Czech Technical University in Prague
Faculty of Electrical Engineering

Department of Computer Science and Engineering

# DIPLOMA THESIS ASSIGNMENT

Student: **Bc. Juraj Juráška**

Study programme: Open Informatics
Specialisation: Artificial Intelligence

Title of Diploma Thesis: **Multi-criteria route planning with emphasis on geospatial result diversity**

Guidelines:

1. Survey existing approaches to solve multi-criteria shortest path problem.
2. Formalize the problem of geospatial diversity-aware, multi-criteria route search in transport networks, i.e., searching for plans optimizing multiple route criteria while maximizing route diversity.
3. Design and implement an algorithm for the route search problem defined in 2.
4. Evaluate the implemented algorithm on large-scale transport networks and compare its results to selected state-of-the-art multi-criteria shortest path algorithms

Bibliography/Sources:

1. Bast, Hannah, et al. "Route planning in transportation networks." arXiv preprint arXiv:1504.05140 (2015).
2. Mandow, Lawrence, and José Luis Pérez De La Cruz. "Multiobjective A* search with consistent heuristics." Journal of the ACM (JACM) 57.5 (2010): 27.
3. Machuca, Enrique, and Lawrence Mandow. "Multiobjective heuristic search in road maps." Expert Systems with Applications 39.7 (2012): 6435-6445.
4. Hrncir, Jan, et al. "Speedups for Multi-Criteria Urban Bicycle Routing." OASIcs-OpenAccess Series in Informatics. Vol. 48. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2015.

Diploma Thesis Supervisor: Ing. Pavol Žilecký

Valid until the end of the summer semester of academic year 2016/2017

ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE Fakulta elektrotechnická

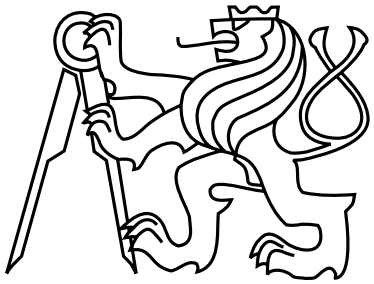
L.S.

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

prof. Ing. Pavel Ripka, CSc.
Dean

Prague, January 18, 2016

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



Master's Thesis

# Multi-Criteria Route Planning with Emphasis on Geospatial Result Diversity

*Juraj Juráška*

Supervisor: Ing. Pavol Žilecký

Study Program: Open Informatics

Field of Study: Artificial Intelligence

May 27, 2016

# Acknowledgements

First and foremost, I would like to extend my sincere gratitude to my advisor Pavol Žilecký for his expert guidance and continuous enthusiasm, as well as an excellent cooperation until the very last day. I wish to thank Jan Nykl for his insightful comments and challenging questions during our meetings with Pavol. My thanks also go to my parents without whose support I would have never made it through the tough times the last year brought. Finally, I wish to thank my brothers who were always there for me when I needed a boost of motivation or just some distraction after the hard work.

I also greatly appreciate the access to computing and storage facilities owned by parties and projects contributing to the National Grid Infrastructure MetaCentrum provided under the program "Projects of Projects of Large Research, Development, and Innovations Infrastructures" (CESNET LM2015042).

# 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 27, 2016 .....

# Abstract

People travel all the time, whether it be commuting, weekend bike trips or transportation of goods. We often require our smartphones or computers to find the quickest route for us to get somewhere. But what if we are not interested merely in the shortest or the fastest option and, instead, desire to choose from several alternatives that meet our expectations in a whole range of criteria? We know that searches accommodating multiple preferences simultaneously often find an excessive number of results, the ultimate selection from which may be awkward. We therefore consider the options for reducing the solution set's size. Our intention is to identify a few representative solutions that exhibit spatial diversity. To accomplish this, we design a multi-criteria algorithm that integrates the preference of geographically distinct solutions directly into the search, and we compare it with the method that simply filters the complete set of solutions based on their distinctness. We perform the evaluation on the road networks of the city of Prague and New York City. The proposed search algorithm indeed finds a set of routes with a convenient size, still covering all of the interesting route segments. The evaluation also shows that it is on average up to two times worse in terms of runtime, although there are certain cases in which it clearly outperforms the filtering method.

# Abstrakt

Ľudia neustále cestujú, či už sa jedná o dochádzanie, víkendové výlety na bicykli alebo prepravu tovaru. Často sa obraciame na svoje smartfóny alebo počítače s požiadavkou nájdenia najrýchlejšej trasy niekam. Avšak, čo ak nás nezaujíma zrovna najkratšia alebo najrýchlejšia možnosť a chceli by sme si vybrať z viacerých alternatív, ktoré spĺňajú naše očakávania v širšom spektre kritérií? Je známe, že vyhľadávanie, ktoré má uspokojiť viaceré preferencie naraz často nájde nadmerné množstvo výsledkov, z ktorých sa finálny výber robí len veľmi ťažko. Preto sme sa rozhodli preskúmať možnosti na zredukovanie množiny riešení. Naším zámerom je určiť zopár reprezentatívnych riešení, ktoré by preukazovali diverzitu v priestore. S týmto cieľom sme navrhli multikriteriálny algoritmus, v ktorom je zabudované uprednostňovanie geograficky jedinečných riešení priamo do procesu vyhľadávania, a porovnali sme ho s metódou, ktorá jednoducho vyfiltruje kompletnú množinu riešení podľa ich jedinečnosti. Vyhodnotenie prebiehalo na cestných sieťach Prahy a New Yorku. Navrhnutý vyhľadávací algoritmus skutočne nájde rozumne veľkú skupinu ciest, ktoré pokrývajú všetky zaujímavé úseky. Experimenty tiež ukázali, že v priemere síce dosahuje zhruba dvakrát horšie časy, no v niektorých prípadoch dokázal jednoznačne prekonať metódu s filtrovaním.

# Contents

# List of Figures

# List of Tables

*"If we cannot now end our differences, at least we can help make the world safe for diversity."*
— John F. Kennedy

# Chapter 1

# Introduction

## 1.1 Motivation

Most people travel every day using various means of transportation available in today's society. Depending mostly on the size of the city or town they reside in and their occupation, people may have to commute to work or they drive a vehicle as a part of their job. In some countries, they rarely walk to a store anymore and, if they have some spare energy on weekends, they try to make up for the time spent in the office during the week by driving off to the mountains or to visit their relatives. Others ride their bicycles to the nearest swimming pool or out of town to explore new tracks in the nearby woods. What all these scenarios have in common is that they require a plan how to get from one place to another.

In most cases, we are interested in either the shortest or the fastest way to reach our destination. We already have apps in our smartphones that can find the fastest route from our current location almost instantly. There are groups of people, however, who travel for enjoyment, which mostly includes the cyclists, or who travel between the same two points frequently and desire some variation, which again chiefly concerns the bicycle commuters. These often have a different set of criteria based on which they decide how good a route is. The amount of uphill climbing, the surface type and quality of the roads, as well as the busyness of the streets, all play an important role in determining the convenience of a route when you are riding a bike, perhaps even more so than the travel distance and time. Planning that takes multiple factors into consideration typically produces many results with different trade-offs between the individual criteria. The results naturally vary in their geographic setting too. This enables the cyclists to choose an alternative that best suits their preferences. However, it also poses a computational challenge far greater than the ordinary route planning with a single criterion.

In the recent years, the research interest in diverse multi-criteria route planning has primarily been attracted by the efforts to improve planning for the hazardous material (hazmat) transportation. The criteria that can be used to optimize the routes range from distance and time (in the carrier's interest) to weather conditions, traffic flow and population density along the route (mostly in public interest). It is thus typically desired to find spatially different routes that can be alternated between for a series of shipments with the goal of spreading the risk they impose across a larger area, while not giving up the efficiency of the routes.

Although accidents involving hazmat are rare, they are disastrous when they do happen, and their consequences may, directly or indirectly, affect thousands of people, as well as they can be very costly. Efforts to enhance the current methods for hazmat transportation planning are therefore deemed worthwhile.

## 1.2 Scope and Goals

Our aim in this thesis is to identify a suitable way of integrating the preference of geographically distinct solutions into the multi-criteria search in road networks. We shall design an algorithm that finds a set of solutions with a more convenient size and an emphasis on spatial diversity among the solutions.

Finding optimal paths when taking multiple criteria into account is a difficult problem to solve, as explained in Section 2.2. The runtime of a multi-criteria search increases substantially compared to its single-criterion counterpart. Furthermore, the number of solutions to such a problem can explode depending on the origin-destination distance and the number of criteria in consideration. When the solution set is enormous, the ability to choose the most suitable one among them is markedly impaired. And we are talking hundreds or thousands of solutions only considering two or three criteria. It is therefore essential to devise a method to filter the exhaustive solution set found by a multi-criteria search.

What we shall strive to accomplish is avoid the necessity of computing the entire solution set in the first place and enhance the search in such a way that will enable it to come up with a reasonably sized and, at the same time, diverse set of solutions directly. The emphasis on diversity should ensure that the solution reduction does not render attractive route segments unavailable in the condensed solution set. By adjusting the focus of the search, this approach could possibly improve the performance, making thus the multi-criteria search not only more convenient for use, but also better suited for real-time applications.

To assess the quality of our accomplishment, we shall evaluate the algorithm against the full multi-criteria search with an added diversity filter supplementarily applied to the solution set. The evaluation will be performed on large real-world road networks in order to maximize the relevance of the results.

## 1.3 Outline

We begin with a brief introduction to the shortest path problem, which is the basis of route planning, in Chapter 2. The necessary background of its multi-criteria extension along with the related work is then reviewed as we move toward the overview of the recent approaches to solution diversity in the multi-criteria search. In Chapter 3, we specify several techniques for determining the path difference, which will enable us to evaluate the diversity of a solution set. We then proceed with a description of the state-of-the-art multi-criteria search algorithm in Chapter 4, followed by our proposition of a diversity-aware extension of the algorithm. The implementation of the algorithms is described in Chapter 5, and their performance, as well as the solution quality, is then evaluated in Chapter 6. Finally, we conclude the thesis with a review of our achievement and a discussion of the future work in Chapter 7.

# Chapter 2

# Related Work

This thesis focuses on diversity-oriented route planning with multiple criteria, where by route planning we mean an application of the shortest path problem in a road network. Graph $G = (V, E, c)$, representing a road network, is defined by a set $V$ of nodes and a set $E$ of edges connecting the nodes. Nodes typically correspond to junctions, while edges represent the road segments between them. Each edge $(u, v) \in E$ has a cost $c(u, v)$ assigned to it. A *path* from an origin $s \in V$ to a destination $t \in V$ is a sequence of edges connecting distinct nodes starting in $s$ and ending in $t$. The shortest path between $s$ and $t$ is that which minimizes the sum of the costs of its edges. It is often called the *distance* between $s$ and $t$, denoted $d(s, t)$.

Dealing with multiple criteria means that each edge has multiple costs. It typically results in a multitude of optimal solution paths, none of them strictly better than another. Depending on the size of the problem, the number of all optimal solutions can be overwhelming. To facilitate the selection of the most suitable solution for a given purpose, the set of all solutions can be filtered based on their distinctness in a specific aspect from the other solutions, thus promoting a high diversity with respect to the chosen aspect in the filtered solution set.

The sections of this chapter will gradually introduce the well-researched problem of finding the shortest paths in a graph, followed by its multi-criteria variant along with several performance-improvement techniques and, finally, various approaches to obtaining a spatially diverse set of solutions.

## 2.1 Shortest Path Problem

Depending on the size of the set of origin nodes and the set of destination nodes, we can distinguish several classes of the shortest path problem. The *point-to-point* problem is to find the shortest path between a single pair of nodes in a graph. In the *one-to-many* problem, a set $T$ of destination nodes is given and the shortest path is to be computed from an origin $s$ to all $t \in T$. When $T = V$, the problem is called *one-to-all*. Analogically, when we have a set $S$ of origin nodes and would like to find the shortest path from each $s \in S$ to a destination $t$, it is a *many-to-one* problem, with its probably most common variant being the *all-to-one* problem. The *many-to-many* problem defines both a set $S$ of origin nodes and a set $T$ of destination nodes between which the shortest paths are to be found. Finally, there is the

*all-pairs* problem, in which we have to compute the shortest path between all pairs of nodes in the graph, which is a variant of the many-to-many problem with $S = T = V$. The actual shortest paths are often not required when solving the problems and it is sufficient to only calculate the distance between the set of origins and the set of destinations.

A static road network is represented by a directed graph with non-negative edge costs. The standard approach to solving a point-to-point shortest path problem in such a graph uses Dijkstra's algorithm [13]. At the beginning, it sets the distance of the origin node $s$ to zero, and the tentative distance of all other nodes from $s$ to infinity. It maintains a priority queue $Q$ in which the nodes are ordered by their distance from $s$. Initially, the only node contained in $Q$ is $s$. In each iteration, a node $u$ with the lowest distance from the origin is extracted from $Q$, and the outgoing edges $(u, v) \in E$ to all of $u$'s unvisited neighbors $v$ are inspected. We will refer to this step as an *expansion*. Whenever $d(s, u) + c(u, v)$ is better than the current tentative distance to $v$, the distance is updated and $v$ is added to $Q$ with its new tentative distance as key. Dijkstra's algorithm is a *label-setting* algorithm, which means that, once a node $v$ is visited, its tentative distance from the origin becomes final and equals the minimal distance $d(s, v)$. As a result, the algorithm can stop as soon as the destination node $t$ is visited. The set of nodes visited by the algorithm is called the *search space*, which is represented by a data structure called the *search graph*. In case of Dijkstra's algorithm the search graph is a tree.

There are multiple variants of Dijkstra's algorithm that optimize its performance, mostly by reducing the search space. The *bidirectional* variant runs two simultaneous searches, one forward from the origin $s$ and the other backward from the destination $t$, until the intersection of their individual search spaces contains a node that lies on the shortest path from $s$ to $t$. Another class of variants uses *goal-directed* techniques. These guide the search in the direction of $t$ in order to avoid exploring an unnecessarily large space that lies within $d(s, t)$ from $s$, as Dijkstra's algorithm does.

A popular goal-directed algorithm is the A* search [20]. It uses a heuristic function $h : V \to \mathbb{R}$ to estimate the distance of a node $u$ from the destination node $t$, where $h(u)$ is a lower bound on $d(u, t)$. The priority of $u$ in the queue is then determined by the sum $d(s, u) + h(u)$, which results in preferring the nodes that are closer to the destination. If $h(u) = d(u, t)$ for $\forall u \in V$, the search visits only the nodes along the shortest path. In road networks, the heuristics typically exploit geometric properties of the graph, such as the Euclidean distance between two points.

The performance gain using geometric lower bounds may, however, be insignificant. The ALT algorithm [17], stemming from A*, improves the lower bounds used in its heuristic function by employing graph preprocessing. In this initial step, a small subset of $V$ is selected as a group of *landmarks*, while storing the distances between them and all nodes in the graph. These are then used in triangular inequalities to compute the lower bounds on the nodes' distances from the destination in the actual query.

A different approach to solving the shortest path problem uses the Bellman-Ford algorithm [9][14]. It repeatedly iterates over all edges and improves the distances of the nodes where appropriate. Since the nodes may be visited and updated multiple times during one search, it belongs to the class of *label-correcting* algorithms. The main advantage of the Bellman-Ford algorithm over Dijkstra's algorithm is that it also works when there are edges

with negative costs in the graph. On the other hand, it intrinsically computes the shortest paths from an origin to all nodes, which is a redundant work for a point-to-point problem.

The last decade has seen a significant development in the area of route planning in transportation networks, and there are recent algorithms that work efficiently in large-scale road networks. There are new goal-directed methods, such as Arc Flags [21][23], alternative approaches including *hierarchical techniques* (e.g. Contraction Hierarchies [16]), *bounded-hop* techniques (e.g. Hub Labeling [11][15] or Transit Node Routing [6][5][38][3]) and various hybrid algorithms that offer additional speedups (e.g. SHARC [7] – combining the computation of shortcuts with multilevel Arc Flags, or CHASE [8] – combining Contraction Hierarchies with Arc Flags). For an extensive survey and performance comparison of these and other shortest path algorithms refer to [4].

## 2.2 Multi-Criteria Shortest Path Problem

The previous section only considers the scenario in which each of the graph's edges is assigned a single cost. In a more advanced scenario, we may be interested in optimizing across multiple criteria simultaneously, typically, minimizing multiple objective functions. Hence, it is often called a multi-objective shortest path problem. In this case, instead of a single cost value each edge is labeled with a cost vector $\vec{c}$, the values of which correspond to the individual criteria. In road networks these criteria could be the obvious – time requirement and travel distance – but also, for example, fuel consumption (which may be an indication of the presence of mountain passes along the path), road type (freeways, residential area streets, etc.) or the number of intersections with traffic lights.

The multiple criteria render the problem more difficult because comparing two vectors is not as straightforward as comparing two scalar values. We can utilize a partial order relation between two cost vectors that determines whether one vector dominates (i.e. is "lower than") the other vector or not. The primary implication of the dominance being a partial order relation is that there may be multiple non-dominated paths from the origin node $s$ to an arbitrary node $u \in V$. Therefore, unlike in Dijkstra's algorithm, it is possible that $u$ gets expanded more than once during the search for the shortest path. Another consequence is that there is no single optimal solution to a multi-criteria shortest path problem. Instead, the solution is a set of all non-dominated paths from $s$ to the destination node $t$. A tree is no longer a sufficient representation of the search space in a multi-criteria problem, so it is replaced by a directed acyclic graph.

Consequently, the multi-criteria search is considerably more complex than its scalar counterpart. The number of node expansions can grow exponentially with the solution depth in the worst case, even in a problem with two criteria [18]. However, it has also been shown that in certain classes of the problem, the size of the solution set grows only polynomially with the destination depth [33]. Such behavior is exhibited in particular by problems with polynomially sized graphs, bounded integer costs and a fixed number of criteria [30].

Similarly to the single-criterion search, there are two major methods of solving the multi-criteria problem – label correcting and label setting. In a multi-criteria search algorithm, a *label* represents a unique non-dominated path to a particular node, and its main component is the cost vector of the path. The *label-correcting* approach [39] builds on its single-criterion

version by keeping a set of labels for each node. When a node $u$ is extracted from the priority queue, all of its labels are expanded along all outgoing edges $(u, v)$. For each label-edge pair there is a new label added to the respective successor node $v$, which is created by summing the edge's cost vector and the cost vector of the label being extended. Any existing label of $v$ that is dominated by the new label is subsequently removed. And conversely, the new label is removed (or not added to the successor node in the first place) if it is dominated. All nodes whose set of labels changes have to be extended in the next iteration. When there is no node to extend, the algorithm terminates.

The main difference of the *label-setting* approach lies in the label selection. Instead of a repeated expansion of the labels of many nodes, the algorithm maintains a queue of labels ordered lexicographically by cost vectors and, in each iteration, selects the first label in the queue for expansion. This label is associated with a particular node $u$ and so the label is expanded along all outgoing edges $(u, v)$ in the same way as in the previous method. Dominated labels must be removed both from the sets of labels of the corresponding nodes and from the queue. The search terminates as soon as the queue is empty. The algorithm using this technique introduced by Martins in [32], may as well be considered a multi-criteria extension of Dijkstra's algorithm. Additional methods for solving the multi-criteria problem are reviewed in [36].

