.1	32.2	2.1	3.8	0	32.2	5

Table 5.1: Percentage of 1-mile that are cyclists willing to travel in order to avoid junction with defined parameters, source [6] (1, 5, 10, 20 – times 1000 cars per day; P – parallel; C – cross; T – traffic signals; R – right turn; S – straight; L – left turn; U – u-turn)

The quietness and the bike friendliness criteria defines the function r_{qu} which prefers an infrastructure dedicated for cyclists, e.g., cycleways and cycle lanes. Consequently, we define an additive function r'_{qu} and all values it can obtain are shown in Table A.3. Suppose the function $f((u, v))$ returns all features associated with an edge $(u, v) \in E$, then the function r_{qu} is given by:

$$r_{\text{qu}}((u, v)) = \frac{\sum^{\forall x \in f((u,v))} r'_{\text{qu}}(x)}{|f((u, v))|}$$

Strongly connected components The constructed bicycle planing graph does not have to be strongly connected, i.e., each node u is reachable from every other node v. Instead, the graph consists of multiple strongly connected components. We find and use only the largest strongly connected component in the constructed bicycle planing graph to ensure that there will exist a bicycle route between all pairs of nodes in the graph. To identify the largest strongly connected component we use code from a library developed at Agent Technology Center (ATG) that is based on Kosaraju's algorithm.

5.3 Application

This section provides an insight to implementation of the cycle planner application. We introduce the utility for searching nearest nodes in the planning graph and provide important parts on the implementation of the planning algorithms. In the end we describe the RESTful API and usage of the cycle planner in applications used by people in the Czech Republic.

The cycle planner application is implemented in Java 7. It consists of five Maven[7] modules which are described below and Figure 5.7 shows dependencies between these modules.

- **cycle-planner** – contains all algorithms and API; its main responsibility is to plan the bicycle route
- **cycle-planner-core** – consists of base data structures common for the whole application, e.g., `CycleNode`, `CycleEdge`

[7] https://maven.apache.org/

- **cycle-city-data-builder** – all optimisation methods on the general graph are implemented here; it builds the cycleway graph consisting of `CycleNode` and `CycleEdge` objects
- **cycle-city-data-storage** – contains cycle city data built by the cycle-city-data-builder module
- **osm-import** – imports OSM data to Java application and constructs the general graphs structure

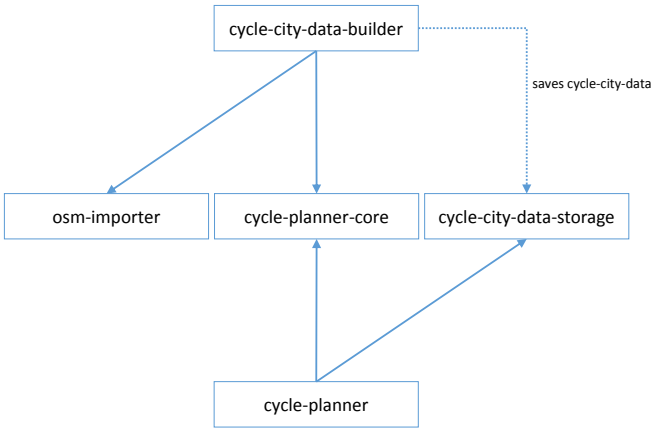


Figure 5.7: Cycle planner application modules

5.3.1 Nearest node

To find the nearest node among both visible graph nodes and contracted nodes (nodes disabled during the simplification process of the general graph) we employ Java implementation of KD-tree with two dimensions: longitude and latitude. The utility returns five nearest nodes at maximum, but all of them must have distance equal or lower to the distance from an origin to the nearest node plus some ϵ distance.

If some of the nearest nodes is the contracted node, we create a new `CycleNode` and two new `CycleEdge` objects referred as a *first mile* if the node is from the set of the nearest nodes to an origin, or a *last mile* if the node is from the set of the nearest nodes to a destination.

5.3.2 Algorithms

Scalarised Multi-Criteria Solution Method We modify the implementation of the Dijkstra's algorithm in a library provided by ATG to support multiple origins and multiple destinations. Such a problem occurs when multiple nearest `CycleNode` objects in the graph G' are found to a location specified by the user. Modification consists of inserting all origins into a priority queue during the initialisation and of changing a terminal condition to stop the search when bicycle routes to all destinations are found. In the end, from several options we select the bicycle route with the minimal cost value.

The implementation of Dijkstra's algorithm uses a binomial heap representation of the priority queue and returns a complex object `GraphPath` from a common Java graph library JGraphT. The `GraphPath` wraps all objects in order to reconstruct the bicycle route and also provides information about the route's cost value, and number of expanded nodes by the algorithm.

Multi-Criteria Solution Method The implementation of the multi-criteria Dijkstra's algorithm (MCD) uses the standard Java implementation of the priority queue and returns object called `ParetoSet` containing collection of lists of `CycleEdge` or `CycleNode` objects, i.e., Java representation of Pareto set where all non-dominated routes between an origin and a destination are represented either as sequence of nodes or sequence of edges.

5.3.3 RESTful API

A client application is responsible for sending the user's request and showing received bicycle routes in an appropriate way. A server application is responsible for the planning of bicycle routes. To separate the client from the server application, we design the cycle planner in **Representational State Transfer** (**REST**) software architecture style. The server supports gzip encoding in order to decrease a size of the transferred data between the client and the backend. That is important mainly for mobile applications where the size of the received data matters a lot.

All planned bicycle routes and feedback received from clients are stored in NoSQL database to enable data persistence and ability to provide statistics. We use NoSQL document type database MongoDB, because it supports saving and loading files in JSON format. Therefore, without any transformations we can store and retrieve bicycle routes and feedback in JSON format. The database behaves as a cache for the planned routes, therefore, the cycle planner is not overwhelmed when the user wants to see earlier bicycle routes. Previous cycle planner implementation did not support it and web client application sends the same request multiple times.

The server communicates with clients over Hypertext Transfer Protocol (HTTP). The following API base URI together with supported HTTP method (GET or POST) are used by the server:

`POST api/v2/journeys` It consumes request in JSON format shown in Figure 5.8 and starts planning a bicycle route between an origin, waypoints if defined, and a destination. By default, if no profile is specified, it plans a bicycle route for four profiles defined in Section 4.1.1. If the planner successfully finds bicycle routes it sends HTTP 201 response with *journeyId* under which the routes can be retrieved from the API. In case of failure, it returns HTTP 400 when request JSON format is invalid and HTTP 404 when application was unable to store the routes in the database.

`GET api/v2/journeys/test` Returns HTTP 200 with routes in JSON format as shown in Figure 5.9 between arbitrary points in Prague city. This URI serves to ensure that server is answering and is functional.

`GET api/v2/journeys/mc` The URI expects five parameters: latitude and longitude for an origin and a destination and the name of the multi-criteria algorithm to use. It returns

a JSON list of edges where each edge is associated with a width and a colour representing the number of occurrences in the Pareto set.

`GET api/v2/journeys/journeyId` Returns a bicycle routes stored under the specified journeyId in JSON format specified in Figure 5.9.

`POST api/v2/feedback` Accepts only feedback in JSON format shown in Figure 5.10 and returns HTTP 201 if it was successfully stored.

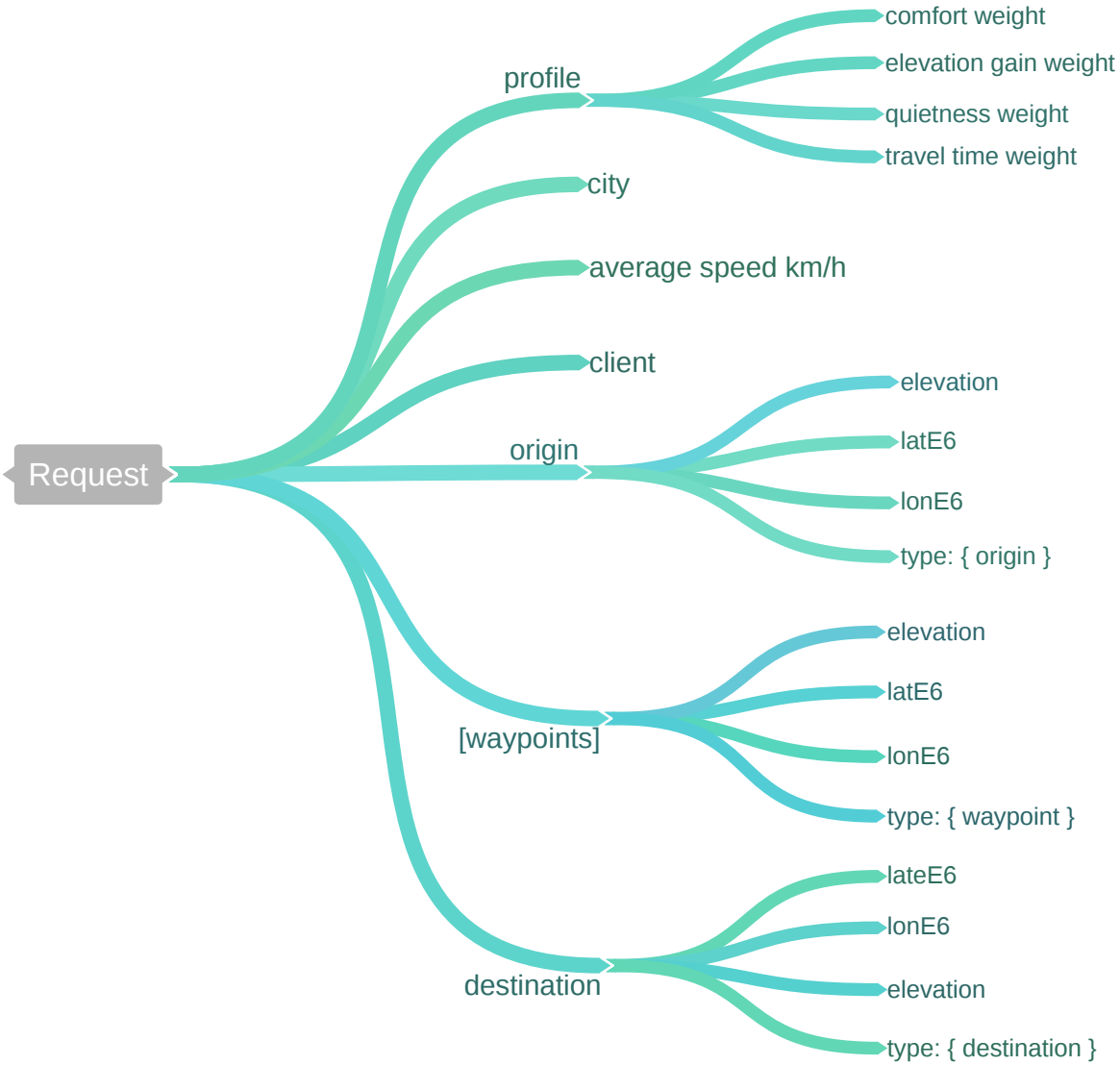


Figure 5.8: JSON schema of the API request

5.4 Real Deployment

Cyclists are able to plan a bicycle route with our cycle planner through the web page <`http://cykloplanovac.cz`>, which provides four alternative paths and displays elevation profile

of routes. A screenshot of the web application is shown in Figure 5.11.