One of the first multi-criteria search algorithms, which became the foundation for numerous extensions, was MOA*. As the name suggests, it is a multi-objective adaptation of A* and hence it implements a lower-bound function for estimating the costs of the paths to the destination for each criterion. It was presented, and proven complete and admissible, by Stewart and White in [40]. Although a label-setting algorithm, MOA* employs *node expansion*, i.e. it expands one node at a time generating new labels for its successors from the whole set of the node's labels. The node selected for expansion must have at least one label that is not dominated by any solution cost already discovered, nor by a label of any other node in the queue (note that the queue contains nodes instead of individual labels). However, node expansion has recently been shown not to be a good choice for the multi-criteria search. Not only can it be outperformed by a blind search technique, but also better informed lower-bound heuristics can actually decrease its performance, which is contrary to the purpose of the heuristics [26][34].

An alternative label-setting approach uses *label expansion*, upon which Mandow and Pérez de la Cruz built the NAMOA* (New Approach to Multi-Objective A*) algorithm [29]. Its efficiency has been shown to improve with better lower-bound estimates. Furthermore, NAMOA* has been proven to be optimal over the class of admissible multi-criteria algorithms, and to strictly dominate MOA* [28]. The algorithm was evaluated on fairly large-scale, realistic road networks (containing up to $10^6$ nodes), achieving a feasible performance for bi-criteria problems [25], but still not good enough for real-time applications. Since our work builds on NAMOA*, the algorithm is presented in detail in Chapter 4. Recent efforts enhanced the algorithm by implementing a more efficient graph structure [27] and by parallelizing the search [37].

Given the complexity of the multi-criteria problem, a complete search is intractable in real time for very large instances. This realization has spawned several heuristic approaches in attempt to speed up the search. The recently proposed NSMOA* algorithm [19] orders the non-dominated labels in the queue according to their crowding distance instead of the

default lexicographic order. The reasoning behind this idea is that a solution with a low crowding distance is in a dense area of the solution space, which presumably makes such a solution superior to those with a higher crowding distance. This should, in particular, lead to a faster discovery of the first solution and, with it, an earlier start of the search space pruning. However, the algorithm's performance was compared to that of NAMOA* only on very small graphs (up to 2000 nodes), and the only performance metric used was the number of expanded nodes before the first solution was found. That makes NSMOA*'s superiority in large graphs with hundreds of solutions questionable at best. Moreover, the absent runtime comparison makes it unclear whether the increased computational complexity, caused by the new sorting method, does not cancel out the modest performance boost.

A more promising extension of NAMOA* is the LEXGO* algorithm [35], which introduces label sorting by lexicographic *goal preferences*. The goal preferences define a *target* value for each criterion. They can be grouped in multiple *priority levels*, each containing a set of one or more preferences; each goal preference is assigned to exactly one priority level. The principle of optimality does not apply to goal preferences and, therefore, a more complex pruning condition is employed so as to make the reduction of the number of explored paths during the search possible. This naturally causes some overhead, but overall, the algorithm displays a significant performance increase (up to four orders of magnitude) at the expense of the size of the solution set.

The more restrictive the goals are, the greater is the reduction of the number of labels expanded during the search as well. It is, however, important to mention that the targets represent preferences, not constraints, and, therefore, feasible solutions do not necessarily have to satisfy all of them, or in fact, any of them. What matters is the difference of the label's cost vector from the goal preferences. The algorithm introduces *deviation vectors* to capture this difference and uses them for determining the order in which the labels become expanded (the lower-bound estimates are only used in case of equality of the deviation vectors).

LEXGO* returns a bounded subset of solutions – those that satisfy a set of predefined goals for each criterion or those that minimize the deviation from the goals in case there is no solution that satisfies all goals at the same time. A subset of solutions does not necessarily imply a worse outcome, for instance, when the complete solution set contains hundreds of paths, which would very likely be redundant in a real-world scenario anyway. The authors of the algorithm also proved that LEXGO* always explores a subset of the labels explored by a full Pareto search, such as in NAMOA*.

The trade-off between the runtime and the number of solutions in LEXGO* could in fact be useful for real-time route planning. However, when reducing the search space, we may be more interested in preserving a limited subset of solution paths that are as diverse as possible.

## 2.3 Solution Diversity

An important issue in the real-world multi-criteria route planning that needs to be addressed, if its results are to be of any practical use, is the size of the solution set. When the search yields a hundred solutions from which none can be declared better than another, at least

not based solely on the criteria as provided, they need to be considered from yet another perspective. There are a few natural ways how humans would filter such a set of optimal solutions in order to pick a handful of the most useful ones. The selection becomes rather infeasible for the humans, however, when the number of solutions found by the search is even greater, such as in thousands or tens of thousands, which is not an exception in problems with three or more criteria. Therefore, it would be convenient to extend the multi-criteria search to perform even this reduction for us and only offer a small subset of all optimal solutions in order to facilitate the final decision making.

The previously described algorithm LEXGO* does indeed accomplish this by introducing a sort of target values for each criterion and different levels of preference for the criteria. Declaring one criterion more important than another is one of the natural ways to decide which solutions stand out among the rest. In this case, a 10-km-long route that only takes 15 minutes would be preferable to an 8-km route that requires 25 minutes, supposing the time criterion is more significant than the distance. LEXGO* integrates this filtering directly into the search with the further benefit of decreasing the size of the search space explored and, with it, the necessary computation time.

Considering that route planning takes place in a geographical space, another intuitive way of reducing the number of solutions is by selecting the most *spatially dissimilar* routes, such as those that share the least part with other solution routes. This, at first, was applied by Akgün et al. in [2] to single-criterion route planning, wherein a number of candidate routes were first generated using various methods, such as $k$-shortest paths or the iterative penalty method (IPM), and subsequently a diverse subset was picked using the $p$-dispersion algorithm. The $p$-dispersion [22] selects a subset of a fixed number of routes such that the minimum difference between two routes in the subset is maximized.

Dell'Olmo et al. [12] took it one step further by using it to find diverse routes in a multi-criteria shortest path problem, which seems only natural, since a multi-criteria search yields a set of optimal solutions that can be supplied as the input candidates to the $p$-dispersion. The authors also argued that computing the dissimilarity of paths based merely on the edge intersection may lead to finding two parallel solution paths not distant enough to be considered satisfactory in some cases. Hence, they proposed defining buffer zones, i.e. virtual bands of a certain width along the paths, and calculating the intersections of these zones instead.

Martí et al. [31] reviewed the previous approaches and devised a GRASP algorithm adapted to the bi-criteria problem. This algorithm finds an approximate subset of non-dominated candidate solutions, which are, however, generated by combining $k$-shortest paths and IPM, and filtered using a randomized destructive method (RDE). This method improves the diversity over the combination of $k$-shortest paths / IPM with $p$-dispersion, but it is not exact and it does not employ a proper multi-criteria approach.

Another strategy, presented by Caramia et al. in [10], employs $k$-means clustering to split up all optimal paths into $k$ groups minimizing the cost vector variance within the groups. A single path is then heuristically chosen from each of the groups such that the spatial diversity among them is maximized. This extension of the fully multi-criteria approach by Dell'Olmo et al. was tested with three objectives and real-world data from an Italian region of Lazio. The corresponding road network consisted of 311 nodes, which can be considered an extremely small problem instance where possible shortcomings of the algorithm are unlikely

to emerge. In a larger instance, it might be quite common that two routes with very similar cost vectors are completely different spatially, but they would fall into the same group during the clustering phase. Eventually, only one of them would be picked as a solution, resulting rather in the suppression of diversity.

Experiments on very small networks are in fact an issue present in the majority of the literature on multi-criteria route planning, probably due to the computational complexity of the problem. This was one of the motivations for us to work with large-scale route networks when developing our own extension of a multi-criteria search algorithm.

All of the above approaches to finding a spatially diverse set of solution routes have one thing in common – they generate a set of candidate solutions first and then filter the most dissimilar ones, mostly a fixed number of them regardless of the problem instance. This, however, appears to be a waste of computational power when the search has to explore a multitude of solutions only to choose a small fraction of them in the end. In this thesis, we will examine an alternative approach, which is more similar to that of LEXGO* (an extension of NAMOA*) in that it directs the multi-criteria search straight toward the solutions with a desired property first. This enables it to stop as soon as the set of solutions found is satisfactory, with no need to search for all the optimal solutions. We present a new extension of NAMOA* that is driven by the path dissimilarity.

# Chapter 3

# Problem Specification

We begin this chapter with several standard definitions that are fundamental for the explanation of the multi-criteria search algorithm in Chapter 4. We are only interested in the point-to-point class of search problems, and all definitions are formulated with this in mind. Afterwards, we go on to discuss the spatial diversity of paths in a set, such as the Pareto set of solutions found by a multi-criteria search. We present several metrics to calculate the difference between two paths, which is used for determining the diversity of a set of paths.

## 3.1 Basic Definitions

In a problem with multiple criteria, it is not always possible to assert whether one cost vector is less than another. Instead, we introduce a strict partial order relation $\prec$, called *dominance*:

$$\vec{x} \prec \vec{x'} \iff x_i \leq x'_i \ \wedge \ \vec{x} \neq \vec{x'} \quad \forall i \in 1, \ldots, n\,,$$

where $\vec{x}, \vec{x'} \in \mathbb{R}^n$ and $x_i$ denotes the $i$-th component of $\vec{x}$ (corresponding to the $i$-th criterion).

Following the above definition, we say that vector $\vec{x}$ *dominates* vector $\vec{x'}$ if there is at least one criterion in which $\vec{x}$ is lower than $\vec{x'}$, and no criterion in which $\vec{x}$ is greater than $\vec{x'}$. To illustrate the relation, let us consider three simple vectors $(1, 2)$, $(1, 3)$ and $(3, 1)$. Vector $(1, 2)$ dominates $(1, 3)$, but it does not dominate $(3, 1)$, nor is it dominated by $(3, 1)$. Therefore, vectors $(1, 2)$ and $(3, 1)$ are *non-dominated* in the set of these three vectors. To express that vector $\vec{x}$ *dominates or equals* vector $\vec{x'}$, we will use the $\preceq$-relation.

Given a set $X$ of vectors, $\mathcal{N}(X)$ is the set of non-dominated vectors in $X$. $\mathcal{N}(X)$ is called a *Pareto set* and is defined as follows:

$$\mathcal{N}(X) = \{\vec{x} \in X \mid \nexists \vec{y} \in X \quad \vec{y} \prec \vec{x}\}\,.$$

The vectors in $\mathcal{N}(X)$ are called *Pareto optimal*, as there exists no vector in $X$ that would improve on one component of a vector from $\mathcal{N}(X)$ without degrading another component. A Pareto set is bounded by two points:

- the *ideal point* $\vec{\alpha} = (\alpha_1, \ldots, \alpha_n)$, where $\alpha_i = \min_{\vec{x} \in \mathcal{N}(X)}\{x_i\}$,
- the *nadir point* $\vec{\beta} = (\beta_1, \ldots, \beta_n)$, where $\beta_i = \max_{\vec{x} \in \mathcal{N}(X)}\{x_i\}$.

While each element $\alpha_i$ of the ideal point can easily be calculated for a multi-criteria shortest path problem by optimizing the $i$-th criterion separately, it is difficult to calculate the exact nadir point for $n > 2$ without finding the complete Pareto set of solutions first.

In the multi-criteria search algorithms, we will not be exclusively interested in the dominance relationship of the vectors though. What we shall find useful in addition to the dominance relation among vectors, is the *lexicographic order* $\prec_L$ defined as:

$$\vec{x} \prec_L \vec{x'} \iff \exists j \mid x_j < x'_j \ \wedge \ \forall i < j \ \ x_i = x'_i \,.$$

Note that the lexicographic minimum of a set $X$ of vectors is necessarily a non-dominated vector in $X$.

In a multi-criteria search algorithm, we typically use *labels*, which are structures containing information about a unique path $P_{su}$ from the origin $s$ to the node $u$. The simplest label $L_u = (u, \vec{g}(P_{su}), L_{u\mathrm{Pred}})$ contains the last node $u$ of the path (also the node to which the label is assigned), the cost vector $\vec{g}(P_{su})$ of the path and the label $L_{u\mathrm{Pred}}$ of which $L_u$ is the extension. $L_{u\mathrm{Pred}}$ is assigned to the immediate predecessor of $u$ on the path $P_{su}$. In these terms, $L_u$ dominates $L_v = (v, \vec{g}(P_{sv}), L_{v\mathrm{Pred}})$ when $\vec{g}(P_{su}) \prec \vec{g}(P_{sv})$. Note that there may be multiple labels assigned to a single node, since there may be different paths from the origin to the node, therefore it is possible for $v$ to equal $u$. The set of all non-dominated paths (i.e. paths whose cost vectors are non-dominated) from the origin node $s$ to the destination node $t$ is the Pareto set of solutions of a multi-criteria shortest path problem. Throughout the thesis, we will often compare labels with paths, as labels essentially represent paths.

## 3.2 Path Diversity

Different paths in a graph can have certain edges in common. When we have to deal with a whole set of paths from the same origin node to the same destination node, it may be useful to differentiate between these paths, as, in some cases, we are simply not interested in all of them. One such case is multi-criteria route planning, where comparing the paths based on their cost vectors might not be convenient or sufficient to determine which path is better. We therefore consider another aspect that enables us to compare paths – the spatial difference. It is especially useful in route planning, since the underlying graphs are typically road networks in a two-dimensional space. There are several ways to define how *different* a path is from another one, each having its pros and cons.

Once we have a way to determine how different two paths are (preferably in relative terms), we can compute a path's *distinctness* in a set of paths. It is obtained by comparing the path with each of the other paths in the set and finding the least difference:

$$\delta(P_i, S) = \min_{P_j \in S,\, j \neq i} d(P_i, P_j) \,,$$

where $P_i$ is a path in $S$ and $d$ is the relative difference function of two paths. The higher the distinctness among the paths in a set, the more *diverse* the set of paths. We may also find it useful to define the term *distinctness threshold* as a distinctness value below which a path is not considered distinct enough for our purposes.

Following is a review of some of the metrics that express the difference of two paths, along with an evaluation of their utility in the solving of our problem.

### 3.2.1 Average Node Distance

According to the first metric, the distance of path $P_1$ from path $P_2$ is calculated as the average distance of a node in $P_1$ from the nearest node in $P_2$. We assume that each node in the road network is defined by coordinates, whether it be two simple coordinates in a plane or geographic coordinates defining the latitude, the longitude and possibly the elevation. Hence, there is a straightforward way to compute the distance $d(u, v)$ between nodes $u$ and $v$, regardless of whether the two nodes are connected by an edge with a distance cost in the graph or not. In the planar case, the Euclidean distance is sufficient:

$$d_{\mathrm{E}}(u, v) = \sqrt{(v_1 - u_1)^2 + (v_2 - u_2)^2}\,,$$

where $u_i$ is the $i$-th coordinate of node $u$.

If the nodes are defined in terms of latitude and longitude, and we insist on a more exact[1] geographical distance, then we can use an approximation by calculating the *great-circle distance*. Using an abstraction of the Earth's surface to a sphere, it is measured as the shortest distance between $u$ and $v$ along the surface of a sphere:

$$d_{\mathrm{GC}}(u, v) = r \cdot \arccos\left(\sin\,\phi_u \cdot \sin\,\phi_v + \cos\,\phi_u \cdot \cos\,\phi_v \cdot \cos\left(\Delta\lambda\right)\right)\,,$$

where $r$ is the Earth's radius, $\phi_u$ and $\lambda_u$ are the latitude and the longitude of $u$, and $\Delta\lambda$ denotes the absolute difference of the longitudes. Considering that the Earth's shape is almost spherical, this approximation's error is no greater than 0.5% [1]. Most of the route planning is performed on a significantly smaller scale anyway, and the error becomes negligible. There are several alternative representations[2] of the above formula more suitable for use in computer programs because they mitigate the consequences of rounding errors caused by the limited floating-point precision of computer systems, especially for small distances.

Having a definition of the distance function between a pair of nodes, the average node distance of path $P_1$ from path $P_2$ is:

$$d_{\mathrm{avg}}(P_1, P_2) = \frac{1}{|V_{P_1}|} \cdot \sum_{u \in V_{P_1}} \min_{v \in V_{P_2}} d(u, v)\,,$$

where $V_{P_i}$ is the set of nodes contained in path $P_i$ and $d(u, v)$ is an arbitrary node distance function, such as $d_{\mathrm{E}}$ or $d_{\mathrm{GC}}$ defined above. The problem with this metric is that it lacks the symmetry property, i.e. $d(P_1, P_2)$ does not necessarily equal $d(P_2, P_1)$. Figure 3.1 depicts several different scenarios that can occur when determining the nearest node. As the nodes $u$ and $a$ illustrate, the relationship can be mutual, i.e. $a$ is the nearest node to $u$ and, at the same time, $u$ is the nearest node to $a$. A different scenario, involving nodes $c$, $v$ and $w$ in the example, shows that a node can be the nearest node to another node (as $c$ is to $v$), but the nearest node to the former is a different one (node $w$). And then there is the case of shared nodes (such as $x$ is shared between $P_1$ and $P_2$), where a node is the nearest node to itself, with a distance equal to zero.

[1] Note that an exact distance is virtually unattainable due to the surface irregularities between any two points on the Earth's surface. Therefore, the computation is often based on abstraction to a certain degree, typically not accounting for the elevation at all.

[2] https://en.wikipedia.org/wiki/Great-circle_distance

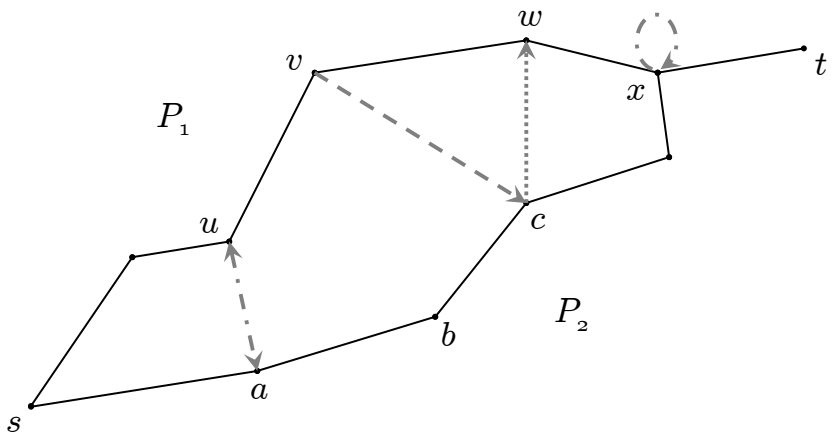



Figure 3.1: Examples of the nearest nodes and the (a)symmetry of this relationship between two paths. The characteristic of being the nearest node is depicted by an arc with an arrow, wherein the arrow points at the nearest node with respect to the node it springs from. If the arc has arrows at both ends, the two nodes are the nearest to each other.

In some cases, the asymmetry might not be an issue, but otherwise, the two average node distances can be averaged in order to obtain the same distance value:

$$d_{\mathrm{AN}}(P_1, P_2) = d_{\mathrm{AN}}(P_2, P_1) = \frac{d_{\mathrm{avg}}(P_1, P_2) + d_{\mathrm{avg}}(P_2, P_1)}{2}\,.$$

Using the average node distance for the assessment of path difference has two major drawbacks. First, a lot of calculations need to be performed to obtain the difference of two paths. This function would have a huge performance impact if called repeatedly. Second, the difference is *absolute* and, therefore, we would require a new scale to determine the degree of the difference for every problem instance depending on its size. Simply put, an average node distance of 100 meters between two routes might be significant in an urban environment but not quite enough when planning long-distance trips.

### 3.2.2 Jaccard Distance

A different metric relies completely on the nodes as entities, without considering their coordinates. The *Jaccard similarity coefficient* [24] of two paths is determined by comparing the paths as sets of nodes they are composed of. The coefficient is defined as the ratio of the number of nodes in the intersection and the number of nodes in the union of the two node sets:

$$J(V_{P_1}, V_{P_2}) = \frac{|V_{P_1} \cap V_{P_2}|}{|V_{P_1} \cup V_{P_2}|}\,,$$

where $V_{P_i}$ is the set of nodes contained in path $P_i$. The Jaccard similarity coefficient can range from 0 to 1, where 0 means the two sets are disjoint and 1 indicates they are identical. This, of course, does not necessarily imply the paths are the same, for the sequence of the

nodes may be different in each of the paths. A simple remedy for that is to use the paths' sets of edges instead in the calculation of the Jaccard similarity index:

$$J(E_{P_1}, E_{P_2}) = \frac{|E_{P_1} \cap E_{P_2}|}{|E_{P_1} \cup E_{P_2}|},$$

where $E_{P_i}$ is the set of edges contained in path $P_i$. If the edges in the graph are directed, then the equality of two edge sets guarantees the equality of the paths they constitute, as long as the paths start in the same node.

What the coefficient expresses is the degree of similarity of two sets. Since we are interested in how dissimilar two paths are, we use the following definition of the *Jaccard distance* [24], which is a complement to the Jaccard similarity coefficient. At the same time, we alter the metric's arguments so as to express the dissimilarity of two paths, not of two edge sets:

$$d_{\mathrm{J}}(P_1, P_2) = 1 - J(E_{P_1}, E_{P_2}) = \frac{|E_{P_1} \cup E_{P_2}| - |E_{P_1} \cap E_{P_2}|}{|E_{P_1} \cup E_{P_2}|}.$$

The alternative interpretation in the above formula enables us to count the edges that are present in only one of the paths (which corresponds to the symmetric difference of the two edge sets) and divide the count by the size of the sets' union in order to obtain the Jaccard distance between the two paths.

The primary advantage of this metric over the previous one is that it indicates the dissimilarity of the paths on a fixed scale (between 0 and 1) no matter what length of the paths or how many edges they contain. Unlike in the case of average node distance, where we do not know the upper bound of the total path difference, using the Jaccard distance the bound is always 1. Hence, the *relative* difference obtained by this metric allows us to decide about the distinctness threshold in advance and regardless of the problem instance we are solving.

What could be perceived as a weakness is that the Jaccard distance neglects the spatial distance of the paths. As a result, a nearby, so to speak, parallel path could be indicated as more different than one generally farther away but sharing a slightly greater portion. However, an even bigger issue is that the Jaccard distance completely ignores the lengths of the diverging sections. One of the consequences is that a long path section with sparse nodes could contribute to the path's difference considerably less than a short one with a higher density of nodes, which, in road networks, typically corresponds to the contrast between rural highways and streets in city downtowns or residential areas with frequent junctions. Figure 3.2 will help us illustrate this on an example. Let us assume there are three Pareto optimal paths from $s$ to $t$: $P_{a,d}$, $P_{b,c}$ and $P_{b,d}$, each going through the nodes indicated in its subscript. Using the Jaccard distance to compare the first two paths with the third one, we discover that $P_{a,d}$ is more different from $P_{b,d}$ than $P_{b,c}$ is. There are eight nodes in the symmetric difference of the former pair of paths, compared to only five in the latter. But clearly the section between the nodes $x$ and $y$ is less significant than that between $y$ and $z$, so declaring $P_{a,d}$ a more different path than $P_{b,c}$ is arguably an undesired result. This renders the Jaccard distance metric rather unsuitable for the computation of path distinctness in a road network with all kinds of roads and streets.

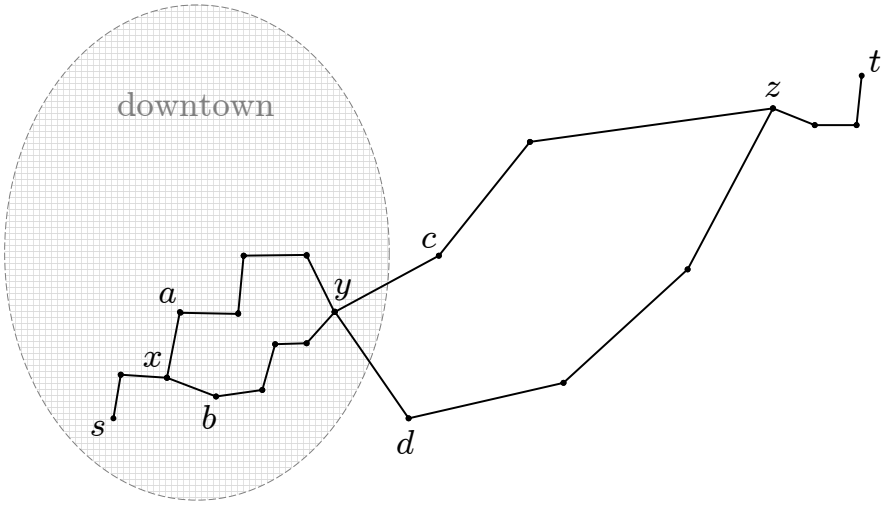



Figure 3.2: The disparity of node densities in different parts of the road network can cause a path to be unreasonably evaluated as very different from another path when using the Jaccard distance.

### 3.2.3 Cumulative Edge Length Difference

Keeping the deficiencies of the previous metric in mind, we can improve on it by accounting for the edge length. The computation of the difference of a path $P_1$ from $P_2$ then amounts to calculating the cumulative length of all edges present exclusively in $P_1$ or $P_2$ and, subsequently, the ratio of this length to the total length of the edges of $P_1$ and $P_2$ combined:

$$d_{\mathrm{CEL}}(P_1, P_2) = \frac{\sum_{e \in (E_{P_1} \cup E_{P_2}) \setminus (E_{P_1} \cap E_{P_2})} l(e)}{\sum_{e \in E_{P_1} \cup E_{P_2}} l(e)},$$

where $E_{P_i}$ is the set of edges contained in path $P_i$ and $l(e)$ denotes the length of edge $e$. As a matter of fact, this is an extension of the Jaccard distance with edges weighted by their lengths. When calculating the Jaccard distance, each edge not shared between the paths contributes to the path's difference with the same value, whereas here an edge contributes with a fraction that corresponds to its relative length to the paths' lengths combined.

This modification makes the metric based on the cumulative edge length a great deal more versatile. Problem instances in which the origin is in the city downtown and the destination is, say, a cabin in the woods fifty kilometers out (such as the exmaple in Figure 3.2) are no longer a complication. Even the other flaw present in the Jaccard distance metric, concerning the preference of nearby paths, is positively affected by the edge weighting. It is still possible that the area (or the sum of areas) enclosed with paths $P_1$ and $P_2$ is smaller than that enclosed with $P_1$ and $P_3$, while $P_2$ is evaluated as more different from $P_1$ than $P_3$ is (see Figure 3.3 for an example). However, supposing the length is the principal measure of difference, this is no issue at all. Calculating the size of the area instead of summing the edge lengths, which could be a separate metric in itself, would considerably increase the complexity after all. It is difficult to argue which of these two approaches would lead to "more different" results, and it may be simply a matter preference.

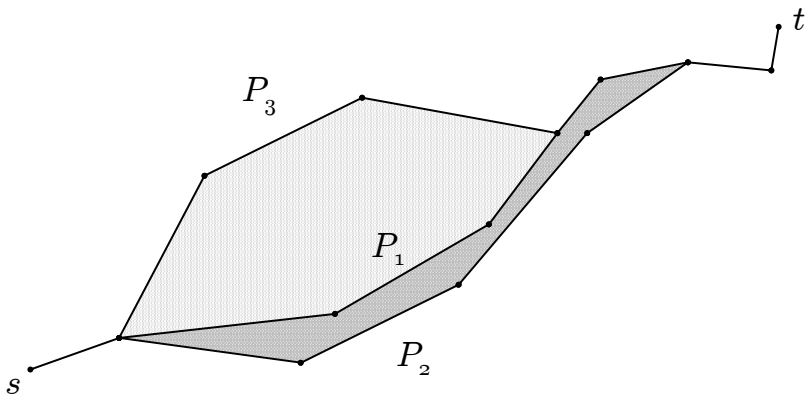



Figure 3.3: An example of the contrast between the cumulative edge length difference and the enclosed area difference.

### Incremental Path Update

The two relative difference metrics mentioned up to this point have a shortcoming that manifests when we try to compute the difference of a path $P_1$ from $P_2$ *incrementally*, i.e. edge by edge without the recalculation of the contribution of the already processed part of $P_1$. During each increment before the final one, we only have a *path segment* – a term we use for a path from the origin up to a node, but not including the destination node yet (in other words, a sub-path of a solution path) – at our disposal whose distinctness we are interested in. At the moment of determining the difference of a path segment $P_{1\text{seg}}$ from $P_2$, neither the length nor the total number of edges of the future solution path $P_1$ is known. Without that information and using, for instance, the length of the current path segment instead, we would not have a constant denominator in the difference formula. That would result in a different relative contribution of the edges in each increment (the contribution would gradually decrease with the increasing length of the path segment) and the edge contributions would eventually not add up to the same value as if the difference was calculated for the entire $P_1$ at once.

It is clear that, in order to be able to utilize a relative difference metric in a scenario with an incremental update, it is necessary to fix the denominator so the relative difference contribution of every edge is calculated with respect to the same whole. To achieve that for the current metric, we can replace the union of $E_{P_1}$ and $E_{P_2}$ in the denominator by $E_{P_2}$ only, which remains constant throughout the whole process. However, we then need to adjust the numerator in the formula for the change in the denominator and only count in the edges that are present in $P_2$. If we did include edges from $P_1$, the metric could eventually take on values greater than 1 and its upper bound would be unknown, nullifying thus the benefit provided by the bounded range for setting a universal distinctness threshold. Following these notions,

the formula reduces to:

$$d_{\mathrm{CEL}_{\mathrm{red}}}(P_{\mathrm{1seg}}, P_2) = \frac{\sum_{e \in E_{P_2} \setminus E_{P_{\mathrm{1seg}}}} l(e)}{\sum_{e \in E_{P_2}} l(e)} = 1 - \frac{\sum_{e \in E_{P_2} \cap E_{P_{\mathrm{1seg}}}} l(e)}{\sum_{e \in E_{P_2}} l(e)}\,.$$

After the modification, the metric behaves more like an estimation, but it does not fundamentally change its usability for our purposes. Although it is no longer symmetric, it still takes on values between 0 and 1, where 0 conveys the paths' equality and 1 indicates there is no overlap. Moreover, it gives us the choice between using the portion of $P_2$ that is different from $P_{\mathrm{1seg}}$ or the portion that is shared (which, in the formula, is expressed by the two alternative interpretations respectively), the latter being more practical because, during the incremental update, each edge in $P_1$ gets compared with the edges in $P_2$ and is confirmed as shared or not shared.

An important characteristic of the above formula is that it assumes a total difference of the yet unknown portion of $P_1$ from $P_2$. Hence, the difference of $P_1$'s segments from $P_2$ monotonically decreases from 1 toward 0 during the incremental update. A threshold can be employed here to stop the update as soon as the difference of a path segment drops below a certain value in case we would like to discard paths with a small difference. Now, when $P_{\mathrm{1seg}}$ is extended by edge $e_{\mathrm{new}}$, the difference of the extended path $P_{\mathrm{1seg}'}$ from $P_2$ can be computed with the help of the previous segment's difference as follows:

$$d_{\mathrm{CEL}_{\mathrm{red}}}(P_{\mathrm{1seg}'}, P_2) = d_{\mathrm{CEL}_{\mathrm{red}}}(P_{\mathrm{1seg}}, P_2) - \frac{l(e_{\mathrm{new}})}{\sum_{e \in E_{P_2}} l(e)}$$

if $e_{\mathrm{new}} \in E_{P_2}$, otherwise $d_{\mathrm{CEL}_{\mathrm{red}}}(P_{\mathrm{1seg}'}, P_2)$ equals $d_{\mathrm{CEL}_{\mathrm{red}}}(P_{\mathrm{1seg}}, P_2)$. The number of shared edges cannot exceed the number of edges in $P_2$, which ensures that the value of $d_{\mathrm{CEL}_{\mathrm{red}}}$ never drops below zero. Note, that the equation would not hold true using the original $d_{\mathrm{CEL}}$ in place of $d_{\mathrm{CEL}_{\mathrm{red}}}$.

The Jaccard distance metric could be modified analogically and adapted to the iterative updating of paths. However, being inferior to the cumulative edge length difference metric, there is no use in elaborating on the Jaccard distance.

### 3.2.4 Cumulative Edge Length Difference with Projection

We also consider a different approach to ensuring the metric's fitness for the incremental path update, with the further benefit of not giving up the symmetry. It is based on the cumulative edge length difference, similar to the previous metric, but the edges are projected onto a straight line between the origin and the destination node before their lengths are used in the calculations of the relative path difference. The projection enables us to compare the relative length of an edge with respect to the length of the virtual origin-destination line, which remains constant throughout the whole process of the difference evaluation. This again gives us the option to assume the remainder of a path does not overlap with the path being compared to when working with path segments.

In order to calculate the length of the projection of edge $e$, we only need to know the coordinates of its endpoints $u$ and $v$, and the coordinates of the origin node $s$ and the destination node $t$. Having these values, we can determine the angle $\phi$ between $e$ and the

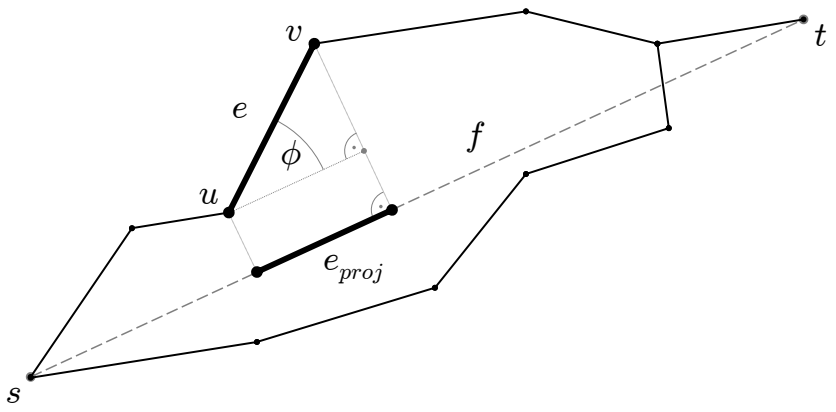



Figure 3.4: Projection of a path's edge onto the origin-destination line.

origin-destination line $f$ (see Figure 3.4). When we regard the edge $e$ as a vector defined by the nodes $u$ and $v$, and $f$ as a vector defined by $s$ and $t$, the cosine of the angle is:

$$\cos\,\phi = \frac{\vec{e}\cdot\vec{f}}{\|\vec{e}\|\cdot\|\vec{f}\|}\,,$$

where $\vec{e} = (v_1 - u_1, v_2 - u_2)$, $\vec{f} = (t_1 - s_1, t_2 - s_2)$, and $\|\vec{e}\|$ and $\|\vec{f}\|$ are their Euclidean lengths. The coordinates of the nodes are denoted using subscripts, so, for instance, $u_1$ and $u_2$ are the first and the second coordinate of node $u$. For the sake of simplicity, we perform the computations in Euclidean plane.

The cosine of $\phi$ can then be substituted in the following formula to calculate the length of the projected edge $e_{\text{proj}}$:

$$l(e_{\text{proj}}) = \|\vec{e}_{\text{proj}}\| = \|\vec{e}\|\cdot\cos\,\phi = \frac{\vec{e}\cdot\vec{f}}{\|\vec{f}\|} = \frac{(v_1-u_1)\cdot(t_1-s_1)+(v_2-u_2)\cdot(t_2-s_2)}{\sqrt{(t_1-s_1)^2+(t_2-s_2)^2}}\,.$$

The dot product of vectors $\vec{e}$ and $\vec{f}$ is written out as the sum of the products of their corresponding components. The length of the origin-destination span can be precomputed at the beginning and used as a constant throughout the entire incremental update, and so can be the differences of the origin's and the destination's coordinates.

The difference of a path segment from a path is then computed in a similar way as in the $d_{CEL_{red}}$ metric, but it operates with projected edges. One of the consequences is that the denominator reduces to the flying distance from the origin to the destination:

$$d_{\text{CEL}_{\text{proj}}}(P_{1\text{seg}}, P_2) = \frac{\sum_{e\,\in\,E_{P_2}\setminus E_{P_{1\text{seg}}}} l(e_{\text{proj}})}{\sqrt{(t_1-s_1)^2+(t_2-s_2)^2}} = 1 - \frac{\sum_{e\,\in\,E_{P_2}\cap E_{P_{1\text{seg}}}} l(e_{\text{proj}})}{\sqrt{(t_1-s_1)^2+(t_2-s_2)^2}}\,,$$

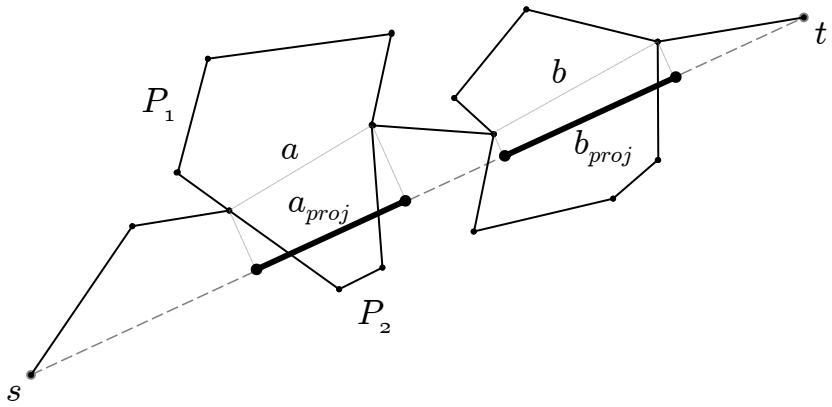



Figure 3.5: Interpretation of the edge projection metric. The relative difference corresponds to the length ratio of $a_{proj} + b_{proj}$ to the origin-destination ($s$-$t$) span.

where $l(e_{\text{proj}})$ is the length of $e$'s projection. Extending a path segment by edge $e$ then works analogically too – with the simplified denominator and using the edge's projection:

$$d_{\text{CEL}_{\text{proj}}}(P_{1\text{seg}'}, P_2) = d_{\text{CEL}_{\text{proj}}}(P_{1\text{seg}}, P_2) - \frac{l(e_{\text{proj}})}{\sqrt{(t_1 - s_1)^2 + (t_2 - s_2)^2}}\,.$$

The projection handles the cases in which the edge goes "away" from the destination (that is, when $\phi$ is greater than 90° and smaller than 270°) in a rather undesired way though. The cosine of such an angle is negative, which is reflected by the dot product, and, therefore, the projected edge's length becomes negative as well. This results in a negative contribution to the relative difference of a path segment from the origin-destination line. An example of such an edge is the third edge of path $P_1$ in Figure 3.5. The negative contribution of this edge essentially lessens the positive contribution of the fourth edge.