Other usage of our cycle planner application is the Android application named "Cyklo-plánovač" developed as the bachelor's project by Jan Linka [40]. As shown in Figure 5.12, the Android application provides turn-by-turn navigation. Also, the Android application and cycle planner are part of the competition "Do práce na kole". Since the start of the competition on May 4, 2015, the cycle planner serves 200 requests per day on average and the Android application has over 700 downloads. Currently, the application is able to plan bicycle routes in 25 cities in the Czech Republic.

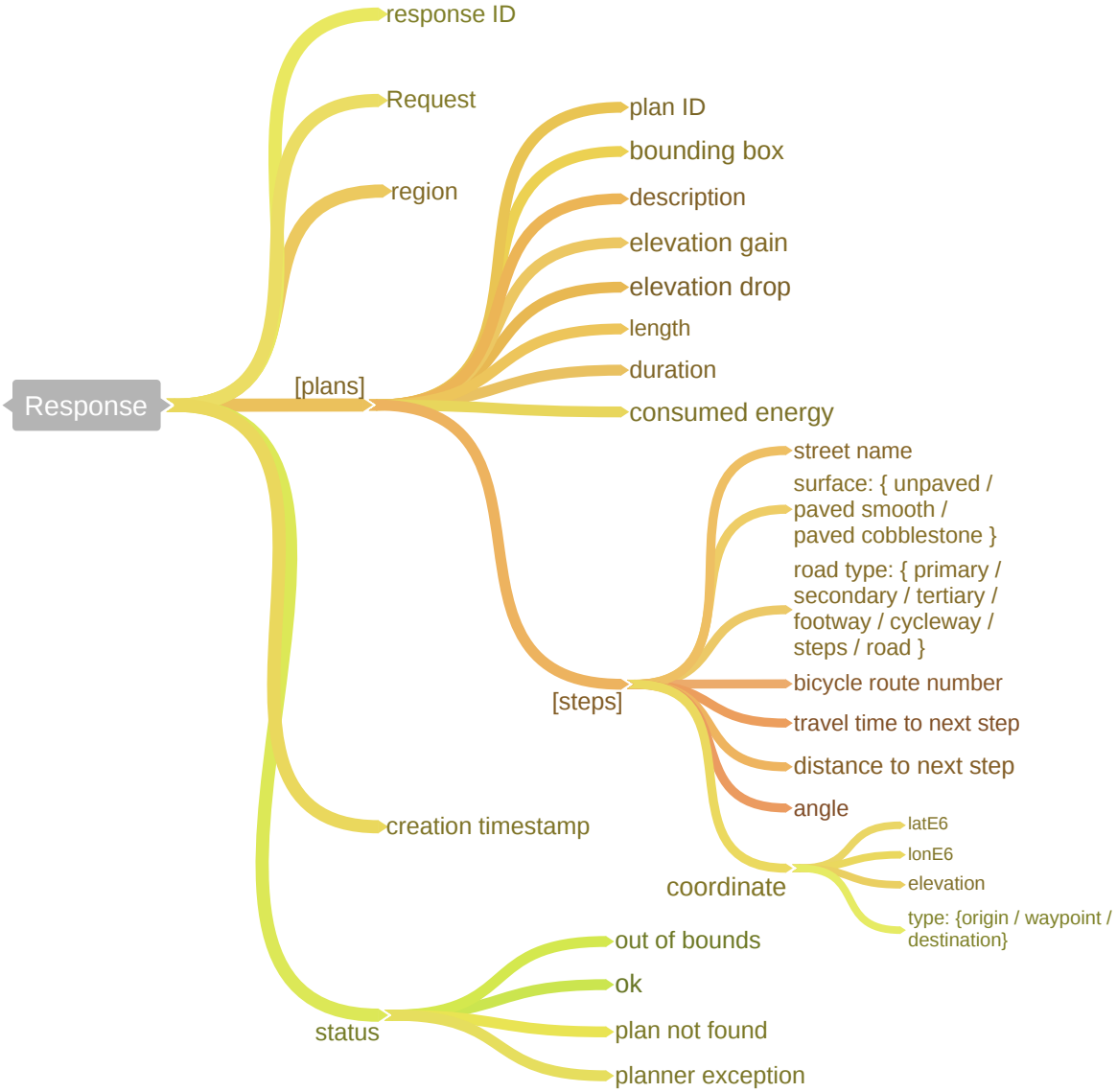


Figure 5.9: JSON schema of the API response

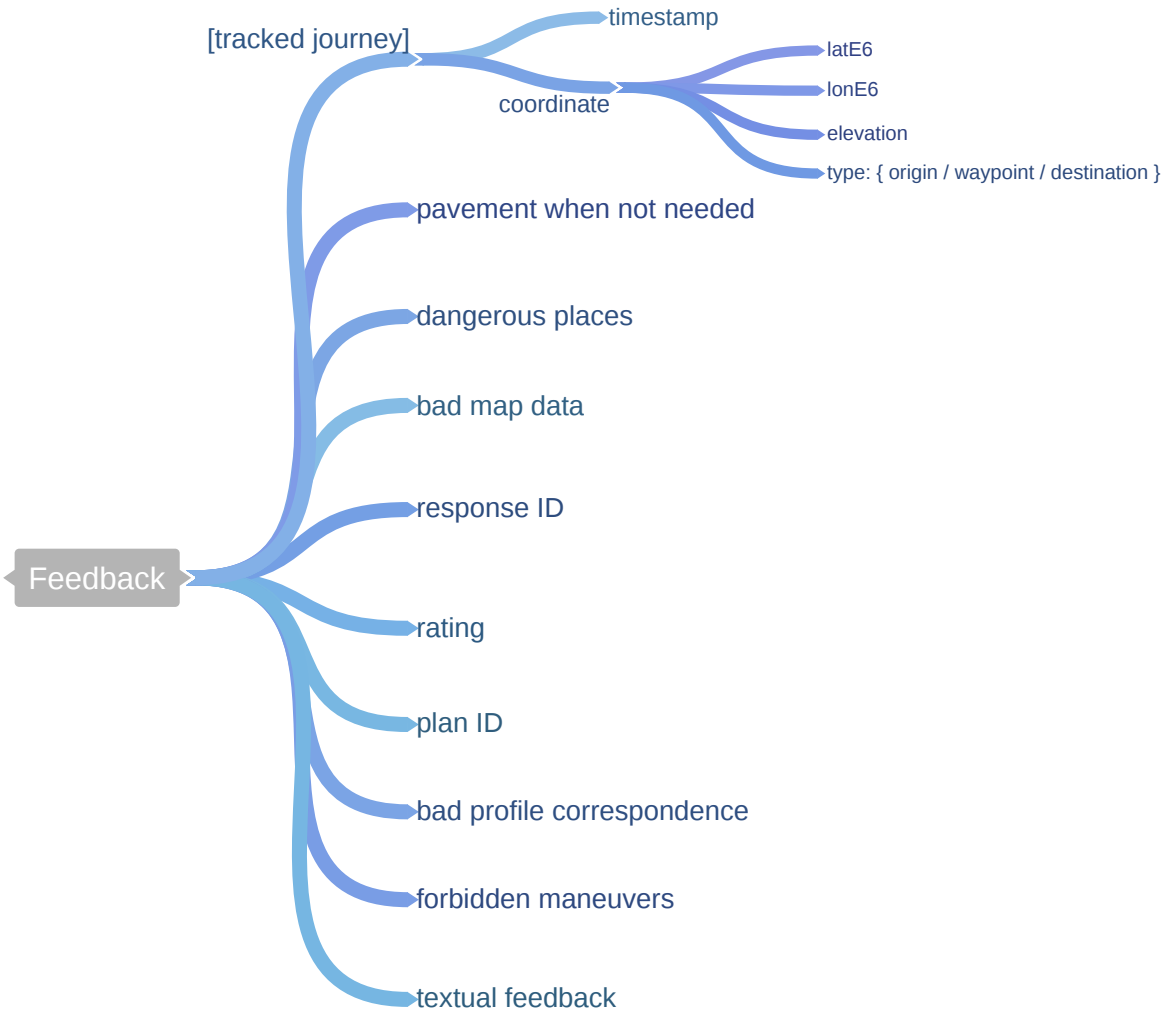


Figure 5.10: JSON schema of the API feedback to API request

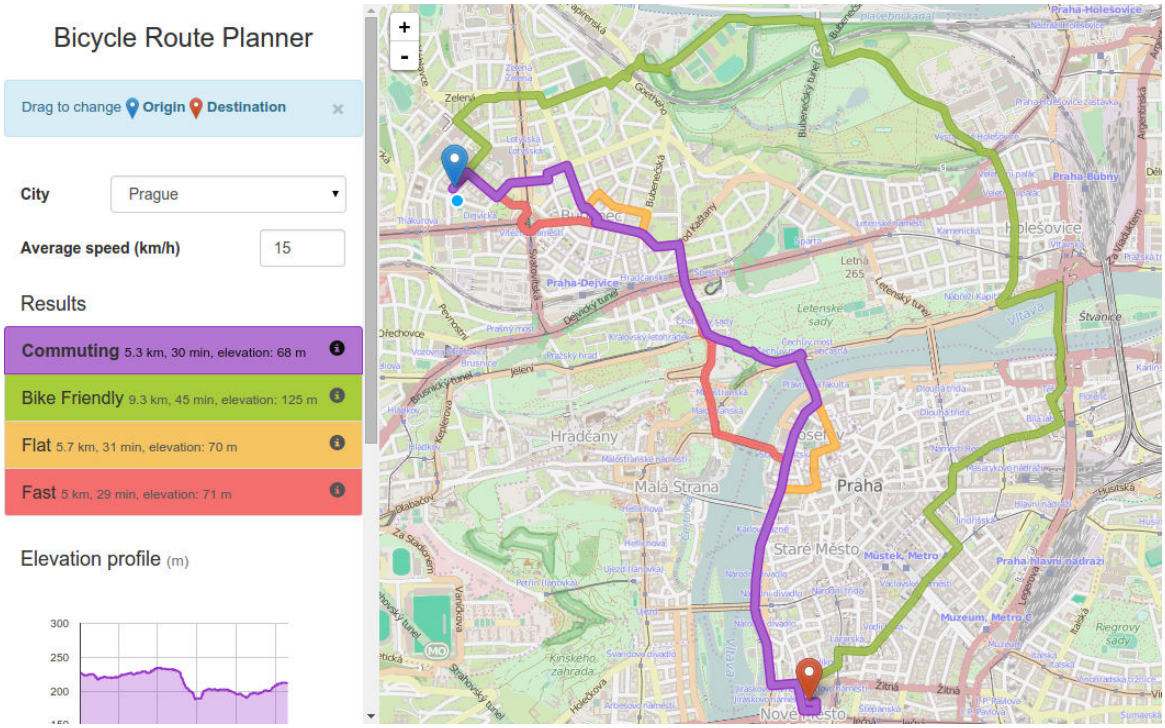


Figure 5.11: Web frontend of the cycle planner – cykloplanovac.cz

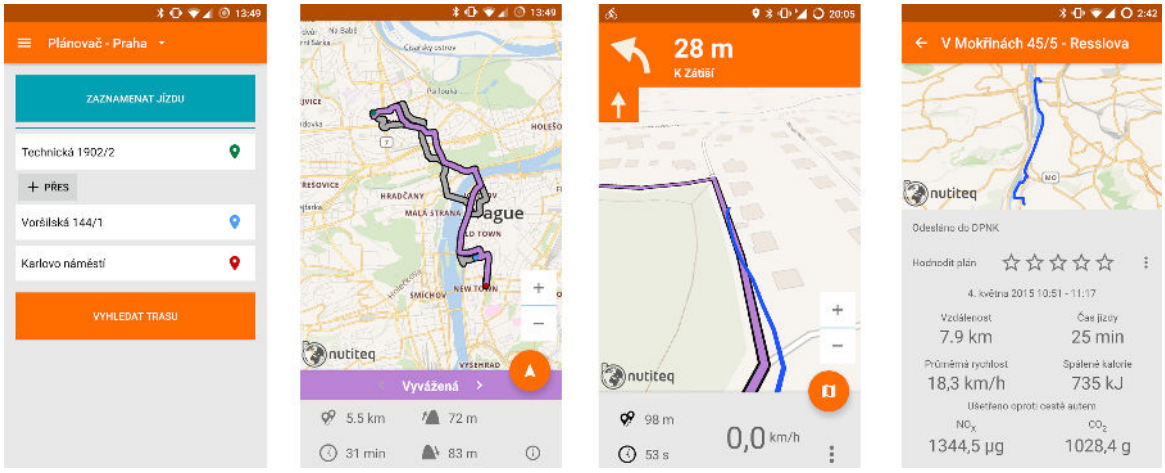


Figure 5.12: The Android application – "Cykloplánovač"

Chapter 6

Evaluation

To evaluate our solution methods presented in Chapter 4, we consider the real cycleway network of the city of Prague constructed as described in Section 5.2. Prague is a challenging experiment location due to its complex geography and fragmented cycling infrastructure, which raises the importance of proper multi-criteria routing. Also, Prague has a large community of cyclists who massively contributes to OSM project, therefore the map data are considered to be precise.

This chapter evaluates the proposed pruning heuristics for the multi-criteria Dijkstra's algorithm (MCD) and the junction extension.

6.1 Multi-Criteria Method

This section starts with a definition of the evaluation metrics which are used to compare the quality of various implemented pruning heuristics. Then, we present how parameters for the pruning heuristics and whole evaluation process are set up.

The evaluation of the multi-criteria solution method is split into two phases. First, we evaluate all methods on three medium-sized regions of Prague in order to compare a quality of solution routes (not just a running time). Due to the reasonable size of the graphs, even the pure multi-criteria Dijkstra algorithm is able to solve a query in a short enough time. Second, we take the methods with a potential to scale well and evaluate them on the whole Prague graph.

We also evaluate combinations of the pruning heuristics. For instance, the ellipse and the ϵ-dominance or the bounded search and the ϵ-dominance, etc., are easily combined because they speedup the search in different parts of the MCD algorithm. Also, the bounded search plus the ellipse heuristic can be combined naturally, the bounded search bounds the search in the cost space and the ellipse in the physical space.

6.1.1 Evaluation Metrics

We consider two categories of evaluation metrics: *speed* and *quality*. We use the following metrics to measure the algorithm speed:

- Average speedup over the standard, optimal multi-criteria Dijkstra's algorithm in terms of algorithm runtime.
- Average runtime t in ms for each origin-destination pair together with its standard deviation σ_{t}.

Some of the proposed pruning heuristics are unable to find the full Pareto set of bicycle routes Π^* (defined in Section 3.3). Therefore, we have to define metrics to measure quality of returned routes with respect to full Pareto set:

- Average distance $d_c(\Pi^*, \Pi)$ of the heuristic Pareto set Π from the optimal Pareto set Π^* in the cost space. Distance $d_c(\pi^*, \pi)$ between two routes π^* and π is measured as the Euclidean distance in the unit three-dimensional space of criteria values normalized to the $[0, 1]$ range.