In the end, what the difference computed by this metric amounts to is the cumulative flying distance between pairs of nodes at the ends of the path's sections that are not shared with the other path, projected onto the virtual origin-destination line. The example in Figure 3.5 illustrates this – the difference of path $P_1$ from $P_2$, and vice versa, correlates with the cumulative length of the line segments $a_{proj}$ and $b_{proj}$. The shape of the different sections clearly does not affect the eventual difference. What matters is the distance of the points where the paths split up and rejoin. This is a considerable weakness when compared to the previous two metrics because the physical length of a different section has only a limited impact on the path's difference.

### 3.2.5 Metric Review

So far in this section, we have presented five metrics for assessing how different a path is from another. Each of them has its pros and cons, as summarized in Table 3.1. While $d_{\text{AN}}$ is

Table 3.1: Overview of the path difference metrics' properties.

| | $d_{\text{AN}}$ | $d_{\text{J}}$ | $d_{\text{CEL}}$ | $d_{\text{CEL}_{\text{red}}}$ | $d_{\text{CEL}_{\text{proj}}}$ |
|---|---|---|---|---|---|
| **Considers spatial distance** | • | – | – | – | – |
| **Considers physical length** | – | – | • | • | ∼ |
| **Symmetric** | • | • | • | – | • |
| **Relative difference (→ allows global setting of a distinctness threshold)** | – | • | • | • | • |
| **Suitable for incremental path update** | ∼ | – | – | • | • |
| **Fast execution** | – | • | • | • | ∼ |

the only one that considers the spatial distance of the paths, it lacks in other departments, such as the performance and the suitability for incremental path update. Although the incremental update is possible, its utilization is hindered in practice by the difference being absolute, as opposed to relative, which complicates the setting of a universal distinctness threshold for a set of paths.

On the other hand, the only two metrics to properly consider the physical length of the paths' diverging sections are $d_{\text{CEL}}$ and $d_{\text{CEL}_{\text{red}}}$. While the former is symmetric and, so to speak, more accurate, the latter is ready for use in the incremental update of paths' differences. The main purpose of the incremental update is that it enables the computation of a path's difference from multiple paths in parallel, edge by edge. This is essential in order to efficiently update the distinctness of a path in a set of paths when it is extended by an edge, which is a typical operation in the multi-criteria shortest path search during the label-expansion phase. The adoption of a path difference metric in the multi-criteria search is discussed in Chapter 4.

Finally, we devised an alternative metric usable incrementally as well, but preserving the symmetry. Unlike $d_{\text{CEL}_{\text{red}}}$, $d_{\text{CEL}_{\text{proj}}}$ can be used when the difference of a path $P_1$ from $P_2$ is required to equal the difference of $P_2$ from $P_1$. That, however, is achieved at the expense of the metric's performance (there is a significant increase of calculations that need to be carried out in the process of projection) and also its accuracy (the length of the diverging sections affects the path's relative difference to a lesser extent).

# Chapter 4

# Solution Method

In this chapter, we first formally describe the NAMOA* algorithm for solving the multi-criteria shortest path problem. We present a few selected heuristics that estimate the path's costs and help thus the multi-criteria search to find the solutions faster. Afterwards, we discuss the diversification options in the process of finding non-dominated solutions. Finally, we introduce our diversity-aware extension of the NAMOA* algorithm, which embeds the diversification directly into the search.

## 4.1 NAMOA* Algorithm

The NAMOA* algorithm, first introduced in [29] in 2005, is a successful extension of the A* algorithm for solving the shortest path problem with multiple criteria. The algorithm was originally designed to work with multiple destinations, i.e. to solve the one-to-many shortest path problem. However, considering that typical queries for shortest paths in a road network are defined by a single origin and a single destination, we present a point-to-point modification of the algorithm.

NAMOA* employs a heuristic lower bound function $h(u)$ to estimate the cost of the path from a node $u$ to the destination node $t$. During the run of the algorithm, a node can be assigned one or more labels. In NAMOA*, a label $L$ is defined as $(u, \vec{g}(P_{su}), \vec{f}(P_{su}), L_{\text{pred}})$, and it represents a unique path $P_{su}$ from the origin $s$ to the node $u$ (note that there may be more than one path between these two nodes). Let $\vec{g}(P_{su})$ be the cost vector of a particular path $P_{su}$, $\vec{f}(P_{su}) = \vec{g}(P_{su}) + \vec{h}(u)$ the *evaluation vector*, and $L_{\text{pred}}$ the label of $L$'s predecessor. The vector $\vec{h}(u)$ contains the lower bound of the cost of any path from $u$ to $t$ provided by the function $h(u)$. To simplify the label notation, we will omit $P_{su}$ from the label description and use $(u, \vec{g_u}, \vec{f_u}, L_{\text{pred}})$ instead, as we are only interested in paths starting in $s$. Label $L_u$ dominates label $L_v$ if for their respective evaluation vectors $\vec{f_u}$ and $\vec{f_v}$ holds that $\vec{f_u} \prec \vec{f_v}$.

Being a label-setting algorithm with the label-selection strategy, NAMOA* handles labels individually. It maintains a priority queue $Q$ of all currently *open* labels. When a label is open, it means it has been created by expanding another open label that was subsequently *closed*. Initially, $Q$ contains a single label $(s, \vec{0}, \vec{h}(s), \texttt{null})$, assigned to the origin node, which has no predecessor and whose evaluation vector equals the lower bounds of each criterion for a path from the origin to the destination. In addition to the queue of all open labels, each

node $u$ has its own sets $G_{\text{op}}(u)$ of open labels and $G_{\text{cl}}(u)$ of closed (i.e. already expanded) labels belonging to the node. Finally, we will use a set $C$ that stores the labels corresponding to the non-dominated solution paths, which means that $C$ is equivalent to $G_{\text{cl}}(t)$.

The algorithm, whose pseudocode is outlined in Algorithm 1, iteratively selects labels offered by the priority queue $Q$ for expansion. In each iteration, it removes a non-dominated label $L_u = (u, \vec{g_u}, \vec{f_u}, L_{\text{pred}})$ from $Q$ and extends it along all edges $(u, v)$, creating a new label $L_v$ with a cost vector $\vec{g_v} = \vec{g_u} + \vec{c}(u, v)$ and an evaluation vector $\vec{f_v} = \vec{g_v} + \vec{h}(v)$ for each $v$. $L_u$ is then moved from $G_{\text{op}}(u)$ to $G_{\text{cl}}(u)$. Each $L_v$, before being added to $G_{\text{op}}(v)$ and $Q$, must be tested for dominance in the following two ways:

1. If there is an existing label in $G_{\text{op}}(v)$ or $G_{\text{cl}}(v)$ dominating $L_v$, then $L_v$ is discarded because an extension of $L_v$ would always be dominated (due to the optimality principle). For the same reason, if there is an existing label in $G_{\text{op}}(v)$ or $G_{\text{cl}}(v)$ dominated by $L_v$, the label is removed from the respective set and from $Q$ as well. This operation is called *pruning*.

2. If there is a label in $C$ that dominates $L_v$, then $L_v$ can never become a non-dominated solution and, therefore, it can safely be discarded. We refer to this as *filtering*.

The iteration continues until $Q$ is empty. The algorithm returns the set $C$, which, at the end of the search, contains the labels corresponding to all non-dominated solution paths. It is then possible to reconstruct all the solution paths from the labels in $C$ recursively using the references to their predecessors.

Much like its single-criterion ancestor A*, NAMOA* is guaranteed to reach the correct result, in this case the complete set of Pareto optimal paths to the destination, as long as the heuristic function $h(u)$ returns a lower bound of the cost of any path from $u$ to the destination node $t$. Furthermore, if $h(u)$ is a consistent heuristic, NAMOA* expands non-dominated labels only and is optimal among the class of exact best-first algorithms [28].

## 4.2 Cost Estimating Heuristics

Heuristic functions are used in search algorithms to estimate the costs of paths. Based on these estimates, the search can be directed toward the destination node $t$, avoiding thus the unnecessary exploration of remote parts of the search space that are certainly not visited by the shortest paths. All the heuristic functions $\vec{h}(u)$ presented in this section return a single vector of cost estimates for the path from $u$ to $t$.

In general, the better informed the chosen heuristic, the more efficient the search. In a multi-criteria search, heuristic $\vec{h_1}$ is said to be *at least as informed* as $\vec{h_2}$ when $\vec{h_2}(u) \preceq \vec{h_1}(u)$ for each node $u \in V$ and, at the same time, $\vec{h_1}$ is admissible. For a heuristic $\vec{h}$ to be *admissible*, it must satisfy the following condition:

$$\vec{g}(P_{su}) + \vec{h}(u) \preceq \vec{g}(P_{st})$$

for each non-dominated solution path $P_{st} = (s, \ldots, u, \ldots, t)$ and each of its sub-paths $P_{su} = (s, \ldots, u)$. In other words, an admissible heuristic never overestimates the real costs and,

**Algorithm 1:** NAMOA*

**Input :** graph $G(V, E, \vec{c})$, origin node $s \in V$, destination node $t \in V$, heuristic function $h$
**Output:** set of all non-dominated solution paths

`/* Initialization */`
$L_s \leftarrow (s, \vec{0}, \vec{h}(s), \texttt{null})$ `// initial label of the origin node` $s$
$G_{\text{op}}(s) \leftarrow \{L_s\},\ G_{\text{cl}}(s) \leftarrow \emptyset$ `// sets of open and closed labels of` $s$
$Q \leftarrow \{L_s\},\ C \leftarrow \emptyset$ `// priority queue and solution set`
**while** $Q$ is not empty **do**
  `/* Label selection */`
  select a non-dominated label $L_u = (u, \vec{g_u}, \vec{f_u}, L_{u\text{Pred}})$ from $Q$
  move $L_u$ from $G_{\text{op}}(u)$ to $G_{\text{cl}}(u)$, and remove $L_u$ from $Q$
  `/* Solution label recording */`
  **if** $u = t$ **then**
    add $L_u$ to $C$
    **foreach** $L_x = (x, \vec{g_x}, \vec{f_x}, L_{x\text{Pred}}) \in Q \mid \vec{g_u} \prec \vec{g_x}$ **do**
      remove $L_x$ from $Q$ `// filtering`
    **continue** with the next label selection
  `/* Label expansion */`
  **forall** successor nodes $v$ of $u$ that do not create a cycle in the path **do**
    $\vec{g_v} \leftarrow \vec{g_u} + \vec{c}(u, v)$
    $\vec{f_v} \leftarrow \vec{g_v} + \vec{h}(v)$
    $L_v \leftarrow (v, \vec{g_v}, \vec{f_v}, L_u)$
    **if** $\exists (x, \vec{g_x}, \vec{f_x}, L_{x\text{Pred}}) \in C \mid \vec{g_x} \prec \vec{f_v}$ **then**
      discard $L_v$ `// filtering`
    **else if** $v$ is a new node, i.e. a node with no labels so far **then**
      $G_{\text{op}}(v) \leftarrow \{L_v\},\ G_{\text{cl}}(v) \leftarrow \emptyset$
      add $L_v$ to $Q$
    **else if** $\exists (x, \vec{g_x}, \vec{f_x}, L_{x\text{Pred}}) \in G_{\text{op}}(v) \cup G_{\text{cl}}(v) \mid \vec{g_x} \prec \vec{g_v}$ **then**
      discard $L_v$ `// pruning`
    **else**
      **foreach** $L_y = (y, \vec{g_y}, \vec{f_y}, L_{y\text{Pred}}) \in G_{\text{op}}(v) \mid \vec{g_v} \prec \vec{g_y}$ **do**
        remove $L_y$ from $G_{\text{op}}(v)$ and from $Q$ `// pruning`
      **foreach** $L_y = (y, \vec{g_y}, \vec{f_y}, L_{y\text{Pred}}) \in G_{\text{cl}}(v) \mid \vec{g_v} \prec \vec{g_y}$ **do**
        remove $L_y$ from $G_{\text{cl}}(v)$ `// pruning`
      add $L_v$ to $G_{\text{op}}(v)$ and to $Q$
**return** $C$

therefore, its estimates may be considered as lower bounds of the path's costs. A multi-criteria search with an admissible heuristic function is guaranteed to find the full Pareto set of solutions [28].

Another important property of heuristics is consistency. A multi-criteria heuristic function $\vec{h}$ is *consistent* if:

$$\vec{h}(u) \preceq \vec{c}(u,v) + \vec{h}(v) \quad \forall (u,v) \in E\,.$$

When a heuristic function is consistent, it is also admissible, but the opposite does not have to be true [40]. The consistency of the heuristic used is essential for the efficiency of NAMOA* because it guarantees that the pruning of paths during the search will be optimal [28].

In the remainder of this section, we will briefly introduce a few consistent heuristics often used when solving the multi-criteria shortest path problem in road networks. Note that the basic heuristic $\vec{h_0}(u) = (0, \ldots, 0)$ corresponds to a *blind search*.

### Flying Distance Heuristics

In planar graphs, the most straightforward heuristic uses the *Euclidean distance*, which is the flying distance, between two nodes to estimate their real distance. The time estimates could be calculated by dividing the distance by the maximum speed, which we know will not be exceeded under any circumstances, but such simple estimates are obviously rather poor.

In case we have a problem where the nodes' coordinates are given as their latitude and longitude, we can use the *great-circle distance* heuristic instead to calculate the distance between two nodes. It is still the flying distance, only it takes the curvature of the Earth's surface into account. The time estimate computation is then analogical to that of the Euclidean distance heuristic.

However, neither of these heuristics is remotely accurate, and they may prove incapable of producing useful estimates for some types of criteria, such as the road incline or the number of signal-controlled intersections. Their major asset is that they are simple and thus fast enough to be used in real time during a search.

### TC Heuristic

A different approach, proposed by Tung and Chew in [41] – the *TC heuristic* – searches for the single-criterion shortest paths from the destination node to all other nodes for each individual criterion, ignoring the other criteria. Note that the shortest path from the destination to an arbitrary node in this reversed search corresponds to the actual shortest path from that node to the destination.

This can be easily achieved by applying Dijkstra's algorithm on the graph with reversed edges. For two criteria the heuristic is defined as $\vec{h}(u) = \big(\, c_1^*(u), c_2^*(u) \,\big)$, where $c_i^*(u)$ denotes the optimal cost of a path from $u$ to the destination node considering the $i$-th criterion only. This heuristic is considerably more informed than the previous two, in fact it is optimal, but it is a fairly costly procedure. For this reason, it must be performed in a preprocessing phase of the search, and it has to be done so for each new instance of the problem, since the estimates are precomputed with a particular destination node in mind.

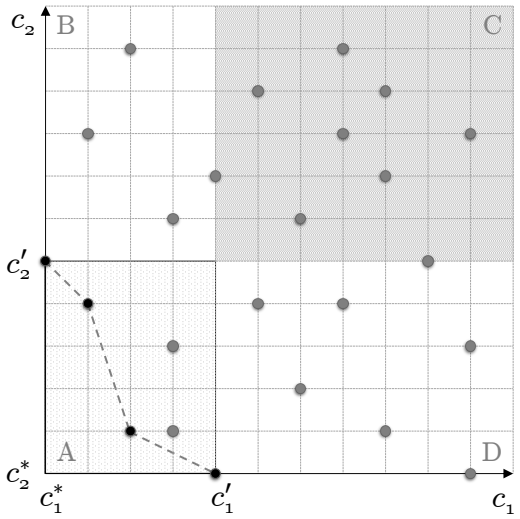



Figure 4.1: An example of the Pareto frontier in the cost space of a problem with two criteria. It is delimited by the ideal point $(c_1^*, c_2^*)$ and the nadir point $(c'_1, c'_2)$.

#### Bounded TC Heuristic

As we mentioned in the introduction to this section, heuristic functions are employed to reduce the area of the search space that gets explored. Apparently, not all nodes are necessarily visited during the search and, therefore, we do not have to precompute the estimates for all of them. That is the idea behind the *bounded TC heuristic*.

It has been shown that during the search in NAMOA*, no path will be expanded whose evaluation vector is dominated by a cost vector of a non-dominated solution path [28]. It is thanks to this property of the algorithm that we can limit the number of nodes for which we calculate the estimates. Let us consider the optimal costs $c_1^*$ and $c_2^*$ of a path from the origin to the destination for the individual criteria. There is a certain minimum value of the second criterion among all the solution paths having the first criterion equal to $c_1^*$. We denote this value by $c'_2$. Analogically, $c'_1$ is the minimum value of the first criterion among the solution paths with the second criterion equal to $c_2^*$. The two extreme points $(c_1^*, c'_2)$ and $(c_2^*, c'_1)$ define the boundaries within which we can find the costs of all the Pareto optimal solution paths. NAMOA* will never expand a label whose cost is dominated by the vector $(c'_1, c'_2)$, as such a label could never lead to a non-dominated solution. Hence, it is the nodes with the optimal costs $c_1^*(u) > c'_1$ and $c_2^*(u) > c'_2$ at the same time (in Figure 4.1 they would correspond to those in the shaded area marked with a C) that we do not need to calculate the heuristic estimates for. This substantially reduces the number of computations, especially in problem instances with a short origin-destination distance.

## 4.3 Diversification of Multi-Criteria Solutions

In a multi-criteria route planning problem, we face the inconvenience of possibly having to deal with an enormous number of Pareto optimal solutions. In most cases, it is pointless to

offer hundreds of solutions, all optimal from the multi-criteria point of view, when they are mostly just various combinations of different path segments and often differ only marginally from many of the other optimal solutions. Reducing the size of the solution set substantially is one of the major objectives of the route planning with an emphasis on diversity, which is the focus of our work. Unlike in other, simpler approaches, such as picking the first few solutions found, our intention is to maximize the mutual difference of the chosen solutions. In other words, we are only interested in the first couple of the most distinct solutions, which are expected to have the highest utility with regard to the variety of choices. If we compare, for instance, the solution routes in Figure 4.2a with their diverse subset in Figure 4.2b, there is no significant change to be observed in the map, besides several short segments missing in the latter that did not end up included in any of the chosen solutions. However, the number of solutions offered was reduced to as little as 3%, which is considerably more convenient.

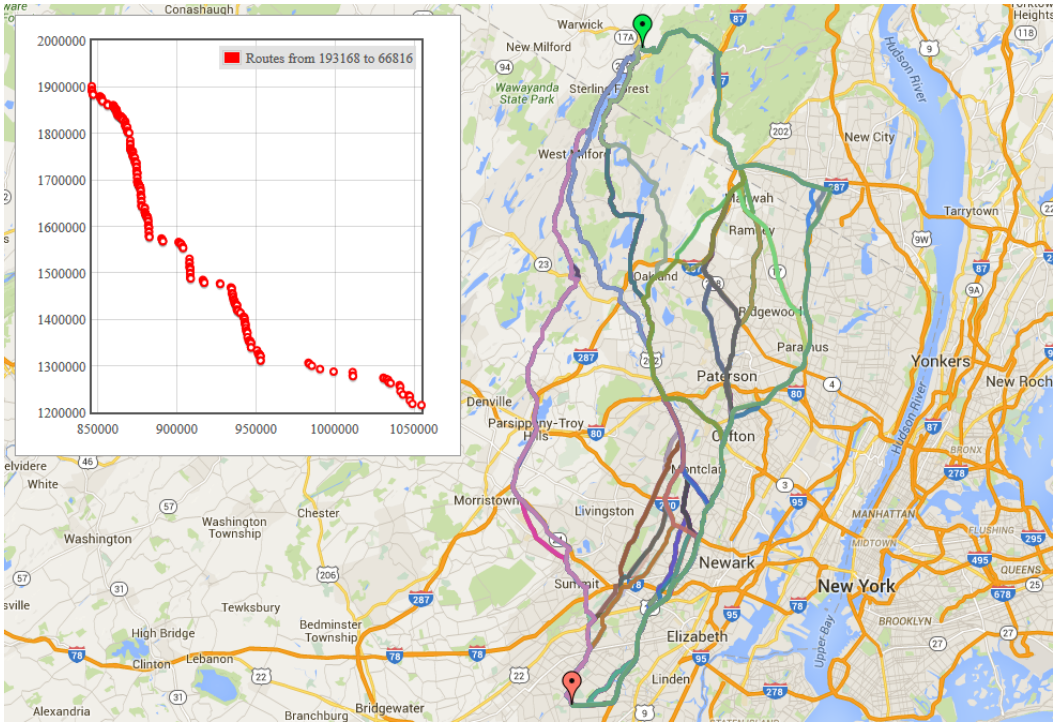
We consider two different approaches to finding a reasonably small but diverse subset of all Pareto optimal solutions. In both we include the solutions corresponding to the single-criterion optima, as, presumably, they are always of certain interest. They also serve as the initial references with which the other candidate solutions are compared. To elaborate on our notion of a "reasonably" small subset, depending on the size of the problem instance, it should range from three to ten solution paths for a bi-criteria problem, and possibly slightly more for three-and-more-criteria problems, inclusive of the default solutions (i.e. the single-criterion optima). In the following, we introduce the two diversification strategies.

### 4.3.1 Post-Processing Filter

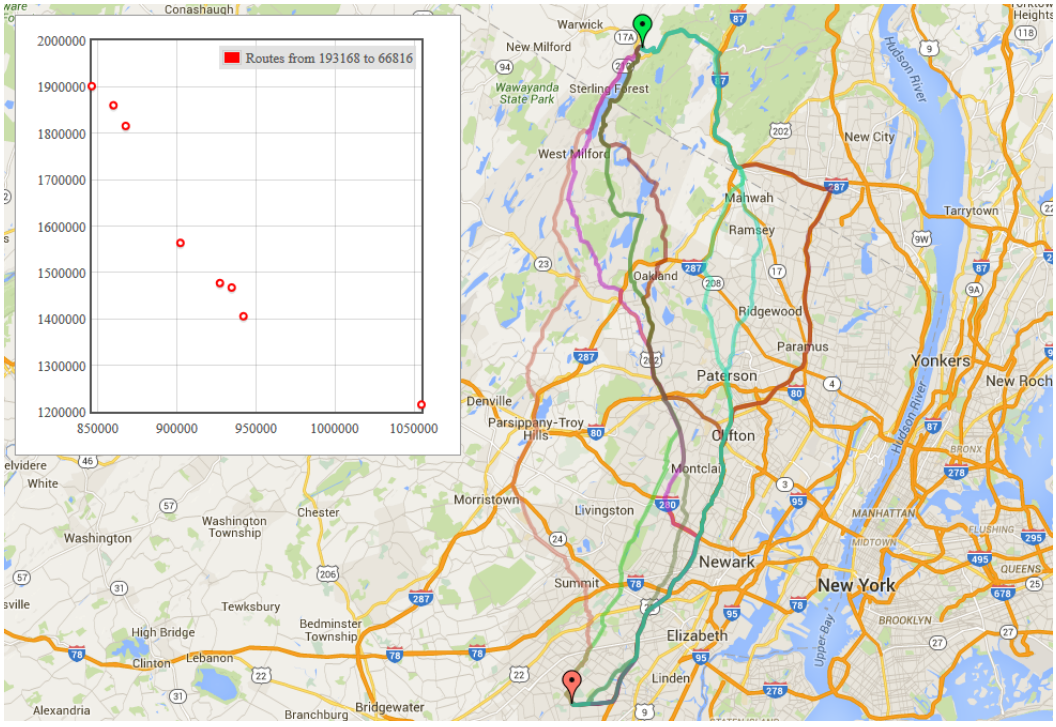
This is the simple and intuitive approach, in which all Pareto optimal solutions are found first, and only then will the most distinct among them be filtered, as a post-processing step. The main advantage here is that all the solutions are already known, and it is only a matter of choosing a method to determine which subset of the solutions is the most diverse one. The subset is delimited either by a fixed number of desired solutions or by a minimum solution distinctness accepted, i.e. a distinctness threshold.

To choose the most dissimilar paths out of the pool of paths found by NAMOA*, we use a greedy selection method. Having one or more default solution paths in the diverse set to start with, further ones are resolved iteratively by comparing the remaining paths with each of those already present in the diverse set. A path's distinctness is determined as the minimum relative difference from an already selected solution path. Therefore, the distinctness must be recalculated every time a new solution is added to the diverse set. Hence in each iteration, all remaining paths have their distinctness updated and, subsequently, the path with the greatest distinctness value is moved to the diverse set. This goes on as long as the selected solution's distinctness is above the defined threshold or until the diverse set has the desired size.

The greedy selection serves as an efficient approximation of the more comprehensive $p$-dispersion filtering, which selects from the Pareto set a subset of $p$ solution paths that maximizes the minimum difference between any pair of paths it contains. While the $p$-dispersion takes a global view of the solutions' distinctness, the greedy selection only considers a path's distinctness at the time of its addition to the gradually growing diverse subset, ignoring that




(a) The full Pareto set of 280 solutions. The Pareto frontier is plotted in the upper left-hand corner.




(b) A diverse subset of 8 Pareto optimal solutions.

Figure 4.2: An example of the solution set reduction for a bi-criteria problem.

its distinctness will continue to decline with the addition of new solutions (although it will never drop below the distinctness threshold).

### 4.3.2 Integration of Distinctness into the Search Algorithm

An alternative to filtering the Pareto set is incorporating path distinctness directly into the search algorithm. That involves the computation of the distinctness for every new label (new path) and the taking of these distinctness values into consideration during the label expansion, so as to be drawn, at any point in the search, toward the next solution that is different from those that were found up to that point. This enables us to stop the search early – before all the paths become explored – as the first few solutions found comprise the diverse subset we are looking for, and the unexplored solutions are thus those that are of no interest.

Considering the fact that the search algorithm discovers the solutions one at a time, this procedure is very similar to the filtering by greedy selection, only the selection is based on partial solutions (i.e. sub-paths of the future solution paths). Every time a new solution is found, the distinctness values of all paths need to be updated accordingly to reflect the distinctness from the most recent solution set. Only then can the search resume exploring further partial solutions.

The primary advantage of this approach, when compared to the filtering, is a reduced number of necessary label expansions and, with it, a potential performance improvement.

## 4.4 Diversity-Aware NAMOA* Algorithm

In this section, we describe how the path distinctness can be employed in the multi-criteria search algorithm NAMOA* to compel it to discover the Pareto optimal solutions in such order that stopping the search after finding only the first few solutions would yield a solution subset with a decent variety among them with regard to their geographical setting. This extension will enable the algorithm to find a small set of diverse solution paths in a straightforward way and without the need for exploring all Pareto optimal solution paths.

In the process of the search, labels, or rather the paths they represent, are compared with the paths in the solution set to have their distictness determined. The difference is evaluated using the cumulative edge length difference metric adapted to the incremental path update, as defined in 3.2.3.

### 4.4.1 Label Augmentation by Path Distinctness

Let us first delve deeper into how the integration of distinctness, introduced at the end of the previous section, affects the creation of successor labels in NAMOA*. When expanding a label $L_u$, the path $P_{L_u}$ is extended by an edge to each neighboring node $v$. The extended path $P_{L_v}$ then needs for its distinctness to be evaluated. To avoid having to do it from scratch for every new path extension, each label retains the information on difference of its path from each of the solution paths, instead of just the minimum that corresponds to the distinctness value. This allows us to calculate the distinctness incrementally.

It is here that the incremental-update property of the $d_{\mathrm{CEL}_{\mathrm{red}}}$ metric comes in handy. The distinctness of $P_{L_v}$ is determined from that of $P_{L_u}$ using the information about the extending edge only. The edge is tracked down in each solution path $P_{\mathrm{sol}}$. If it is present in $P_{\mathrm{sol}}$, it means it shares this edge with $P_{L_v}$ and, hence, the difference of $P_{L_v}$ from $P_{\mathrm{sol}}$ is decreased by the relative contribution of the edge's length to the total length of $P_{\mathrm{sol}}$. As explained in the metric's description, the length of the solution path to which $P_{L_v}$ will eventually be extended is unknown at this point, therefore the length of $P_{\mathrm{sol}}$ is regarded as the whole when computing the relative difference.

In order to increase the efficiency of the edge tracking in a solution path, in addition to recording the difference, we can keep track of the last shared edge in the solution path and only search from there on. This relies on the assumption that, if two edges are present in both paths, they are also visited in the same order on the way from the origin to the destination. The effect is the greatest when comparing longer sections that the two paths have in common because the search picks up where it left off when it tracked down the preceding edge. The next edge in the shared sequence is thus found immediately. It is when the paths finally diverge that the next edge is searched for possibly up to the very last edge of the solution path, but only from the last shared edge. Once the paths rejoin, the information is updated, which shortens the future edge searches in the solution path.

### 4.4.2 Diversity Priority Queue

In order to utilize the paths' distinctness to point the search in the right direction with the goal of discovering the most distinct solution paths first, the default priority queue used in NAMOA* has to be replaced by a more complex structure. The default queue, implemented as a heap, maintains the lexicographically lowest label at the top. This ensures that the label chosen for extension is always non-dominated, while the dominance relationships among the rest of the labels in the queue are of no importance. However, if the distinctness of the path represented by a label is to influence when the label is offered by the queue for extension, the heap is not sufficient anymore. In this section, we describe our custom priority queue designed for the purpose of directing the search of NAMOA* toward a diverse subset of the Pareto optimal solutions.

The diversity priority queue maintains two separate data structures: a heap for the non-dominated labels (ordered by distinctness) and a set for the dominated labels. It is important to keep the dominance relationships among the labels up-to-date at all times, so as to keep track of all the non-dominated labels in the queue. Therefore, the queue internally works with structures consisting of a label $L$, a set of labels dominated by $L$ and another set containing labels that dominate $L$. Not all the dominated and dominating labels are necessarily recorded for each label, as the queue relies on the hierarchy of dominance, as depicted in Figure 4.3, which naturally arises when inserting new labels into it.

To illustrate this on an example, let us assume the state of the priority queue captured in Figure 4.3, in which we have three non-dominated labels and five labels each directly dominated by at least one of the non-dominated labels. When inserting a label $L_{\mathrm{new}}$ that dominates all of the previously non-dominated labels, only these three labels will comprise $L_{\mathrm{new}}$'s set of dominated labels. Due to the transitivity of the dominance relation, it is automatically inferred that $L_{\mathrm{new}}$ also dominates the other five labels, but there is no need to

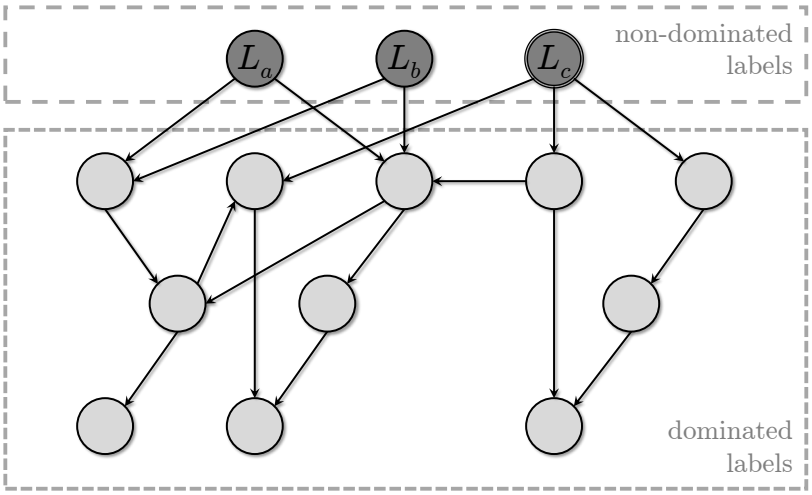



Figure 4.3: The dominance hierarchy in the diversity priority queue. The arrows point from a dominant to a dominated label, but the dominance relationship is recorded in both labels. The non-dominated label with the highest distinctness is indicated by a double-circle.

add them to $L_{\text{new}}$'s set of dominated labels too. The reason for that is that when a label $L_3$ is dominated by a label $L_2$, which again is dominated by a non-dominated label $L_1$, removing $L_1$ from the queue does not directly affect $L_3$. However, it does affect $L_2$, since $L_2$ would become a non-dominated label in case $L_1$ was its sole dominant label. We will discuss this in more detail when we describe the individual operations of the priority queue.

As a consequence of insertions, extractions and removals from the diversity priority queue, the dominance relationships established among the dominated labels can be in various "directions" within the set, but a cycle is impossible to arise. If it did, it would suggest there is a label $L_1$ that is dominated by a label $L_2$ which is dominated by $L_1$ itself. Nor can there be a dominated label whose dominance relationships, if followed recursively in the reversed direction of the arrows in Figure 4.3, would not ultimately lead to a non-dominated label.

### Inserting a Label

When inserting a new label $L_{\text{new}}$ to the priority queue (Figure 4.4) we need to determine whether it should be included in the heap of non-dominated labels or the set of dominated labels. In order to do that, $L_{\text{new}}$ is compared with each of the non-dominated labels and their dominance relationship is evaluated. There are three possible outcomes of the evaluation.

First, if $L_{\text{new}}$ is dominated by a label $L_{\text{nd}}$, $L_{\text{new}}$ is added to $L_{\text{nd}}$'s set of dominated labels and $L_{\text{nd}}$ is added to $L_{\text{new}}$'s dominant labels. Second, if $L_{\text{new}}$ dominates $L_{\text{nd}}$, the labels are added to each other's respective sets recording the dominance relationship in a similar fashion as in the previous evaluation outcome. In addition, $L_{\text{nd}}$ is moved from the heap of non-dominated labels to the set of dominated labels. In these two cases, the new label is marked as dominated or dominant, respectively. The marking may help avoid unnecessary dominance checks when $L_{\text{new}}$ is compared with the remaining non-dominated labels, as it eliminates the need to verify the opposite dominance relationship. In other words, once

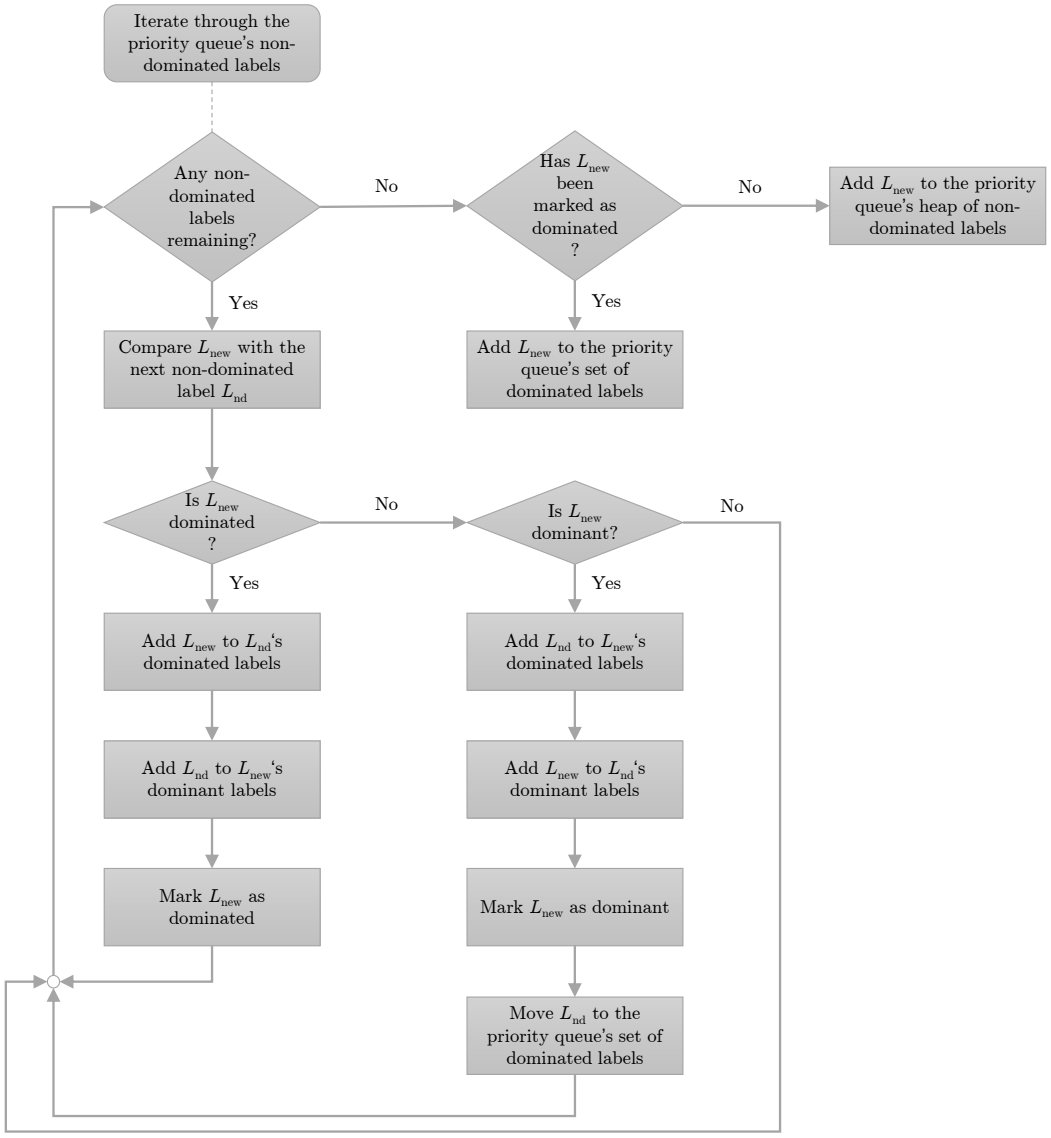



Figure 4.4: A flowchart outlining the insertion of a label into the diversity priority queue.