$$d_c(\Pi^*, \Pi) := \frac{1}{|\Pi^*|} \sum_{\pi^* \in \Pi^*} \min_{\pi \in \Pi} d_c(\pi^*, \pi)$$

- Average distance $d_J(\Pi^*, \Pi)$ of the heuristic Pareto set Π from the optimal Pareto set Π^* in the physical space. Jaccard distance $d_J(\pi^*, \pi)$ [41] is used to measure the dissimilarity between Pareto routes. For routes π^* and π, i.e., sequences of edges, the physical distance is computed by dividing the difference of the sizes of the union and the intersection of the two route sets by the size of their union.

$$d_J(\pi^*, \pi) := \frac{|\ (\pi^* \cup \pi)\ | - |\ (\pi^* \cap \pi)\ |}{|\ (\pi^* \cup \pi)\ |}$$

$$d_J(\Pi^*, \Pi) := \frac{1}{|\Pi^*|} \sum_{\pi^* \in \Pi^*} \min_{\pi \in \Pi} d_J(\pi^*, \pi)$$

- Average number of routes $|\Pi|$ in the Pareto set Π together with its standard deviation $\sigma_{|\Pi|}$.
- The percentage of Pareto routes $\Pi_{\%}$ in heuristic Pareto set Π that are equal to routes in the optimal Pareto set Π^*.

6.1.2 Evaluation Settings

To evaluate the proposed pruning heuristics within the multi-criteria solution method we consider four graphs. We use three different *medium-sized* cycleway graphs corresponding to three distinct areas of the city of Prague. We have chosen parts Prague A, Prague B, and Prague C to be different in terms of network density, nature of the cycling network, and terrain topology so as to evaluate the performance of heuristics across a range of conditions. The size of the medium size evaluation graphs allows us to run the multi-criteria Dijkstra's algorithm without any speedups, which is crucial for comparing the quality of heuristic and optimal solutions. The fourth graph belongs to the whole Prague city, which we use to evaluate the scalability of the best pruning heuristics. The sizes of the evaluation graphs together with the specifics of the areas are described in Table 6.1.

Table 6.1: Graph sizes for the experiments.

Graph	Nodes	Edges	Area
Prague A	9709	22928	This graph covers a flat city centre area of the Old Town with many narrow cobblestone streets and Vinohrady with the grid layout of streets
Prague B	6733	5218	This graph covers a very hilly area of Strahov and Brevnov with many parks.
Prague C	9834	22798	This graph covers residential areas of Liben and Vysocany further from the city centre. There are many good cyclepaths in this area.
Whole Prague	141117	323114	The whole city of Prague.

In order to solve multi-criteria problem we consider three criteria: the distance, the bike friendliness and the elevation gain. These criteria do not depends on each other, e.g., the comfort and the quietness use the travel time value, therefore they are the best candidates to be used in the multi-criteria approach.

Three of four proposed pruning heuristics needs to set up their parameters to run efficiently. To select the correct values for the specific parameters we consider two factors: the running time and the average cost space distance d_c between the pruning methods' Pareto set and the multi-criteria Dijkstra's Pareto set (see more detailed description in Section 6.1.1 above). All experiments for considering various pruning techniques' parameter values were run on all medium-sized graphs.

The ellipse pruning heuristic expects at the input the constant ratio $\frac{a}{b}$ between a semi-major axis a and a semi-minor axis b. Figure 6.1 shows results for three values of parameter $\frac{a}{b}$. We select the value $\frac{a}{b} = 1.5$ due to small derivation from the optimal Pareto set and huge speed up over the value $\frac{a}{b} = 1.25$. The difference between values $\frac{a}{b} = 1.5$ and $\frac{a}{b} = 1.75$ is insignificant and we prefer one with better quality.