$L_{\text{new}}$ is identified as dominant among the non-dominated labels, it is not possible for it to be dominated by another label because the label $L_{\text{nd}}$ it dominates was previously non-dominated. Utilizing the transitivity of dominance, $L_{\text{new}}$ can be declared non-dominated instantly. Analogically, if $L_{\text{new}}$ is dominated by $L_{\text{nd}}$, then $L_{\text{new}}$ certainly cannot dominate any of the other non-dominated labels.

When, after all evaluations, $L_{\text{new}}$ is marked as dominated, we add it to the queue's set of dominated labels, and when it is marked as dominant, it goes straight to the heap of non-dominated labels. Here, the third possible evaluation scenario can occur, when $L_{\text{new}}$ is neither dominated by a non-dominated label nor does it dominate any of them. Keeping the transitivity property in mind, it is clear that $L_{\text{new}}$ cannot be dominated by any of the remaining labels either (note that each of them is dominated by at least one of the non-dominated labels). Hence, it can safely join the non-dominated labels.

#### Extracting a Label

An extraction from the priority queue (Figure 4.5) returns the currently most distinct non-dominated label $L_{\text{extr}}$, which is retained at the top of the heap of non-dominated labels. Therefore, identifying and removing such a label from the queue is swift, especially, considering that the non-dominated labels typically constitute only a small proportion of all labels in the queue at any time during the search. However, what slows the extraction down is the following update that needs to be performed subsequently in order to maintain the queue's consistency.

The update deals with the labels immediately dominated by $L_{\text{extr}}$, i.e. those that are contained in $L_{\text{extr}}$'s set of dominated labels. In case $L_{\text{extr}}$ was the only label dominating them, these labels become candidates for a transfer to the heap of non-dominated labels, whereas for the labels that are deeper in the hierarchy nothing changes with the removal of $L_{\text{extr}}$. So, iterating through all $L_{\text{extr}}$'s dominated labels, $L_{\text{extr}}$ is removed from their set of dominant labels. Whenever the remaining set is empty, the dominated label $L_{\text{dom}}$ is compared against all currently non-dominated labels. If none of them dominates $L_{\text{dom}}$, it is moved to the non-dominated heap. Before it is actually inserted into the heap, its distinctness must be updated to reflect the latest additions to the solutions. It is likely that, at this point, the partial path represented by $L_{\text{dom}}$ has not been compared to one or more solution paths that were added most recently. This *just-in-time* approach is essential in reducing the amount of time spent computing the difference of partial paths, represented by the labels in the queue, from newly discovered solution paths. The reasoning behind that is that an up-to-date distinctness is only ever significant once the label becomes non-dominated and, hence, a candidate for extraction, which is affected by the distinctness.

If $L_{\text{dom}}$ is dominated by at least one of the non-dominated labels, it remains in the dominated set. During the comparing, however, new dominance relationships might be established between $L_{\text{dom}}$ and the non-dominated labels. Although the comparing could be stopped as soon as the first non-dominated label that dominates $L_{\text{dom}}$ is encountered, following through with the evaluations serves the purpose of reducing the number of repeated comparisons in the future. Recording only one dominant label for $L_{\text{dom}}$ would cause it to be compared with the non-dominated labels (or at least the first few of them) frequently because the only dominant label would, most likely, soon be extracted from the queue, leaving $L_{\text{dom}}$

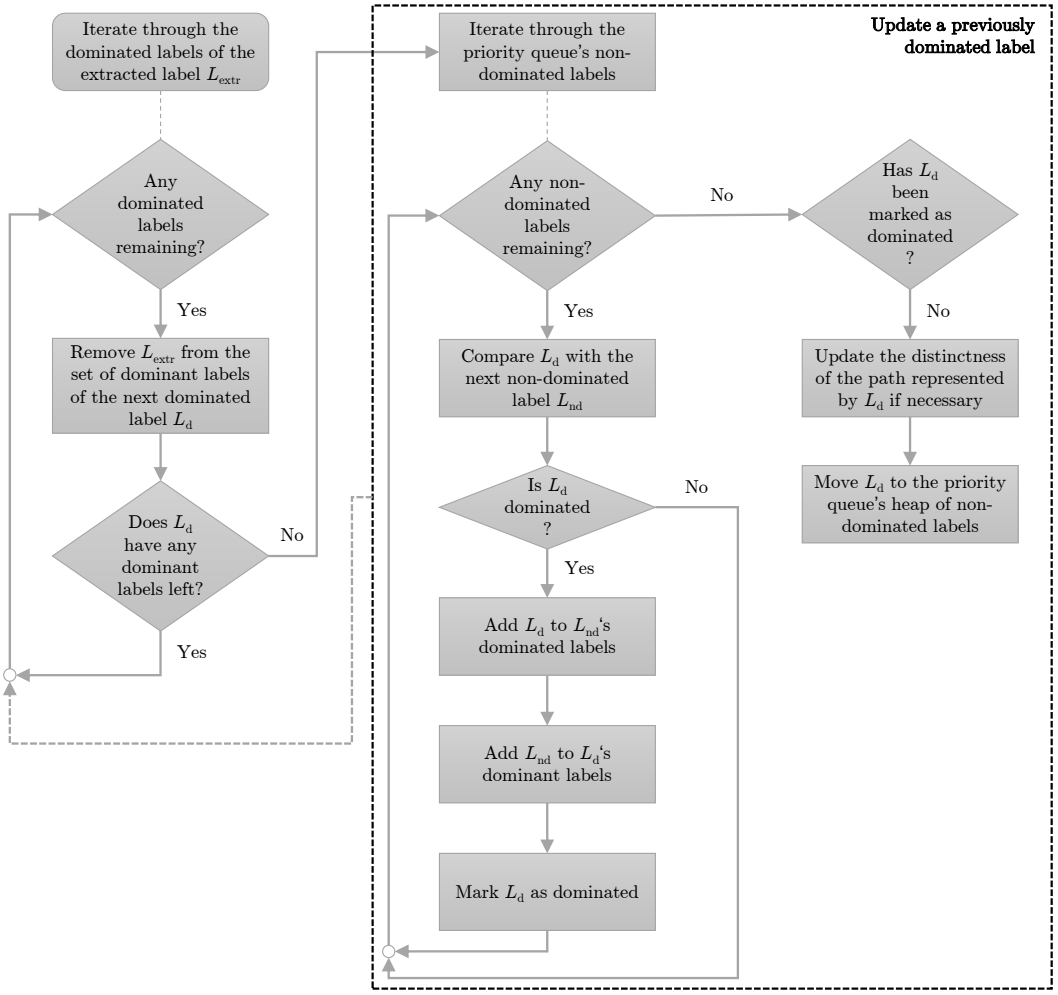



Figure 4.5: A flowchart outlining the extraction of a label from the diversity priority queue.

with an empty set of dominant labels and in need of comparison with the non-dominated labels again. This can easily become inefficient in case the single dominated label assigned to $L_{\text{dom}}$ is the first one in the heap, which gets extracted right in the next iteration of the search, and then the new top label in the heap becomes $L_{\text{dom}}$'s dominant label, and so on. On the other hand, acquiring multiple dominant labels at once considerably decreases the probability of $L_{\text{dom}}$ running out of dominant labels any time soon.

In the special case that the extracted label represents the destination node, all labels dominated by it can be removed from the priority queue, since there is no way the relationship between them and the destination label could become anything else than dominated. Not only does this provide a beneficial pruning of the search space, it avoids the cumbersome update process described above. The labels to be removed include not only the immediately dominated ones, but all dominated labels down to the bottom of the hierarchy. It is true that this might not cover all the labels dominated by $L_{\text{extr}}$ in the entire priority queue, but it is as good as it gets without having to compare $L_{\text{extr}}$ with all labels in the queue when it is being extracted. The rest of the labels dominated by this destination label, which are not

in the dominated subtree of $L_{\text{extr}}$, will be intercepted and filtered if they are ever extracted from the queue (refer to the filtering in NAMOA*).

### Removing a Label

In the majority of cases, the removal operation (Figure 4.6) is called on a dominated label in the queue. That means that, in addition to the immediately dominated labels of the removed label $L_{\text{rem}}$, we have to handle its immediately dominant labels too. When a label that both dominates and is dominated by others is removed, the reference to it is deleted from all those labels that had a dominance relationship with it recorded. Subsequently, the dominance references require mending so the information about $L_{\text{rem}}$'s dominant labels dominating its dominated labels does not get lost. To achieve that, the labels that were previously dominated by $L_{\text{rem}}$ need to be relinked directly to each of the labels that previously dominated $L_{\text{rem}}$, and vice versa, so as to bridge the gap in the dominance hierarchy caused by the removal.

If $L_{\text{rem}}$ is non-dominated, then the scenario is identical to the extracting of a label, except the label being "extracted" is not necessarily the top label in the heap. Its immediately dominated labels must be updated in the same fashion nonetheless, including the path distinctness update, as they may potentially become non-dominated. This is not necessary when $L_{\text{rem}}$ is dominated, since all the labels that were dominated by $L_{\text{rem}}$ remain dominated, only the dominant labels are those that dominated $L_{\text{rem}}$ before its removal, in addition to their original dominant labels if there were any.

Note that, due to the way removing and extracting of a label from the queue works, it cannot happen that the heap of non-dominated labels is empty and, at the same time, there are some labels in the set of dominated labels.

### Batch Update of the Labels' Distinctness

When a new path is added to the solution set, all existing labels in the queue become affected, since the distinctness of the paths they represent depends on the solution paths found thus far. The non-dominated labels in the queue are ordered by their path's distinctness when compared to the solution paths, and the label to be extracted from the queue next is decided by the greatest distinctness among them. Therefore, at least this subset of labels must be kept up-to-date in this aspect at all times.

The newly added solution path $P_{\text{sol}}$ naturally does not affect every other path in the same way. Some of them may share a substantial portion with $P_{\text{sol}}$, which could decrease their distinctness if they were more different from the previous solution paths. Others might be entirely different from $P_{\text{sol}}$, which would make them equally distinct in the new solution set as they were before the addition of $P_{\text{sol}}$ to the solutions. Hence, it is very likely that updating the distinctness of the non-dominated labels will shuffle the heap and a different label will take the top position, while the former top label may sink to the bottom and possibly never get extracted before the distinctness threshold is reached and the search is terminated. This is important in keeping the search on the right track – following those paths that are the most different from the previous solutions found.

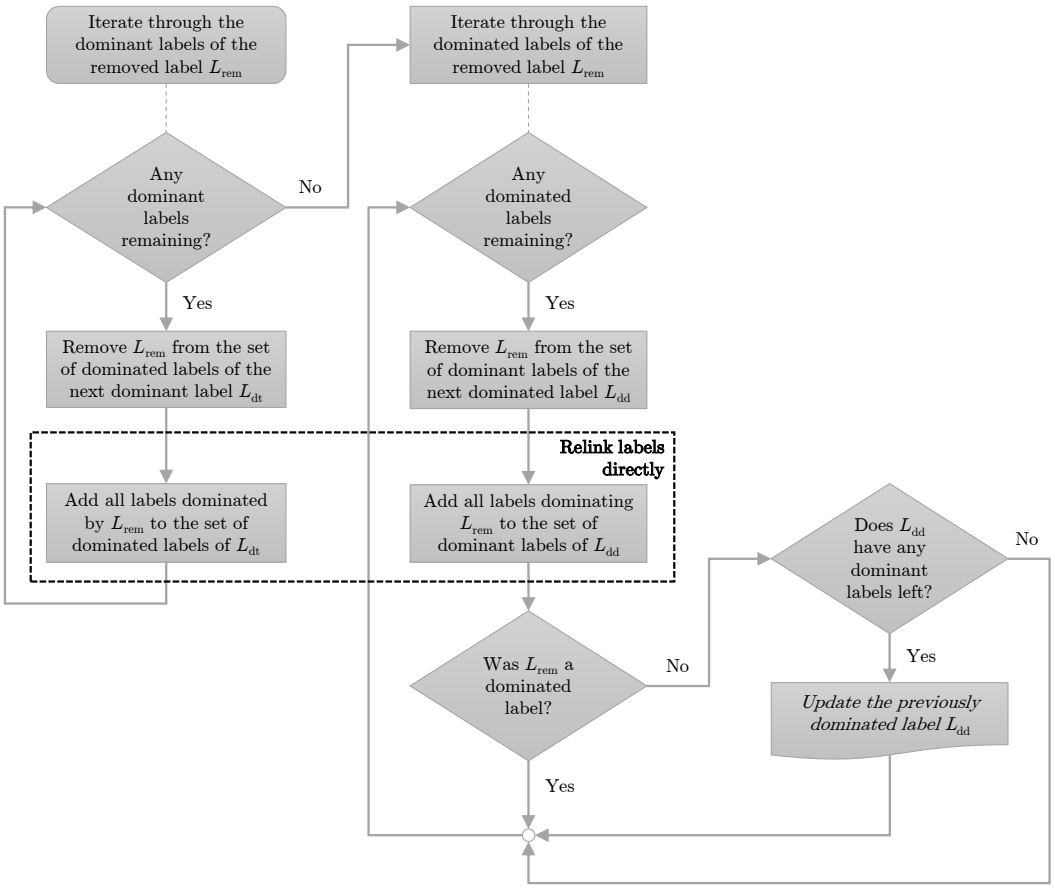



Figure 4.6: A flowchart outlining the removal of a label from the diversity priority queue. It refers to the *Update a previously dominated label* component in Figure 4.5.

The dominated labels, which, incidentally, comprise the great majority of all labels in the priority queue during the search, do not rely on up-to-date distinctness values in any way. Hence, the distinctness reassessment can safely be postponed until a previously dominated label becomes non-dominated (such as during the extraction or the removal of a label from the queue). Only then, just before it gets integrated into the heap of non-dominated labels, does the distinctness of the label's path need to be updated. At that point, the update might involve the comparison with more than one new solution path. Still, a significant amount of computational power is saved by performing the distinctness update *lazily*, since many of the dominant labels never actually make it to the non-dominated heap.

### 4.4.3 Integration of the Diversity Priority Queue into NAMOA*

Integrating the diversity priority queue into NAMOA* introduces multiple essential additions to the algorithm, as can be observed in Algorithm 2. First of all, before the algorithm can commence the search for diverse solutions, it requires at least one initial solution path to start comparing the labels in the priority queue with. The most straightforward way to achieve that is to initialize the solution set $S$ with the shortest paths identified by the TC heuristic looking for the optimum values for each individual criterion in the preprocessing stage (line 4). We prefer to have these paths among the solutions anyway, as they offer a good reference and are often highly desirable due to their optimality in one of the criteria.

The initial label $L_s$ created for the origin node in line 1, which is added to the priority queue at the beginning, has a distinctness value of 1. This is a crucial initialization that affects all future labels because, ultimately, they all originate from $L_s$. The setting of the distinctness to 1 is a prerequisite for the incremental update, which gradually decreases the path's difference from a solution path whenever it is extended by an edge that is shared with the solution path. Note that, besides the distinctness value, labels in the diversity-aware NAMOA* also contain the corresponding path, as it is required for the evaluation of their distinctness.

As we have mentioned earlier, whenever a new solution is found, the distinctness of the labels in the queue must be updated, so as to take into consideration the new solution's contribution in their distinctness values. This is a necessary adjustment for the search to resume extending paths that maximize the difference from the most up-to-date solution set. The batch update is carried out during the solution path recording in line 13.

Finally, the algorithm contains a new termination condition (line 7) that checks whether an extracted destination label satisfies a predefined distinctness threshold $\theta$. If not (i.e. its distinctness is lower), the label's path is not added to the solution set and the search is stopped. This early termination relies on the discovery of solutions in a descending order of distinctness, and it is the primary benefit of using the diversity priority queue. Once the distinctness threshold is reached, the algorithm assumes that the future solutions would only be less distinct, and so it returns the set of Pareto optimal solutions it found thus far and does not explore the remaining paths.

There is another potential use for the distinctness threshold that we feel compelled to warn about. It may be tempting to use $\theta$ for additional pruning based on distinctness, wherein a new label would be discarded if its path had a lower distinctness than $\theta$. This would, however, cause the algorithm to lose the optimality property because it would allow

**Algorithm 2:** NAMOA*$_{\text{div}}$

**Input :** graph $G(V, E, \vec{c})$, origin node $s \in V$, destination node $t \in V$, heuristic function $h$, difference function $d$, distinctness threshold $\theta$
**Output:** a diverse subset of non-dominated solution paths

1 $L_s \leftarrow (s, P_s, \vec{0}, \vec{h}(s), 1, \texttt{null})$ `// initial label of the origin node` $s$
2 $G_{\text{op}}(s) \leftarrow \{L_s\},\ G_{\text{cl}}(s) \leftarrow \emptyset$ `// sets of open and closed labels of` $s$
3 $Q \leftarrow \{L_s\},\ C \leftarrow \emptyset$ `// priority queue and solution label set`
4 $S \leftarrow$ single-criterion shortest paths `// solution path set`
5 **while** $Q$ is not empty **do**
`/* Label selection */`
6 select a non-dominated label $L_u = (u, P_u, \vec{g_u}, \vec{f_u}, \delta_{P_u}, L_{u\text{Pred}})$ with the highest distinctness $\delta_{P_u}$ from $Q$
7 **if** $\delta_{P_u} < \theta$ **then break** `// check termination condition`
8 move $L_u$ from $G_{\text{op}}(u)$ to $G_{\text{cl}}(u)$, and remove $L_u$ from $Q$
9 **foreach** $L_t = (t, P_t, \vec{g_t}, \vec{f_t}, \delta_{P_t}, L_{t\text{Pred}}) \in C$ **do** `// lazy filtering`
10 **if** $\vec{g_t} \prec \vec{f_u}$ **then continue** with the next label selection
`/* Solution path recording */`
11 **if** $u = t$ **then**
12 add $L_u$ to $C$ and add $P_u$ to $S$
13 update distinctness of labels in $Q$
14 **continue** with the next label selection
`/* Label expansion */`
15 **forall** successor nodes $v$ of $u$ that do not create a cycle in the path **do**
16 $\vec{g_v} \leftarrow \vec{g_u} + \vec{c}(u, v),\ \vec{f_v} \leftarrow \vec{g_v} + \vec{h}(v)$
17 $\delta_{P_v} \leftarrow \delta(P_v, S)$ `// evaluate distinctness`
18 $L_v \leftarrow (v, P_v, \vec{g_v}, \vec{f_v}, \delta_{P_v}, L_u)$
19 **if** $\exists (x, P_x, \vec{g_x}, \vec{f_x}, \delta_{P_x}, L_{x\text{Pred}}) \in C \mid \vec{g_x} \prec \vec{f_v}$ **then**
20 discard $L_v$ `// filtering`
21 **else if** $v$ is a new node, i.e. a node with no labels so far **then**
22 $G_{\text{op}}(v) \leftarrow \{L_v\},\ G_{\text{cl}}(v) \leftarrow \emptyset,$ add $L_v$ to $Q$
23 **else if** $\exists (x, P_x, \vec{g_x}, \vec{f_x}, \delta_{P_x}, L_{x\text{Pred}}) \in G_{\text{op}}(v) \cup G_{\text{cl}}(v) \mid \vec{g_x} \prec \vec{g_v}$ **then**
24 discard $L_v$ `// pruning`
25 **else**
26 **foreach** $L_y = (y, P_y, \vec{g_y}, \vec{f_y}, \delta_{P_y}, L_{y\text{Pred}}) \in G_{\text{op}}(v) \mid \vec{g_v} \prec \vec{g_y}$ **do**
27 remove $L_y$ from $G_{\text{op}}(v)$ and from $Q$ `// pruning`
28 **foreach** $L_y = (y, P_y, \vec{g_y}, \vec{f_y}, \delta_{P_y}, L_{y\text{Pred}}) \in G_{\text{cl}}(v) \mid \vec{g_v} \prec \vec{g_y}$ **do**
29 remove $L_y$ from $G_{\text{cl}}(v)$ `// pruning`
30 add $L_v$ to $G_{\text{op}}(v)$ and to $Q$
31 **return** $S$

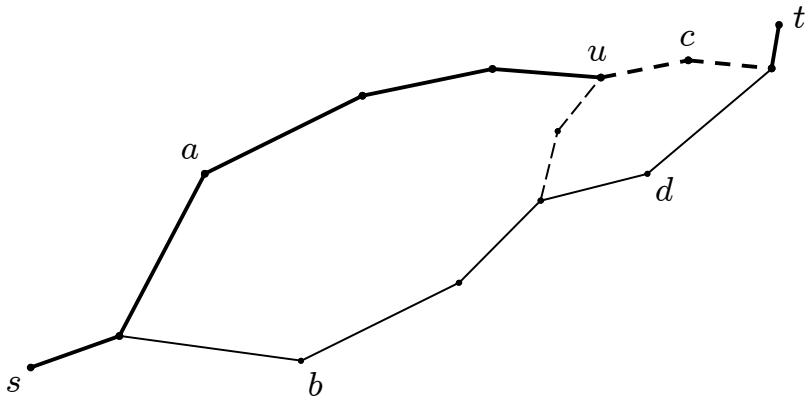



Figure 4.7: An example of the consequence of pruning using the distinctness threshold.

dominated solutions to enter the solution set. Let us demonstrate this on a simple example using the graph in Figure 4.7 and $\theta$ equal to 0.5. If path $P_{b,d}$ (the path going through nodes $b$ and $d$) was the first Pareto optimal solution to be discovered, the path $P_{b,c}$ would be disqualified for being too similar to $P_{b,d}$ (apparently it shares more than a half of its length with $P_{b,d}$). Supposing path $P_{a,c}$ is dominated by $P_{b,c}$, we would run into trouble in case $P_{b,c}$ was pruned based on the distinctness before it "reaches" node $u$ because, once $P_{a,c}$ reaches $u$, there will be no label of $u$ to dominate it (and prune it) and it would potentially have a free passage to the destination node. It could still be filtered due to being dominated by the first solution path, but considering $P_{b,d}$ does not dominate $P_{b,c}$ chances are that $P_{a,c}$ is not dominated by it either. For this reason, we cannot prune non-destination labels with a distinctness lower than a threshold, for these labels, even though they will not make it to the solution set, are still required to prune dominated labels with a higher distinctness.

We should also point out that NAMOA*$_{\text{div}}$ intrinsically returns a set of solution paths, not merely labels like the original algorithm does, since they are required for the distinctness evaluation during the search. Therefore, as soon as a label is added to the solution label set $C$, the corresponding path is added to the solution paths in $S$ (line 12). Depending on how the paths associated with the labels are represented, calling a backtracking procedure might be necessary at this point in order to recreate the solution path from the destination label.

# Chapter 5

# Implementation

This chapter offers an insight into the implementation details and the type of data that were used in the search algorithms.

## 5.1 Graph Construction from Road Network Data

Searching for the shortest paths in road networks being the focus of our work, it was imperative that we ran the algorithms with some real-world data, or as close to them as possible. A set of such road networks was put together for the "9th DIMACS Implementation Challenge – Shortest Paths"[1], and they remained available for public use. There are twelve datasets of an increasing size: the smaller ones representing a big city or a state in the USA, and the larger ones describing the road network of a greater region, such as the Western USA or even the whole of the USA. The information provided in the datasets includes geographic coordinates (latitude and logitude) of junctions, and two types of costs (physical distance and travel time) for each of the road segments between a pair of junctions. These costs represent the two criteria that we will optimize during the search. The level of detail of the DIMACS data can be appraised in Figure 5.1, which provides a visualization of all roads and junctions in the smallest dataset, namely, New York City. As the creators acknowledge on their website, however, the DIMACS data are known to contain errors, which may cause a discrepancy between the shortest paths in these and the real-world networks. This is acceptable for our bi-criteria route planning experiments nonetheless.

The second source of real-world data for our testing purposes is the OpenStreetMap (OSM) project[2]. Being a project driven by volunteers and supported by the non-profit OpenStreetMap Foundation[3], it allows the crowdsourced data to be used, as well as edited and distributed, freely under the Open Database License. Incidentally, OSM's collaborative nature is its foremost advantage over the proprietary competition, as it draws in both enthusiasts with local knowledge and GIS[4] professionals from all over the world, rendering the data as real-world as it gets. A great proportion of the OSM contributors are cyclists,

[1] http://www.dis.uniroma1.it/challenge9/download.shtml
[2] https://wiki.openstreetmap.org/
[3] https://wiki.osmfoundation.org/
[4] https://en.wikipedia.org/wiki/Geographic_information_system

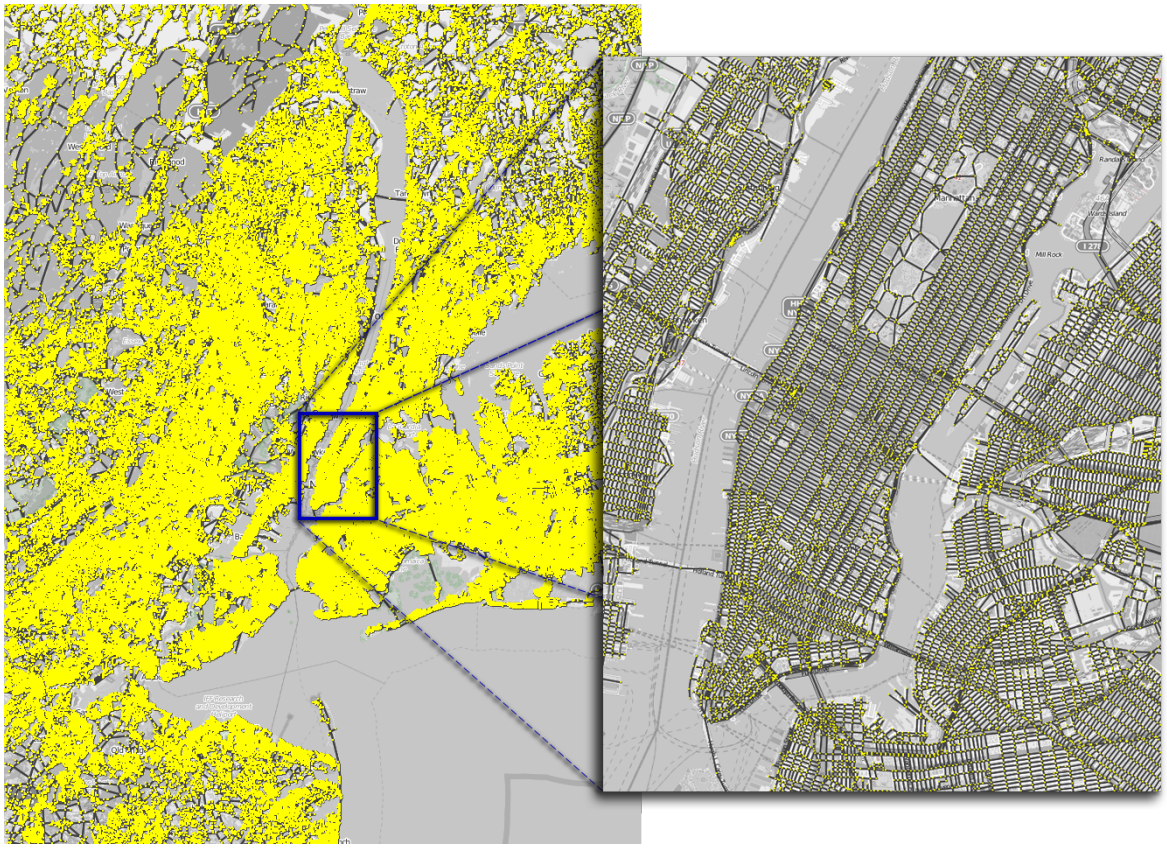


Figure 5.1: Visualization of the New York City DIMACS road network. Nodes (junctions) are displayed as yellow dots and edges (road segments) are the black lines connecting them.

Figure 5.2: Visualization of the Prague OSM road network (bicycle-navigable paths only).

who meticulously chart bike paths and other navigable trails. This makes the OSM data particularly suitable for planning bike trips, which is exactly what we use this data for in our experiments. The entire world map is available for download from the OSM wiki page, either as a whole or by smaller regions. Choosing the region of the city of Prague for our experiments, we extracted and calculated the essential information from the OSM data and transformed it into the simple DIMACS format. Roads that do not permit bicycle use, such as most freeways and long tunnels, or are in another way ill-suited to riding a bicycle, were ignored during the extraction. The three types of costs that we obtained are the physical distance, the travel time and the level of comfort the bike path or road offers, which enables us to run and test our search algorithm with three different criteria. The comfort cost corresponds to the bicycle-friendliness based on the road/path type and the traffic. A lower cost indicates a better suitability for riding a bicycle.

Now that we have uniform input data for our multi-criteria search algorithm, we can commence the construction of the graph that will be searched by the algorithm for Pareto optimal paths according to the provided costs. The costs are defined in separate files – one for each criterion – and the junction coordinates are in yet another file. Hence, we need to create the graph's nodes first and then add the edges as defined by a pair of nodes in the cost files, while combining the costs of each edge into a single cost vector $\vec{c} = (c_1, c_2, \ldots, c_n)$ for $n$ different criteria. To build and store the graph we utilize the graph structure from the *basestructures* library developed by the Artificial Intelligence Center[5] at the Czech Technical University.

## 5.2 Multi-Criteria Search Implementation

We implemented the NAMOA* algorithm in Java 8 using some of its native data structures in the process, but naturally developing custom classes for more complex objects. We started with the basic NAMOA*, which, apart from laying the foundation for our diversity-aware extension NAMOA*$_{\text{div}}$, would serve as a performance reference and for the verification of the solution's correctness. The NAMOA* algorithm was then modified and extended to incorporate the diversity awareness into the search, developing thus the NAMOA*$_{\text{div}}$.

### NAMOA* with a Post-Processing Filter

The standard NAMOA* algorithm, as described in Section 4.1, works with labels of the custom `LabelWithHeuristic` class, which stores the ID of the node the label is associated with, the cost vector of the path from the origin up to the node, the evaluation vector with the heuristic cost estimates for the entire path to the destination, and a reference to the previous label on the path (the predecessor). For the priority queue of labels we use the Java's own `PriorityQueue` class, which implements a balanced binary heap. The labels are sorted lexicographically by their evaluation vector, which ensures that a non-dominated label is always extracted from the priority queue.

The heuristic function of our choice was the bounded TC heuristic, using which the algorithm had performed best according to [25], especially in easier problem instances where

[5] http://agents.felk.cvut.cz/

the heuristic ended up precomputing the estimates only for a small portion of the nodes in the graph.

There are also a few tweaks that we introduced to the pseudocode outlined in Algorithm 1 in Chapter 4 when we implemented the search algorithm. First of all, we merged the $G_{\text{op}}(u)$ and $G_{\text{cl}}(u)$ sets of each node $u$ into one set and, instead, equipped each label with a flag determining whether the label is open or closed. Working with a single set is more convenient, and setting a binary flag may even be slightly more efficient than moving the labels between two data structures.

Another modification concerns the cycle checking when generating successor labels for a label currently being expanded, performed in line 12 of Algorithm 1. Testing whether the successor node is not already included in the path represented by the label is an extremely costly operation to perform for all candidate successors at each expansion. Instead we resort to performing the test for the immediate predecessor on the path only, that is, we discard the candidate successor node if it is the node immediately preceding the node whose label is being expanded. In other words, we explicitly avoid going there and back between two neighboring nodes on a path. However, longer cycles are left to be taken care of by pruning, as labels representing a path with a cycle are always dominated in the set of labels of a particular node. That, of course, is only true in a graph with non-negative edge costs and using a consistent heuristic function, which is our case.

When the search is finished, the solution labels are transformed into solution paths by backtracking the predecessor references to the origin prepending edge after edge along the way. Only then are the diverse solutions filtered. We use the greedy selection method (see Section 4.3.1) for a better comparability of the diverse subset of solutions with those found by NAMOA*$_{\text{div}}$, considering that NAMOA*$_{\text{div}}$ discovers the diverse solutions intrinsically in a greedy fashion.

### Diversity-Aware NAMOA*

The first step toward integrating the diversity awareness into NAMOA* is to augment the labels by including the path they represent. The new `LabelWithHeuristicExtended` class thus acquires a new member – a `PathSegment` structure that represents the path from the origin up to the current node.

In order to maximize the memory efficiency, instead of holding the whole list of edges, the `PathSegment` only records the last edge on the path, and a reference to the corresponding `PathSegment` that ends at the first node of the edge and is thus a member of the predecessor label. When the path is required in its entirety, it can be followed, through recursion, edge by edge in reverse order, much like iterating through a linked list. Such complete path traversal is used when the `PathSegment` is being compared with a brand new solution path, from which the difference was not previously computed. This only happens either when a new solution is found and the non-dominated labels get updated, or when a previously dominated label becomes non-dominated and was not evaluated against the latest solutions before.

The `PathSegment` also maintains a list in which it records the differences from all the solution paths. So, when a new label is created by extension, its `PathSegment` is created from that of the predecessor by updating the differences from all the solution paths remembered by the `PathSegment` based on the extending edge, which is also the one and only edge registered

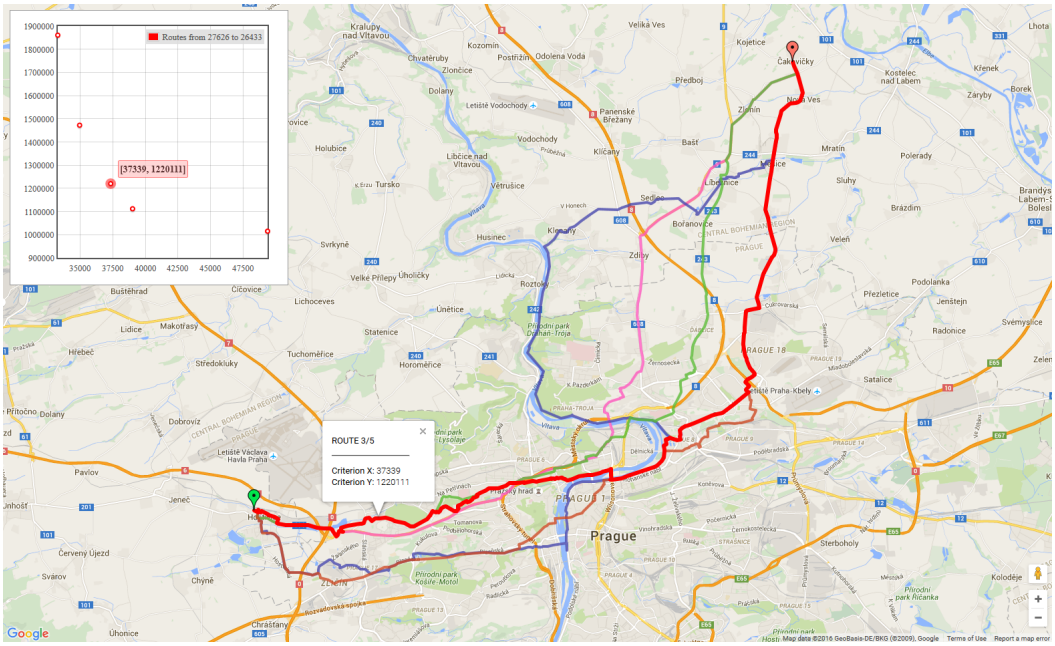



Figure 5.3: The solution visualizer. The solutions are displayed both as routes in the map and as points in the cost space plot.

in the new `PathSegment`. It also contains a separate distinctness value, which corresponds to the minimum of the listed differences, for a faster access when non-dominated labels are compared and sorted based on their distinctness, since the value may be queried repeatedly.

The next key modification is the replacement of the default priority queue with the diversity priority queue presented in Section 4.4.2. By taking the labels' distinctness into consideration, the new priority queue alters the order in which the non-dominated labels become expanded. It then depends on the setting of the distinctness threshold when the search ends because once a destination label with a lower distinctness is extracted, the search is stopped.

When finished, there is no post-processing required, since each solution is transformed into a path at the time of its discovery, which allows for a more efficient distinctness computation. Considering that every new label has its difference evaluated against the solution paths, repeated backtracking from the corresponding destination label would be very inconvenient, and it would impair the speed-up technique described at the end of Section 4.4.1 involving the records of the last shared edge.

The tweaks introduced in the standard NAMOA*, including the merging of the $G_{\text{op}}$ with the $G_{\text{cl}}$ set and the limited cycle checking during label expansion, are retained in NAMOA*$_{\text{div}}$. The bounded TC heuristic is retained for calculating the cost estimates in the preprocessing stage of the search as well.

## 5.3 Solution Presentation

Finally, we set up the visualization of the search results on the map so they can be presented in a useful and convenient way. The route visualizer is implemented in JavaScript and displays the solution routes in a web browser. As the underlaying map we use Google Maps, the most popular web mapping service. The well-documented JavaScript API[6] that Google provides for the service makes the displaying of custom routes in Google Maps fairly straightforward and allows a good deal of customization. The routes are colored randomly to make them easily distinguishable. Clicking a route toggles a small tooltip with the route's costs. The origin (green) and the destination (red) markers are also clickable and their tooltip notifies about their geographic coordinates.

In the upper left-hand corner of the map, the solutions are plotted in the cost space. Being a two-dimensional plot, if the paths have more than two costs, the points are projected onto the $xy$-plane. The $x$-axis represents the first criterion and the $y$-axis the second one, as provided in the input data. When a point is clicked in the plot, the corresponding route in the map is highlighted in red for easy following. The interactive plot is powered by the Flot library[7] for jQuery.

[6] https://developers.google.com/maps/documentation/javascript/
[7] http://www.flotcharts.org/

# Chapter 6

# Evaluation

To evaluate our implementations of the solution methods presented in Chapter 4, we conducted a series of experiments on two different datasets. We used one of the DIMACS road networks – New York City (NYC) – with the travel distance and the time costs provided. The other road network is that of Prague, containing only bicycle-navigable streets and roads, and it includes comfort costs instead of time costs. These two datasets allowed us to observe the effect of using different criterion combinations on the solutions of a multi-criteria search. The overview of their sizes is in Table 6.1.