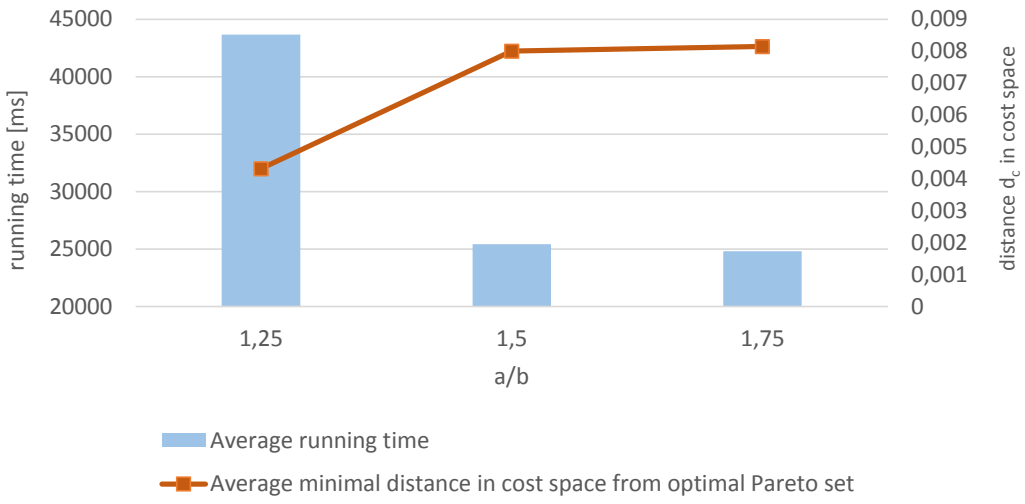


Figure 6.1: Ellipse $\frac{a}{b}$ parameter settings

The ϵ-dominance pruning heuristic uses the ϵ value to approximate the optimal Pareto set. We choose three values to assign to ϵ: 0.01, 0.025, and 0.05. Results in Figure 6.2 shows that the distance from optimal Pareto set linearly grows and the running time strictly drops, with larger values for ϵ parameter. For the future evaluation of ϵ-dominance pruning heuristic, we consider the value $\epsilon = 0.025$ for its still acceptable value for distance metric and feasible running time.

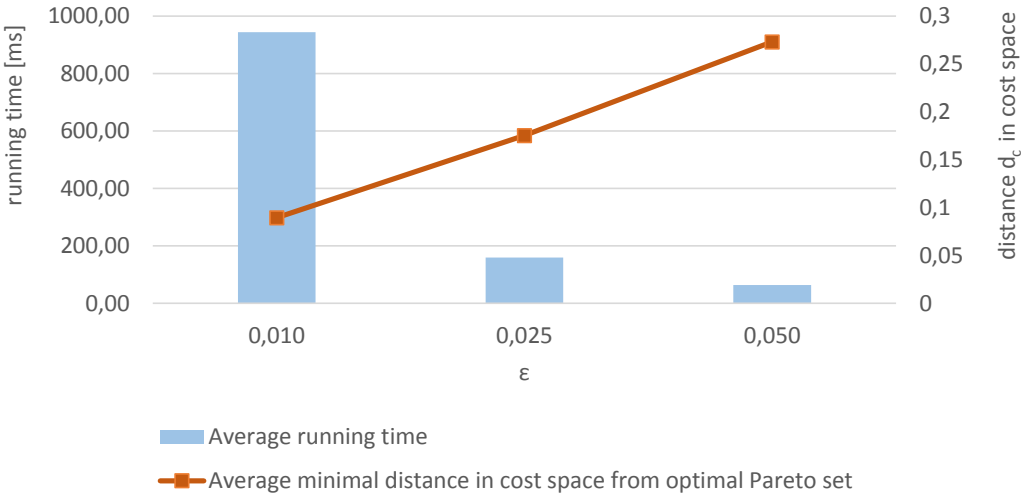


Figure 6.2: ϵ-dominance parameter settings

At last, in the buckets pruning heuristic we have to define size of the individual buckets. We have proposed the 3-criteria bicycle routing problem (distance, bike friendliness, elevation gain), therefore 3 bucket's sizes are needed. For the buckets we consider the following sizes: the travel time ($\{32, 80, 160\}$, the bike friendliness $\{448, 1120, 2240\}$, and the elevation gain $\{2, 5, 10\}$). The sizes have been derived from the observation of the optimal Pareto set criteria values. On average, the ratio between the criteria values in Pareto set was $16 : 224 : 1$. The running time and the distance from optimal Pareto set for all possible combination of bucket sizes are plotted in Figure 6.3. For the future evaluation of the buckets pruning heuristic we consider buckets sizes $(80, 1120, 2)$, due to relatively good compromise between both considered metrics.

Each evaluation process was run on a single core of a 2.60GHz Intel Xeon E5-2650v2 processor of server machine running Debian operating system. For each evaluation on the medium-sized graphs, it was allocated 5GB of RAM and for each evaluation on the whole Prague graph, it was allocated 10 GB of RAM.

For each graph evaluation area, a set of origin-destination pairs generated randomly with a uniform spatial distribution was used. We generated 105 origin-destination pairs for each graph. The minimum origin-destination distance is set to 500 m and the maximum origin-destination distance for medium-sized graphs is 4.5 km and for the whole Prague graph is 12 km. After successful evaluation, we discard first 5 results, which we consider as a warm up period for Java virtual machine.

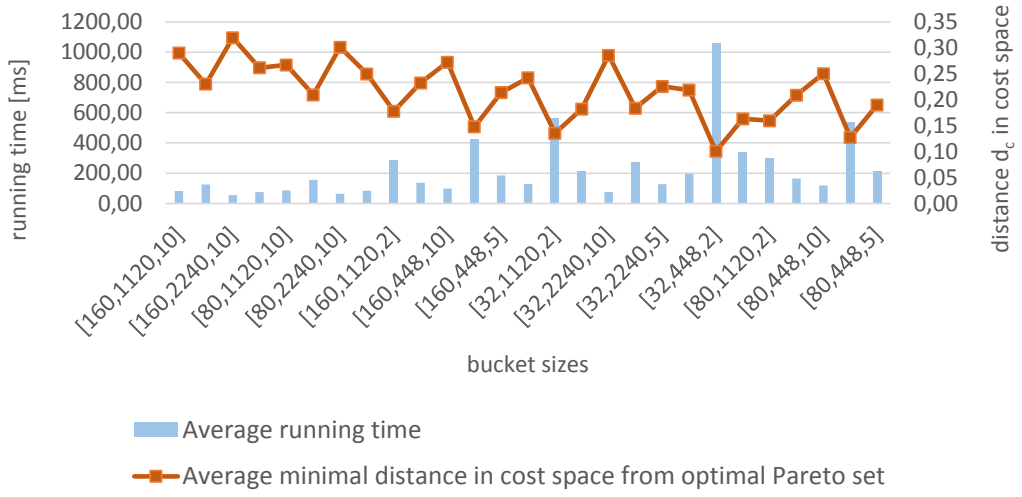


Figure 6.3: Buckets parameter settings

The parameters in the cost functions were set as follows. The average cruising speed is $s = 14$ km/h, the penalty coefficient for uphill is $a_l = 13$ (according to the route choice model developed in the user study [6]), the maximum downhill speed multiplier is $s_{\text{dmax}} = 2.5$, and the critical grade value is $d'_c = 0.1$.

6.1.3 Results on Prague A, B, C Subgraphs

Table 6.2 summarises the evaluation of the multi-criteria Dijkstra's algorithm and the proposed pruning heuristics using the graphs Prague A, B, and C. Columns d_c, d_J and $\Pi_{\%}$ are calculated with respect to the optimal Pareto set Π^* returned by the MCD algorithm which is used as a baseline for the evaluation of the proposed pruning heuristics and their combinations.

Pruning Heuristic	speedup	**t [ms]**	σ_t	$\lvert\Pi\rvert$	$\sigma_{\lvert\Pi\rvert}$	$\mathbf{d_c}$	d_J	$\Pi_{\%}$
MCD	0	153 494	200 452	326	582	-	-	100.00
MCD_Bounded	4	35 695	87 389	326	582	0	0	100.00
MCD_Ellipse	7	20 731	70 158	314	565	0.008	0.012	99.77
MCD_Bounded_Ellipse	10	15 882	64 713	314	565	0.008	0.012	99.77
MCD_Buckets	369	415	347	14	15	0.160	0.301	58.62
MCD_Bounded_Buckets	815	188	196	14	15	0.165	0.304	58.90
MCD_Ellipse_Buckets	2 108	73	161	14	14	0.168	0.310	58.88
MCD_Bounded_Ellipse_Buckets	1 801	85	140	13	14	0.171	0.312	59.14
MCD_Epsilon	1 023	150	74	9	7	0.175	0.334	64.73
MCD_Bounded_Epsilon	1 073	143	106	9	7	0.180	0.337	65.12
MCD_Ellipse_Epsilon	3 955	39	42	9	7	0.184	0.341	64.90
MCD_Bounded_Ellipse_Epsilon	2 239	69	53	9	7	0.187	0.343	65.25

Table 6.2: Evaluation of the pruning heuristics performance on the Prague A, B, C graphs.

The MCD algorithm returned the optimal Pareto set with an average size of 326 routes at the cost of a prohibitively high running time. In contrast to MCD, the *MCD_Bounded* pruning heuristic computes the same amount of routes four time faster without the loss of optimality.

The *MCD_Bounded*, the *MCD_Ellipse*, and the *MCD_Bounded_Ellipse* perform best in the quality of the solutions. The running time is between 15-35 seconds on average which is a rapid speed up compare to 135 seconds of the MCD algorithm. Note that the *MCD_Ellipse* and the *MCD_Bounded_Ellipse* do not return the full optimal Pareto set, but the quality loss is negligible (99.77% of the routes in the heuristic Pareto set Π are equal to the ones in the optimal Pareto set Π^*).

Figure 6.4 shows heuristics in increasing order by the average distance of the Pareto sets in the cost space. We observe that combinations with *MCD_Ellipse* has better running time than *MCD_Bounded_Ellipse*, this could be caused either by the cost for precomputing single criteria solutions by the Dijkstra's algorithm or by additional dominance relation checks against already found routes to destination. The line showing the values of the average distance of Pareto sets in the cost space contains sharp increase between *MCD_Bounded_Ellipse* and *MCD_Buckets*. This increase is a proof of that the ellipse pruning heuristics prunes only the real outliers from the Pareto set unlike the buckets or the ϵ-dominance.

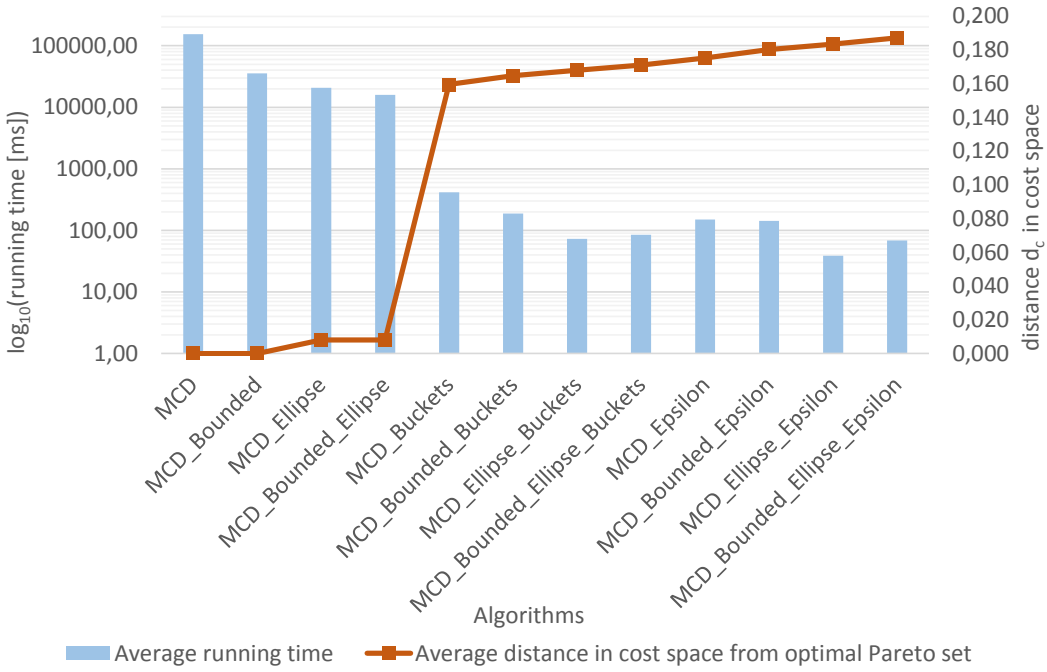
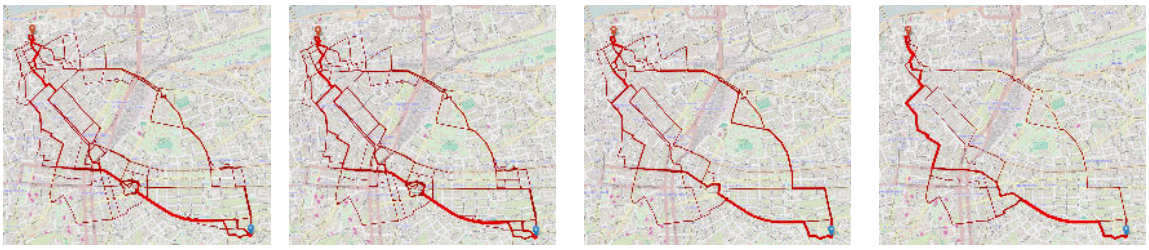


Figure 6.4: Running time and the average distance of the Pareto sets in the cost space metric d_c of the pruning heuristics on the Prague A, B, C graphs.

Figure 6.5 shows changes in distribution of cycleway segments in a Pareto set of a three

pruning methods from the Pareto set of MCD algorithm. The wider and lighter the cycleway segment is, the more times it appeared in routes in the Pareto set. The ellipse pruning techniques does not change the Pareto set, but the buckets and the ϵ-dominance seem to lower the number of the solutions. In a dense area were MCD provides a lot of alternatives, the buckets and the ϵ-dominance provide only a few.



(a) MCD (b) MCD_Ellipse (c) MCD_Buckets (d) MCD_Epsilon

Figure 6.5: Distribution of cycleway segments in the physical space within a Pareto set of particular algorithm

6.1.4 Results on the Whole Prague Graph

In this section, the pruning heuristics with the best performance on the medium size graphs are evaluated on the whole Prague cycleway graph. The results are shown in Table 6.3. We set the maximum running time for one query to 15 minutes. Therefore a column "In 15 min" shows how many queries were finished in 15 minutes.

We observe that the buckets pruning heuristic with the particular setting of the buckets' sizes did not scale well to the whole Prague graph. Only the *MCD_Bounded_Ellipse_Buckets* heuristic was able to finish all queries before the time limit and had the running time 46 seconds on average which is 3 times slower than *MCD_Epsilon*.

All combination with ϵ-dominance pruning heuristics yield to a reasonable running time (below 15 seconds) and all have the same average size of the Pareto set which means that ϵ-dominance mainly effects the final solutions and the ellipse and the bounded search prunes only unimportant labels from the search space.

To compare, the scalarised approach using multiple origins and multiple destinations version of the Dijkstra's algorithm (Section 5.3.2) achieves on the same origin-destination pairs an average running time 265 milliseconds while considering the *Commuting profile*.

The application that solves the multi-criteria bicycle route planning problem by the fastest pruning techniques (*MCD_Buckets*, the *MCD_Bounded_Buckets*, and the *MCD_Ellipse_Buckets*) is available to try at a live deployment[1]. The Pareto set is shown in a form of the distribution of cycleway segments in the physical space.

[1] http://its.felk.cvut.cz/cykloplanovac-mlc/

Table 6.3: Evaluation of the pruning heuristics' performance on the whole Prague graph.

Pruning Heuristic	In 15 min	Runtime [ms]	σ_{runtime}	$\lvert\Pi\rvert$	$\sigma_{\lvert\Pi\rvert}$
MCD_Ellipse_Epsilon	100	2325	2731	12	6
MCD_Bounded_Ellipse_Epsilon	100	3941	3808	12	6
MCD_Bounded_Epsilon	100	7426	4958	12	6
MCD_Epsilon	100	14010	4316	12	6
MCD_Bounded_Ellipse_Buckets	100	46084	99831	158	160
MCD_Ellipse_Buckets	99	63553	127411	154	153
MCD_Bounded_Buckets	93	125850	130808	138	128
MCD_Buckets	36	540912	228385	140	111

6.2 Junction Extension

To evaluate the planning on the extended cycleway graph, we describe the cycle planner's behaviour on a chosen set of plans. Particularly, we look at the commuting profile (purple colour) in the scalarised multi-criteria solution method. Plans on the extended cycleway graph, i.e., junction aware plans, should avoid left turn and u-turn manoeuvres unless the junction is controlled by traffic signals. The proposed method for the graph extension provides trade-off between a more precise model of junctions and the number of edges and nodes in the graph. The extended cycleway graph for the whole Prague contains 141 117 nodes and 323 114 edges in contrast to the standard cycleway graph which contains 68 058 nodes and 186 554 edges. The edge-based graph used in the literature (see 2.4) would have 186 554 nodes.

In Figure 6.6, the junction aware plan uses the traffic signals to cross the street "Evropská" and to turn to street "Milady Horákové". All the described manoeuvres are highlighted in blue circle. It did not follow the path planned on the standard graph, because the path is crossing the street "Jugoslávských partyzánů" at junction without traffic signals and from the street with a much lower priority. However, the junction aware plan forces to cyclist pointlessly ride uphill along "Svatovítská" street.

Figure 6.7 shows other example of different routes on the extended and the standard cycleway graph. On the one hand, the junction aware plan avoids to ride on street "Na Slupí" by using the official bicycle route and then it uses junctions with traffic lights when it needs to cross the road. On the other hand, the commuting plan planned on standard cycleway graph contains several left turn manoeuvres (on street "Vyšehradská", "Na Moráni", and "Trojanova").

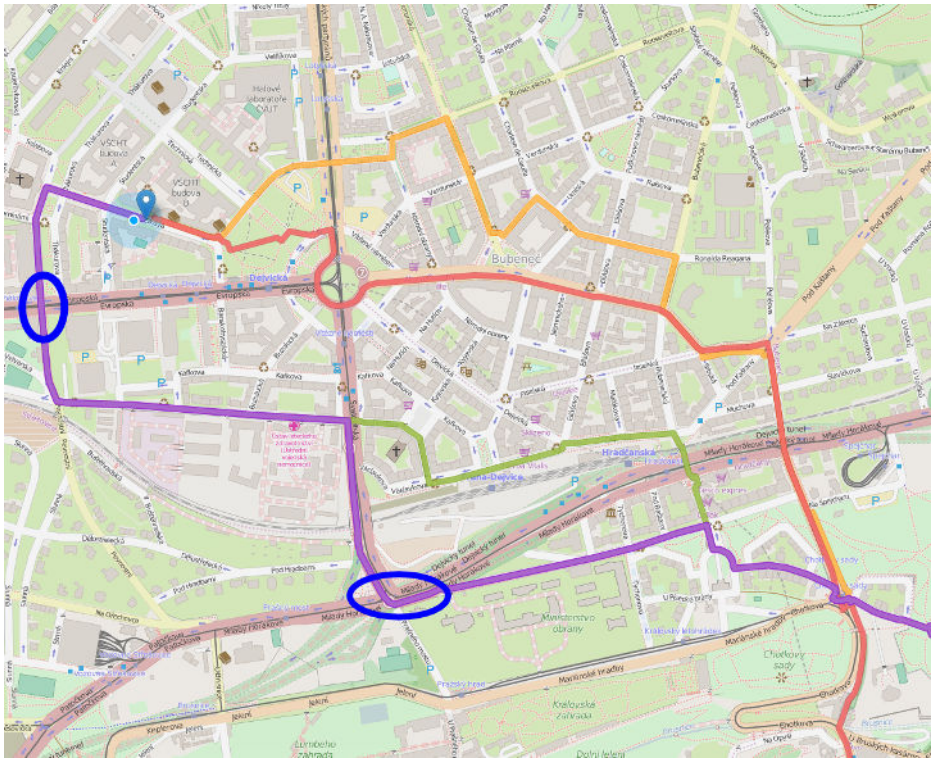
However, Figure 6.8 shows a case of an unfavourable manoeuvre. The case is caused by inappropriate setting of the straight manoeuvre parameter on that particular junction. This can be fixed by a more practical evaluation in order to set parameters for cycleway segments inside the junction properly. The bicycle routes planned on the standard graph can be viewed at web page[2] and the junction aware routes are avaliable here[3].

[2] cykloplanovac.cz
[3] its.felk.cvut.cz/cykloplanovac-its/

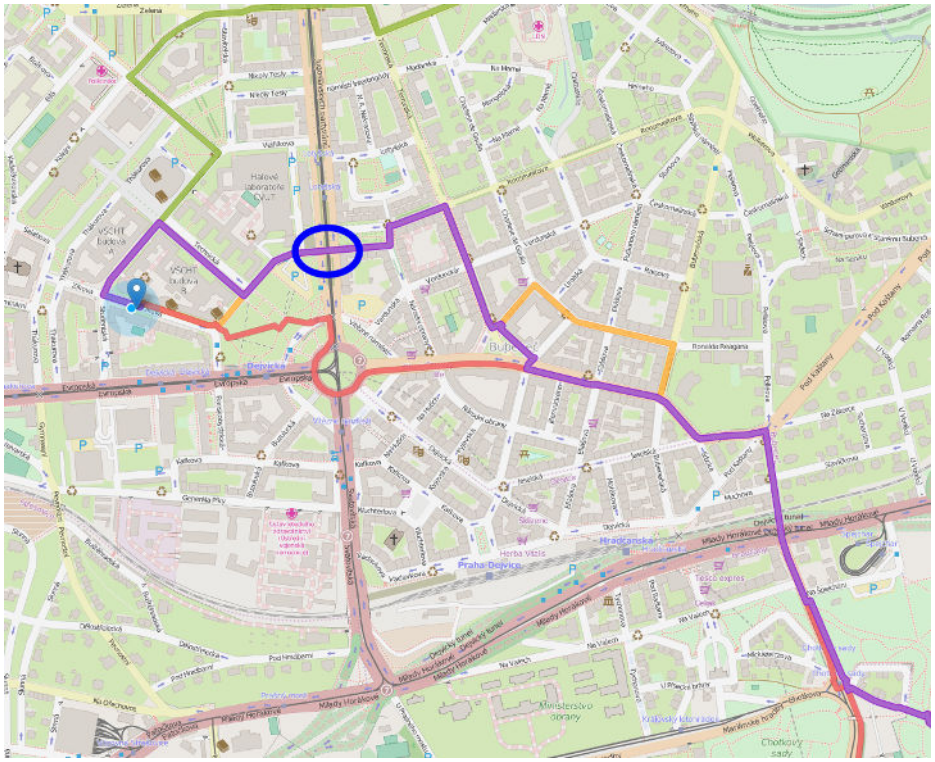
6.3 Summary

The evaluation of the proposed pruning heuristics shows that the combination of the ellipse and the ϵ-dominance is 4000 times faster then the pure multi-criteria Dijkstra's algorithm. This combination of pruning heuristics, also scale well and achieved the average running time 2.3 seconds on the whole Prague graph. Such a running time is a remarkable result, because it shows that the method is suitable to be used in real-time applications. On the other hand, the buckets heuristic yields to not sufficient results on the whole Prague graph. This could be caused by inappropriate setting of the bucket sizes.

The empirical evaluation of the extended cycleway graph confirmed that the model fulfils the requirements for which was designed except for the rare manoeuvres as shown in Figure 6.8. The cycle planner definitely behaves differently in commuting profile which prefers the comfort criteria.

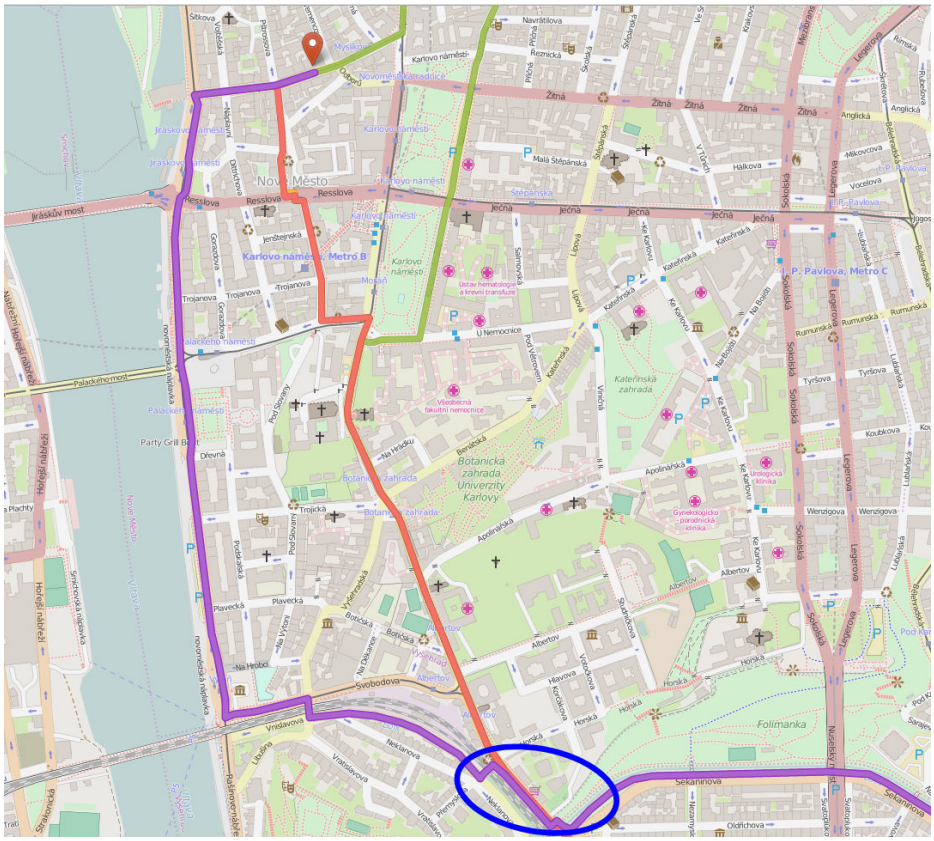


(a) the extended cycleway graph

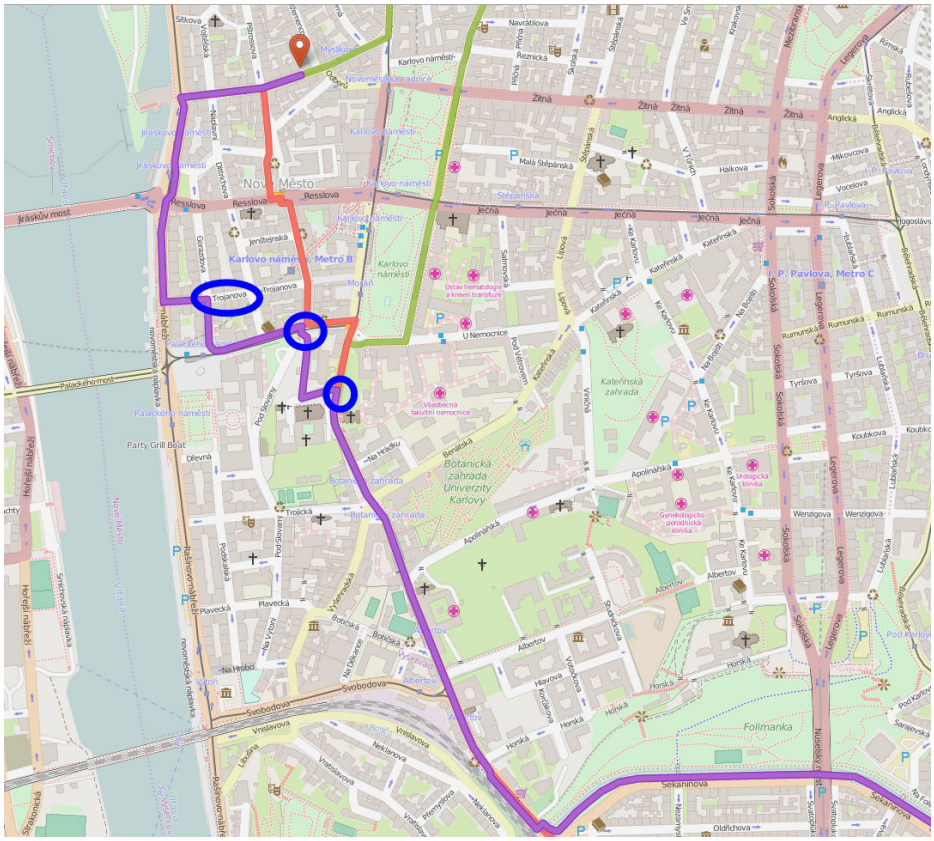


(b) the standard cycleway graph

Figure 6.6: Comparison of the commuting profile (*purple* route)



(a) the extended cycleway graph



(b) the standard cycleway graph

Figure 6.7: Comparison of the commuting profile (*purple* route)

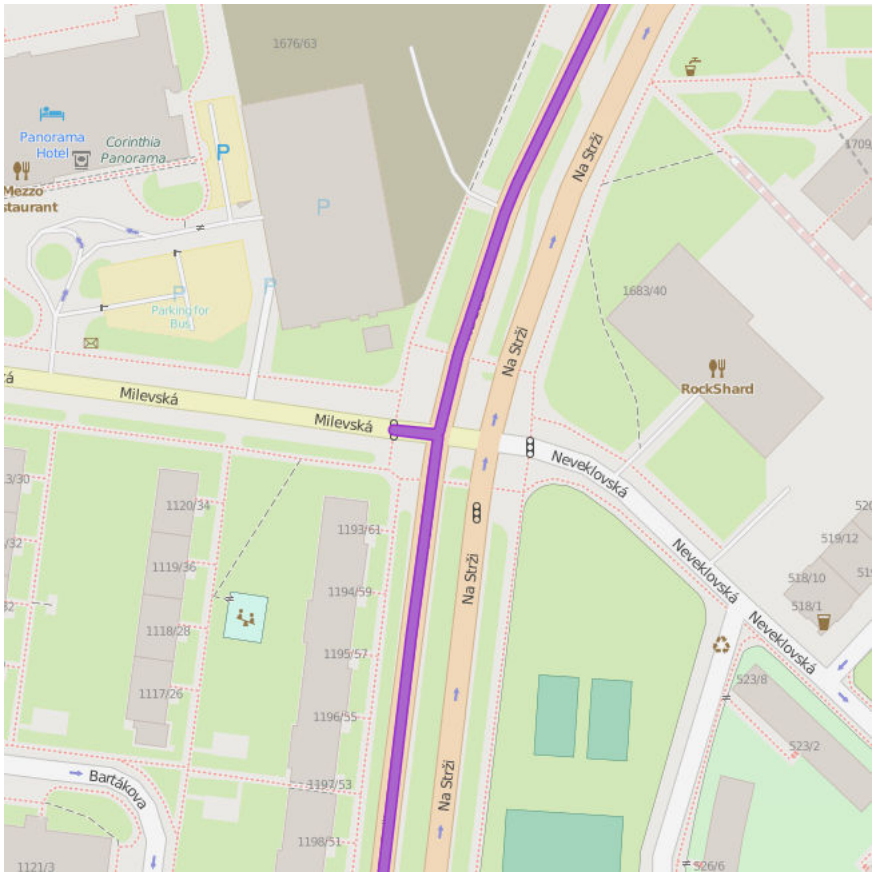


Figure 6.8: Unfavourable manoeuvre

Chapter 7

Conclusion

The primary aim of the thesis was to build a junction-aware cycle planner application capable of providing the multi-criteria search. In order to achieve this goal, we proceeded in the following steps.

First, we modelled a cycleway network as a graph structure and extended it to support turn penalisation on junctions. We introduced new entrance and exit nodes instead of the node representing the junction. Then, we connect these new nodes in order to create an edge representing a manoeuvre (e.g., left turn). This approach is different than methods described in Section 2.4 because we wanted to preserve the typical graph structure in order to not change the implemented algorithms.

To provide the multi-criteria search we started by proposing the scalarised solution method together with profiles describing the importance of individual criteria. Also, we proposed the pruning enabled multi-criteria Dijkstra's algorithm (MCD) together with 11 combinations of pruning heuristics in order to lower the MCD's running time. Although the pruning heuristics are relatively simple, their application in the context of bicycle route planning problem is novel and their effect is very significant.

In the implementation part, we described methods for more detailed OSM data preprocessing and methods for improving and optimising the cycleway graph in order to enhance the plan quality. To see the differences in bicycle routes based on the detailed data preprocessing and the new graph construction methods, visit the former[1] and the current[2] version of the cycle planner application. We also described how we implemented the junction extension procedure and we presented some of the important details of the whole cycle planner application and the algorithms. We show that the cycle planner is currently used in 25 cities in the Czech Republic and serves approximately 200 requests per day through web and Android frontends.

The evaluation part demonstrated that three of eleven proposed pruning heuristics (*MCD_Ellipse_Epsilon*, *MCD_Bounded_Epsilon*, and *MCD_Bounded_Ellipse_Epsilon*) are able to return the Pareto set in less then 10 seconds on a graph consisting of 323 000 edges and covering the whole region of Prague city. The best speed up over the MCD algorithm on the medium-sized graphs (containing around 20 000 edges) was almost 4 000

[1] http://transport.felk.cvut.cz/cykloplanovac-2014
[2] http://transport.felk.cvut.cz/cykloplanovac

and was achieved by the combination of the ellipse and the ϵ-dominance pruning heuristics. These are significant results because we are now able to return a much wider set of bicycle routes than in a scalarised approach.

Bicycle routes planned on the extended graph support the required properties such as avoiding left turn and u-turn manoeuvres unless the junction is controlled by traffic signals. However, it can be shown that manoeuvre penalisation parameters are not yet properly set. A disadvantage of this approach lies in the size of the graph. It increases with respect to the number of junction nodes, therefore, this could be critical in an environment with a low memory.

The multi-criteria search produces often large Pareto sets with many similar routes. As a future work, we plan to provide a filtering method that would extract several representative routes from a potentially very large set of Pareto routes. We aim to tune the penalisation parameters in the junction aware cycleway graph to improve the plans quality and to make this graph usable in a live deployment of the cycle planner application. The correct information about the traffic volume would improve this model fundamentally and make it more reliable. The data collected from the Android application could be used to improve the plan quality by taking into account the routes which cyclists prefer.

Bibliography

[1] D. Herlihy, *Bicycle: the history*. Yale University Press, 2004.

[2] W. H. Organization, "Obesity and overweight." <`http://www.who.int/mediacentre/factsheets/fs311/en/`>, Jan. 2014.

[3] J. Woodcock, P. Edwards, C. Tonne, B. G. Armstrong, O. Ashiru, D. Banister, S. Beevers, Z. Chalabi, Z. Chowdhury, A. Cohen, *et al.*, "Public health benefits of strategies to reduce greenhouse-gas emissions: urban land transport," *The Lancet*, vol. 374, no. 9705, pp. 1930–1943, 2009.

[4] Filler, V. (AUTO*MAT) Private communication, 2013-11-18. Why a cycle planner is important for cyclists.

[5] M. Němet, "Otevřený plánovač cyklistických tras," 2013.

[6] J. Broach, J. Dill, and J. Gliebe, "Where do cyclists ride? a route choice model developed with revealed preference gps data," *Transportation Research Part A: Policy and Practice*, vol. 46, no. 10, pp. 1730–1740, 2012.

[7] M. Winters, G. Davidson, D. Kao, and K. Teschke, "Motivators and deterrents of bicycling: comparing influences on decisions to ride," *Transportation*, vol. 38, no. 1, pp. 153–168, 2011.

[8] E. W. Dijkstra, "A Note on Two Problems in Connexion with Graphs," in *NUMERISCHE MATHEMATIK*, pp. 269–271, 1959.

[9] D. Delling, P. Sanders, D. Schultes, and D. Wagner, "Engineering route planning algorithms," in *Algorithmics of large and complex networks*, pp. 117–139, Springer, 2009.

[10] H. Bast, D. Delling, A. Goldberg, M. Müller-Hannemann, T. Pajor, P. Sanders, D. Wagner, and R. Werneck, "Route planning in transportation networks," in *Technical Report MSR-TR-2014-4*, Microsoft Research, Microsoft Corporation, 2014.

[11] P. Hart, N. Nilsson, and B. Raphael, "A Formal Basis for the Heuristic Determination of Minimum Cost Paths," *IEEE Transactions on Systems Science and Cybernetics*, vol. 4, pp. 100–107, Feb. 1968.

[12] F. B. Zhan and C. E. Noon, "A comparison between label-setting and label-correcting algorithms for computing one-to-one shortest paths," *Journal of Geographic Information and Decision Analysis*, vol. 4, no. 2, pp. 1–11, 2000.

[13] R. Geisberger, P. Sanders, D. Schultes, and D. Delling, "Contraction Hierarchies: Faster and Simpler Hierarchical Routing in Road Networks," in *Proceedings of the 7th international conference on Experimental algorithms*, WEA'08, pp. 319–333, Springer-Verlag, 2008.

[14] R. Bellman, "On a routing problem," tech. rep., DTIC Document, 1956.

[15] L. R. Ford Jr, "Network flow theory," tech. rep., DTIC Document, 1956.

[16] E. Q. V. Martins, "On a multicriteria shortest path problem," *European Journal of Operational Research*, vol. 16, no. 2, pp. 236–245, 1984.

[17] L. Mandow and J. L. P. De La Cruz, "Multiobjective A* search with consistent heuristics," *Journal of the ACM (JACM)*, vol. 57, no. 5, p. 27, 2010.

[18] Wikipedia, "Pareto efficiency." <`http://en.wikipedia.org/wiki/Pareto_efficiency`>, Apr. 2015.

[19] P. Hansen, "Bicriterion path problems," in *Multiple criteria decision making theory and application*, pp. 109–127, Springer, 1980.

[20] L. Han, H. Wang, and W. Mackey Jr., "Finding shortest paths under time-bandwidth constraints by using elliptical minimal search area," *Transportation Research Record: Journal of the Transportation Research Board*, vol. No. 1977, pp. 225–233, 2006.

[21] P. Perny and O. Spanjaard, "Near admissible algorithms for multiobjective search.," 2008.

[22] D. Delling, J. Dibbelt, T. Pajor, D. Wagner, and R. F. Werneck, "Computing multimodal journeys in practice.," in *SEA*, pp. 260–271, Springer, 2013.

[23] B. S. Stewart and C. C. White III, "Multiobjective A*," *Journal of the ACM (JACM)*, vol. 38, no. 4, pp. 775–814, 1991.

[24] C. T. Tung and K. L. Chew, "A multicriteria pareto-optimal path algorithm," *European Journal of Operational Research*, vol. 62, no. 2, pp. 203–209, 1992.

[25] E. Machuca and L. Mandow, "Multiobjective heuristic search in road maps," *Expert Systems with Applications*, vol. 39, no. 7, pp. 6435–6445, 2012.

[26] P. Sanders and L. Mandow, "Parallel label-setting multi-objective shortest path search," in *Parallel & Distributed Processing (IPDPS), 2013 IEEE 27th International Symposium on*, pp. 215–224, IEEE, 2013.

[27] M. Haqqani, X. Li, and X. Yu, "A multi-objective a* search based on non-dominated sorting," in *Simulated Evolution and Learning*, pp. 228–238, Springer, 2014.

[28] H. H. Hochmair and J. Fu, "Web Based Bicycle Trip Planning for Broward County, Florida," in *ESRI User Conference*, 2009.

[29] J. G. Su, M. Winters, M. Nunes, and M. Brauer, "Designing a route planner to facilitate and promote cycling in Metro Vancouver, Canada," *Transportation Research Part A: Policy and Practice*, vol. 44, no. 7, pp. 495–505, 2010.

[30] J. Hrncir, Q. Song, P. Zilecky, M. Nemet, and M. Jakob, "Bicycle route planning with route choice preferences," 2014.

[31] R. J. Turverey, D. D. Cheng, O. N. Blair, J. T. Roth, G. M. Lamp, and R. Cogill, "Charlottesville bike route planner," in *Systems and Information Engineering Design Symposium (SIEDS)*, 2010.

[32] D. W. Corne, "The good of the many outweighs the good of the one: evolutionary multi-objective optimization," *IEEE Connections Newsletter*, pp. 9–13, 2003.

[33] D. Delling, J. Dibbelt, T. Pajor, D. Wagner, and R. F. Werneck, "Computing and Evaluating Multimodal Journeys," Tech. Rep. 2012-20, Faculty of Informatics, Karlsruhe Institut of Technology, 2012.

[34] G. Sauvanet and E. Neron, "Search for the best compromise solution on multiobjective shortest path problem," *Electronic Notes in Discrete Mathematics*, vol. 36, pp. 615–622, 2010.

[35] Q. Song, P. Zilecky, M. Jakob, and J. Hrncir, "Exploring pareto routes in multi-criteria urban bicycle routing," in *Intelligent Transportation Systems (ITSC), 2014 IEEE 17th International Conference on*, pp. 1781–1787, IEEE, 2014.

[36] A. Byrd, "Opentripplanner: Routing bibliography." <`https://github.com/opentripplanner/OpenTripPlanner/wiki/RoutingBibliography`>, 2014.

[37] L. Volker, "Route planning in road networks with turn costs," *Studienarbeit, Universität Karlsruhe, Institut für theoretische Informatik*, 2008.

[38] R. Geisberger and C. Vetter, "Efficient routing in road networks with turn costs," in *Experimental Algorithms*, pp. 100–111, Springer, 2011.

[39] D. Delling, A. V. Goldberg, T. Pajor, and R. F. Werneck, "Customizable route planning," in *Experimental Algorithms*, pp. 376–387, Springer, 2011.

[40] J. Linka, "Bachelor's project: Android app for bicycle route planning and navigation," 2015 (to appear).

[41] M. Levandowsky and D. Winter, "Distance between sets," *Nature*, no. 5323, pp. 34—-35, 1971.

Appendix A

Tables

OSM element	key	value	r'_{tt}
way	footway	crossing	0.5
way	bicycle	dismount	0.5
way	footway	sidewalk	0.5
way	highway	footway	0.5
way	highway	"footway;path"	0.5
way	highway	pedestrian	0.5
way	highway	steps	0.1
way	highway	bridleway	0.7
way	access	agricultural	0.8
way	access	forestry	0.8
way	smoothness	bad	0.9
way	smoothness	very_bad	0.8
way	smoothness	horrible	0.7
way	smoothness	very_horrible	0.4
way	smoothness	impassable	0.4
way	surface	wood	0.9
way	surface	cobblestone	0.8
way	surface	paving_stones	0.8
way	surface	setts	0.8
way	surface	sett	0.8
way	surface	unpaved	0.7
way	surface	dirt	0.7
way	surface	grass	0.7
way	surface	gravel	0.7
way	surface	ground	0.7
way	surface	mud	0.4
way	surface	sand	0.4
way	oneway	opposite	5

Table A.1: The r'_{tt} function values

OSM element	key	value	q'
node	crossing	traffic_signals	15
node	highway	traffic_signals	15
node	highway	stop	8
node	crossing	island	8
node	crossing	uncontrolled	8
node	crossing	unmarked	8
node	crossing	yes	8
node	crossing	zebra	8
node	highway	crossing	8
node	highway	elevator	38
node	highway	steps	8
node	barrier	yes	8
node	barrier	block	8
node	barrier	chain	8
node	barrier	rope	8
node	barrier	cycle_barrier	8
node	barrier	motorcycle_barrier	8
node	barrier	gate	8
node	barrier	lift_gate	8
node	barrier	swing_gate	8
node	traffic_calming	yes	2
node	traffic_calming	bump	2

Table A.2: The q' function values

OSM element	key	value	r'_{co}
way	footway	crossing	1.5
way	bicycle	dismount	1.5
way	footway	sidewalk	1.5
way	highway	footway	1.5
way	highway	"footway;path"	1.5
way	highway	pedestrian	1.5
way	highway	steps	5
way	highway	bridleway	2
way	access	agricultural	2
way	access	forestry	2
way	smoothness	excellent	0.5
way	smoothness	good	0.8
way	smoothness	intermediate	-1
way	smoothness	bad	2
way	smoothness	very_bad	3
way	smoothness	horrible	4
way	smoothness	very_horrible	5
way	smoothness	impassable	5
way	surface	asphalt	0.5
way	surface	concrete	0.8
way	surface	paved	-1
way	surface	compacted	-1
way	surface	wood	2
way	surface	cobblestone	3
way	surface	paving_stones	3
way	surface	setts	3
way	surface	sett	3
way	surface	unpaved	4
way	surface	dirt	4
way	surface	grass	4
way	surface	gravel	4
way	surface	ground	4
way	surface	mud	5
way	surface	sand	5
way	oneway	opposite	4

Table A.3: The r'_{co} function values

OSM element	key	value	r'_{qu}
way	footway	sidewalk	0.9
way	highway	footway	0.9
way	highway	"footway;path"	0.9
way	highway	pedestrian	0.9
relation	route	bicycle	0.2
way	bicycle	designated	0.2
way	highway	cycleway	0.2
way	cycleway	track	0.4
way	cycleway	lane	0.4
way	cycleway:left	lane	0.4
way	cycleway:right	lane	0.4
way	cycleway	share_busway	0.6
way	cycleway:left	share_busway	0.6
way	cycleway:right	share_busway	0.6
way	cycleway	shared_lane	0.8
way	cycleway:left	shared_lane	0.8
way	cycleway:right	shared_lane	0.8
way	highway	living_street	0.7
way	highway	primary	5
way	highway	primary_link	5
way	highway	secondary	3
way	highway	secondary_link	3
way	oneway	opposite	5

Table A.4: The r'_{qu} function values

OSM element	key	value	p'
way	turn	straight_5parallel	0
way	turn	traffic_lights_straight_5parallel	34
way	turn	left_5parallel	97
way	turn	traffic_lights_left_5parallel	34
way	turn	right_5parallel	0
way	turn	traffic_lights_right_5parallel	0
way	turn	u_turn_5parallel	97
way	turn	traffic_lights_u_turn_5parallel	80
way	turn	straight_10parallel	0
way	turn	traffic_lights_straight_10parallel	34
way	turn	left_10parallel	146
way	turn	traffic_lights_left_10parallel	34
way	turn	right_10parallel	0
way	turn	traffic_lights_right_10parallel	0
way	turn	u_turn_10parallel	146
way	turn	traffic_lights_u_turn_10parallel	80
way	turn	straight_20parallel	0
way	turn	traffic_lights_straight_20parallel	34
way	turn	left_20parallel	372
way	turn	traffic_lights_left_20parallel	34
way	turn	right_20parallel	0
way	turn	traffic_lights_right_20parallel	0
way	turn	u_turn_20parallel	372
way	turn	traffic_lights_u_turn_20parallel	80
way	turn	straight_1cross	16
way	turn	traffic_lights_straight_1cross	16
way	turn	left_1cross	32
way	turn	traffic_lights_left_1cross	16
way	turn	right_1cross	0
way	turn	traffic_lights_right_1cross	0
way	turn	u_turn_1cross	32
way	turn	traffic_lights_u_turn_1cross	80
way	turn	straight_5cross	66
way	turn	traffic_lights_straight_5cross	34
way	turn	left_5cross	66
way	turn	traffic_lights_left_5cross	34
way	turn	right_5cross	32
way	turn	traffic_lights_right_5cross	0
way	turn	u_turn_5cross	66
way	turn	traffic_lights_u_turn_5cross	80
way	turn	straight_10cross	95
way	turn	traffic_lights_straight_10cross	34
way	turn	left_10cross	95
way	turn	traffic_lights_left_10cross	34
way	turn	right_10cross	61
way	turn	traffic_lights_right_10cross	0
way	turn	u_turn_10cross	95
way	turn	traffic_lights_u_turn_10cross	80
way	turn	straight_20cross	518
way	turn	traffic_lights_straight_20cross	34
way	turn	left_20cross	518
way	turn	traffic_lights_left_20cross	34
way	turn	right_20cross	61
way	turn	traffic_lights_right_20cross	0
way	turn	u_turn_20cross	518
way	turn	traffic_lights_u_turn_20cross	80

Table A.5: The p' function values

Appendix B

CD content

Attached CD contains source files of the cycle planner Java application.

To run the experiments specified in the evaluation chapter copy the following jar file to you local directory:

`code/experiment/target/MCExperiment.jar`

and then run e.g., ϵ-dominance pruning heuristic on the Prague medium-sized graph A type:

```
$ java -jar MCExperiment.jar -alg mc_dijkstra_epsilon -reg praguemediumajunction
-aOverB 1.5 -buckets 80 1120 2 -epsilon 0.025 -r 105
```

The results will be stored in the *csv* file in the directory

`results/praguemediumajunction/mc\_dijkstra_epsilon`

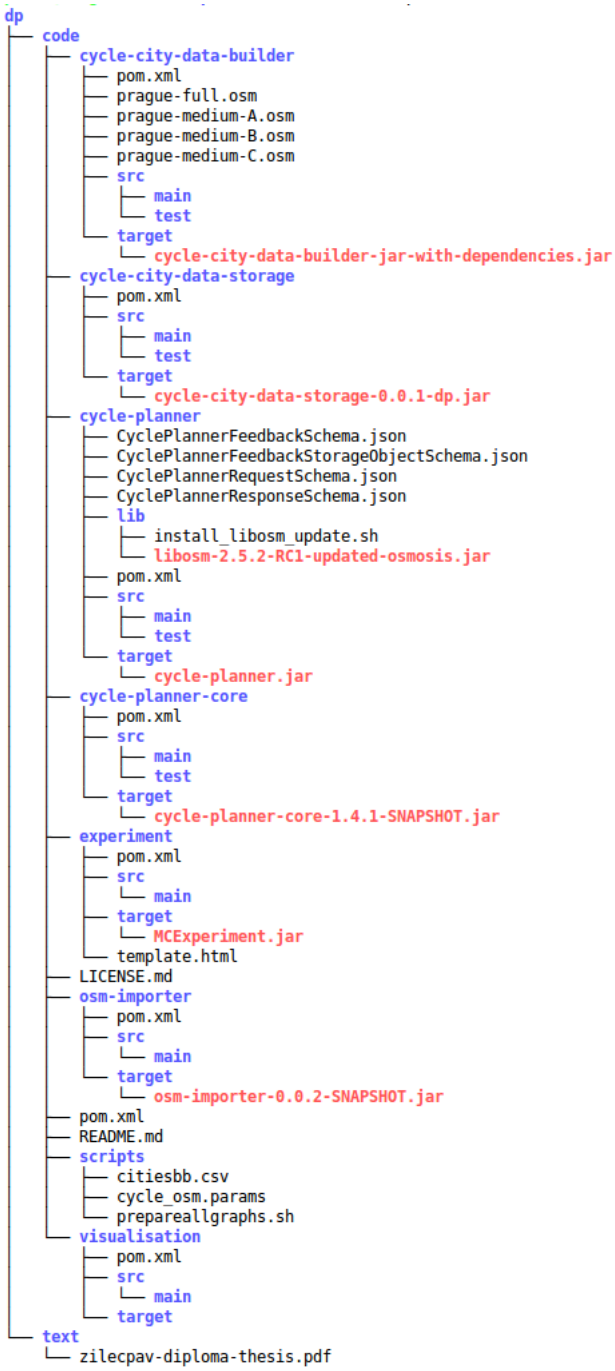


Figure B.1: CD content