There are three aspects in which we compare the performance of the algorithms. First, we look at the *number of label expansions* performed during the search, which indicates the size of the search space explored. The label expansion involves different processes in different algorithms, which affects their complexity. Therefore, we also measure their total *runtime* for an absolute performance comparison. The third metric – the *number of solutions* – serves the evaluation of the diversity threshold parameter's effect on the solution set reduction. We tested the algorithms with the threshold values of 0.3, 0.4 and 0.5, which correspond to the minimum diversity requirements of 30%, 40% and 50%, respectively.

The experiments were carried out in a cluster of supercomputers provided by MetaCentrum[1]. This allowed us to both tap the great computational power they offer and save time by having multiple problem instances run simultaneously and remotely. The machines were equipped with Intel Xeon E5-2670 CPUs with 2.60 GHz per core and 20 MB of cache. The amount of RAM available was 64 GB, although the tested instances never required more than 16 GB to be solved.

## 6.1 Multi-Criteria Search

The search algorithms were tested on the same 50 NYC problem instances as used in [25], so as to compare the performance of our implementation of the basic NAMOA* with that of its authors. Since they included the size of the Pareto set of solutions for each problem in their results, we also used them to double-check the correctness of our results. The instances are randomly generated pairs of nodes with various distances between them, resulting in a

[1] http://metavo.metacentrum.cz/en/index.html

Table 6.1: Road network sizes and optimization criteria in the datasets.

| | Number of nodes | Number of edges | Criteria |
|---|---|---|---|
| **New York City (NYC)** | 264,346 | 733,846 | distance, time |
| **Prague (PRG)** | 109,415 | 241,755 | distance, comfort |

high variance of problem difficulties. Each problem instance was solved 8 times discarding the first 3 results, which should account for the warm-up period of the Java virtual machine during which the performance is decreased. The resulting runtimes that we present were averaged over the remaining 5 runs. If a run took more than 15 minutes, it was terminated, since we are mostly interested in performance that is satisfactory more or less in real time. Hence, the difference between 15 minutes and 5 hours was irrelevant for our purposes.

### 6.1.1 Basic Search

The first thing to notice in our results is that our implementation of NAMOA* performs considerably better than the original implementation of its authors [25], which also used the bounded TC heuristic. When the relative runtime difference is averaged over the 50 problem instances, it turns out our implementation performs more than 20 times better. This may possibly be attributed to a more memory-efficient graph representation or even the programming language. The authors implemented the algorithm in ANSI Common Lisp, whereas we did in Java, but it appears to be unlikely that a language difference could account for a gap of an order of magnitude in performance. We also tested the algorithms on a few problem instances on a computer with a similar CPU rating as that of the authors, and the runtimes were comparable with those achieved in the cluster. Thus, the CPU of the machine on which the experiments are conducted does not seem to have a significant impact on the search performance. If anything, it is the RAM that could become the bottleneck for the more difficult instances in which a great number of labels is expanded, but the authors had an abundance of memory at their disposal.

Overall, our NAMOA* found the Pareto set of solutions within 10 seconds in a great majority of the NYC problems (see Table A.3 for a comprehensive overview of the search runtimes), and in under 2 seconds in more than a half of them. There were only a few exceptions that took up to 3 minutes. These are promising results with a real-time application in mind.

### 6.1.2 Diversity-Oriented Search

Having established the performance baseline by the NAMOA*, we can start evaluating the two diversity-oriented extensions, whose purpose is to decrease the number of solutions and make the search results more practical. As we can see in Table A.1, the whole Pareto set, which is found by the basic NAMOA*, indeed often contains hundreds of solutions. The aim

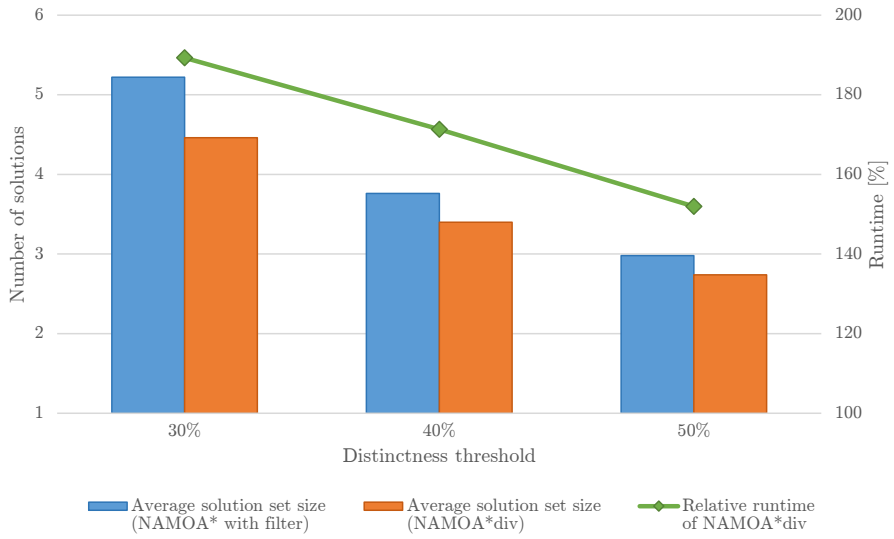


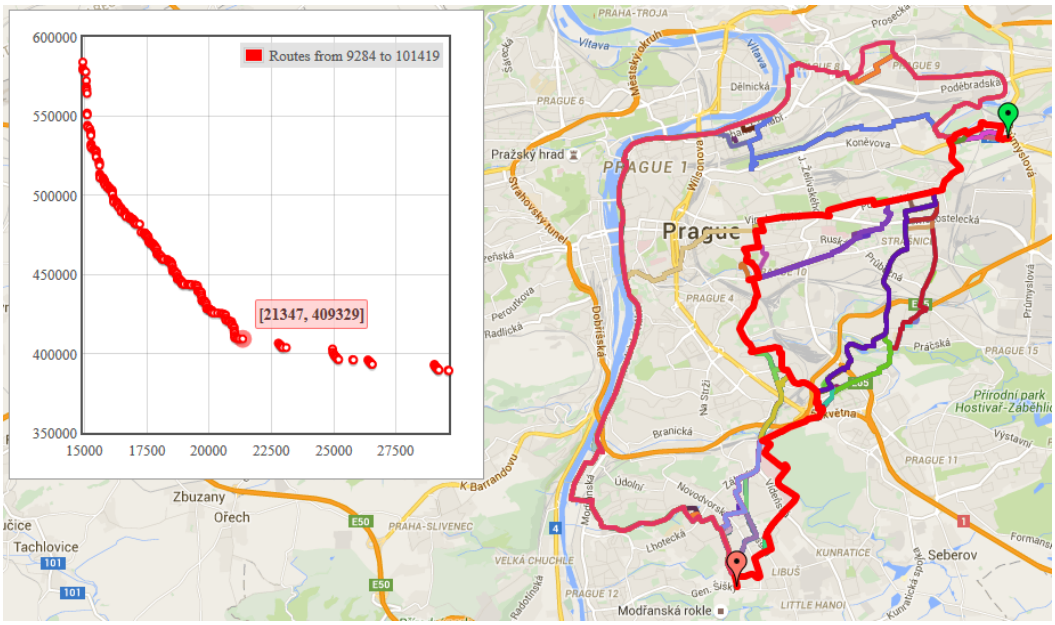

Figure 6.1: A comparison of the average size of the solution set yielded by the two diversity-oriented algorithms using various distinctness thresholds. The line indicates the relative runtime increase of NAMOA*$_{\text{div}}$ compared to its NAMOA* counterpart using the post-processing filter and the same threshold. The values are averaged over the 50 NYC instances.

of employing the path diversity in the search was to reduce it down to a set of a convenient size (3 to 10 solutions), from which it is easier to choose the desired solution.

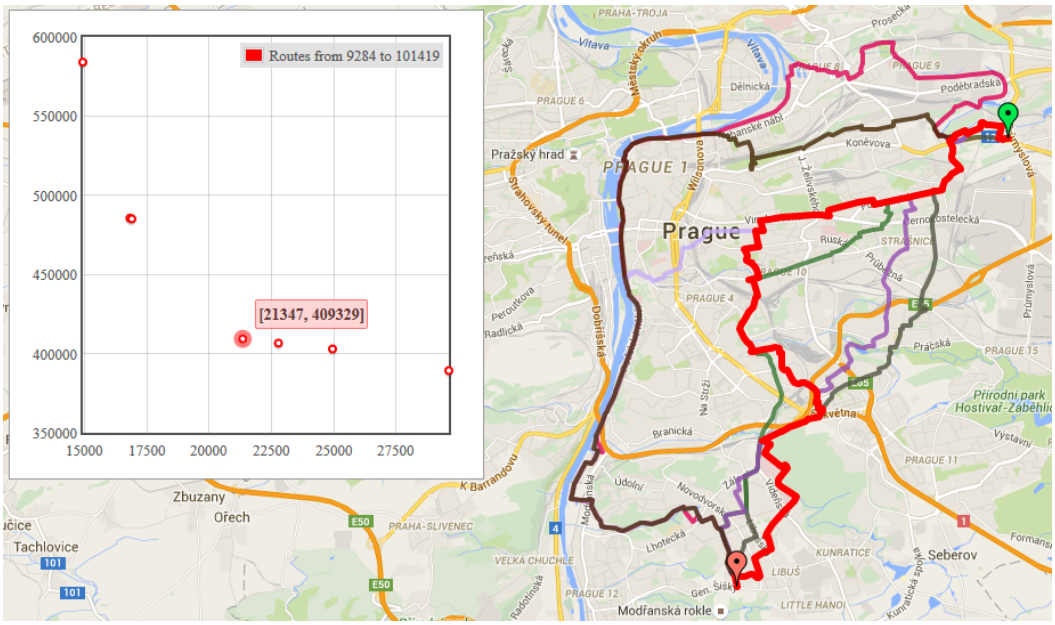
### Solution Quality

The number of solutions is indirectly controlled by the distinctness threshold setting. The threshold has a different effect in a problem instance with a great origin-destination distance than in one with a short origin-destination distance. In the former, the paths have naturally more opportunities to branch off and more space to spread out, which produces more solutions that are distinct from each other. Setting a universal threshold is therefore tricky and requires a certain amount of fine-tuning. We tested the algorithms with the threshold values between 30% and 50%. The plot in Figure 6.1 shows that applying the less strict threshold leads to an average of 5.22 solutions using the filter and 4.46 with the diversity priority queue. Up to 13 diverse solutions are offered, depending on the problem instance. On the other hand, allowing only paths that are at least 50% different from the others in the solution set reduces their average number to 2.98, or 2.74, meaning in practice that most problems have 2 to 4 solutions offered (for individual results refer to Table A.1). A 2-member solution set merely corresponds to the single-criterion shortest paths, which means the multi-criteria search added no value at all to the results. Unless the problem instance is very small, such result is not satisfactory. We therefore consider the threshold of 50% too strict and opt for 40% or less in order to obtain more useful results.

To evaluate the adequacy of the solution subset found by NAMOA*$_{\text{div}}$, it is best to




(a) The full Pareto set of 266 solutions. The Pareto frontier is plotted in the upper left-hand corner.




(b) A diverse subset of 7 Pareto optimal solutions found by NAMOA$^*_{\text{div}}$ with the distinctness threshold set to 40%.

Figure 6.2: The path coverage provided by a diverse subset of the Pareto set demonstrated on a PRG problem instance.

Table 6.2: Evaluation of both the NAMOA* with a post-processing filter and the NAMOA*$_{\text{div}}$ with three different distinctness thresholds each. The values are measured relative to those of the basic NAMOA* (NYC dataset).

| | **NAMOA*** | **NAMOA* with filter** | | | **NAMOA*$_{\text{div}}$** | | |
|---|---|---|---|---|---|---|---|
| | | **30%** | **40%** | **50%** | **30%** | **40%** | **50%** |
| **Average size of the solution set** [%] | 100.00 | 10.74 | 9.47 | 8.71 | 10.36 | 9.32 | 8.64 |
| **Average number of expansions** [%] | 100.00 | 100.00 | 100.00 | 100.00 | 78.49 | 71.03 | 65.00 |
| **Average runtime** [%] | 100.00 | 115.83 | 106.24 | 105.74 | 225.53 | 182.99 | 160.19 |

compare it with the full Pareto set of solutions visually. Figure 6.2 shows us the solution routes visualized on the map, which allows us to review the path coverage of the diverse subset. We can immediately see that the streets and roads used by the routes in the entire Pareto set at the top are almost completely covered by the 7 diverse routes at the bottom. Upon a closer inspection of the two maps, a few missing segments are noticeable in 6.2b, but they are all very short and insignificant. Note that the route colors are generated randomly and, therefore, the colors in one figure do not correspond to those in the other. There is a lot of overlapping present, particularly among the routes in the Pareto set, so matching colors between the two sets would be of no help in recognizing matching pairs of routes. One selected route pair was highlighted to demonstrate that the diverse routes are indeed a subset of the Pareto set. This can also be observed in the cost space plots, where the 7 points displayed in 6.2b belong to the Pareto frontier in 6.2a. In conclusion, the solutions found by NAMOA*$_{\text{div}}$ can certainly be pronounced adequately diverse as far as the empirical evaluation is concerned.

### Search Performance

Now, let us examine how the diversity integration affects the performance of the multi-criteria search. Consulting Table 6.2, we see that the NAMOA* that filters the solutions in a post-processing step is 6–16% slower than the basic NAMOA*. This increase in runtime comes from the additional computations that are performed after the normal run of NAMOA* in order to filter the most diverse solutions using a greedy technique. The number of label expansions performed using this approach is naturally the same as in the basic NAMOA*. On the other hand, as we can see in the rightmost section of the table, NAMOA*$_{\text{div}}$ reduces the number of expansions required for discovering a set of solutions of a roughly equal size. It drops, on average, to 71% using the moderately strict distinctness threshold. Although it is a fairly significant reduction, this value still seems too high considering that the number of solutions decreases to an average of 9% of the entire Pareto set.

To explain this only mediocre decrease, we need to consider what effect the diversity priority queue has on the search progress. Apparently, it converts it into more of a breadth-first search and causes it to expand labels that represent shorter path segments before those

Table 6.3: An example of the solution discovery in NAMOA*$_{\text{div}}$ with a distinctness threshold of 40% in the NYC problem instance number 36. The first two solutions are the single-criterion optima. For the remaining solutions the second column indicates the number of label expansions performed before discovering the solution, while the third column displays its distinctness among the solutions at the moment of discovery. The distinctness of the 7$^{\text{th}}$ solution is below the threshold.

| **Solution number** | **Number of expansions** | **Distinctness** |
|---|---|---|
| 1 | 0 | - |
| 2 | 0 | - |
| 3 | 83852 | 65.14% |
| 4 | 208300 | 47.19% |
| 5 | 260290 | 47.37% |
| 6 | 266215 | 42.07% |
| (7) | 301989 | 37.51% |

that are closer to the destination. This behavior arises from the way the distinctness is computed – assuming a total difference from all solution paths at the beginning and gradually decreasing it by the respective shared portions. Approaching the destination, the distinctness of a path segment is bound to decrease, as, sooner or later, it starts sharing some parts with the solution paths already found. The shorter path segments, which do not share as much just yet and are assumed to be different for the rest of the path, are then preferred by the priority queue. They delay the extension of those path segments that are perhaps already near the destination, and for a good reason too because chances are that one of the currently shorter path segments will develop into a more distinct solution. As a result, the algorithm extends all the path segments bit by bit, as opposed to extending one of them all the way to the destination, then doing the same with another, and so on.

The discovery of the first distinct solution therefore does not happen until a later stage of the search, when the algorithm has a better idea of the distinctness of all the path segments. Other solutions then follow with an increased frequency. See Table 6.3 for an example of a diversity-aware search terminating after 301,989 expansions, while its filtering counterpart performs all 521,546 of them (Table A.2). Also note that, in the example, the 5$^{\text{th}}$ solution is more distinct than the 4$^{\text{th}}$. This phenomenon is addressed later in Section 6.2.

The number of expansions carried out during the search directly affects its runtime. However, the 30-percent reduction of expansions is not enough to balance the computational demand added by the calculation of distinctness and by the use of the diversity priority queue. The last row in Table 6.2 summarizes the runtimes relative to the basic NAMOA*, whereas Figure 6.1 displays the runtime of NAMOA*$_{\text{div}}$ relative to the NAMOA* with the filter. We can observe that increasing the threshold reduces the relative runtime increase of NAMOA*$_{\text{div}}$ rapidly. In fact, using the strictest threshold reduces the performance gap

Table 6.4: Evaluation of both the NAMOA* with a post-processing filter and the NAMOA*$_{\text{div}}$ with three different distinctness thresholds each. The expansion and runtime values of NAMOA*$_{\text{div}}$ are measured relative to their NAMOA* counterpart with the filter (PRG dataset).

| | **NAMOA* with filter** | | | **NAMOA*$_{\text{div}}$** | | |
|---|---|---|---|---|---|---|
| | **30%** | **40%** | **50%** | **30%** | **40%** | **50%** |
| **Average size of the solution set** [%] | 9.14 | 8.16 | 7.53 | 9.07 | 8.02 | 7.47 |
| **Average number of expansions** [%] | 100.00 | 100.00 | 100.00 | 84.99 | 79.02 | 68.41 |
| **Average runtime** [%] | 100.00 | 100.00 | 100.00 | 202.97 | 204.52 | 169.96 |

down to almost 50%. But we have to keep in mind that using the threshold of 50% already yields rather unsatisfactory results, so increasing it even more with the aim of closing the performance gap completely would not be feasible.

### Distance-Comfort Criterion Combination

The results of the experiments that we ran on the PRG dataset, which contains comfort costs instead of time costs, were similar to those achieved with the NYC dataset. There were a few differences worth pointing out though. Despite being a smaller dataset (Table 6.1) with shorter distances, the average size of the Pareto set across the 50 PRG instances was 491 (Table A.4), in contrast to the 199 in NYC (Table A.1). This discrepancy can be attributed to the difference in correlation between the two criteria used in a dataset. While the traveling time highly correlates with the traveling distance (i.e. longer routes tend to take more time), the comfort cost does so to a considerably lesser degree. A direct consequence of this is a greater number of Pareto optimal solutions using the destination-comfort combination. If two criteria correlated totally, there would be only one Pareto optimal solution because the shortest path according to one criterion would necessarily be the shortest according to the other one as well. On the other hand, the less correlated the criteria are, the greater the Pareto set may become.

Comparing the statistics in Table 6.4 with Table 6.2, we notice that the solution set got reduced on average slightly more using the PRG data, while the relative number of required label expansions grew. Both of these trends are likely products of the lower criterion correlation too. During the search, the labels have a wider competition in the form of other non-dominated labels, which possibly leads to an even greater delay of the solution discovery and, with it, a greater number of label expansions performed. On the other hand, the substantially higher proportion of Pareto optimal solutions inevitably brings in even more spatially similar paths to the pool from which only the most distinct ones are eventually selected. This may be the reason for the decrease of the average solution set size, which dropped from 9.32% in NYC to 8.02% in PRG when using NAMOA*$_{\text{div}}$ with the threshold

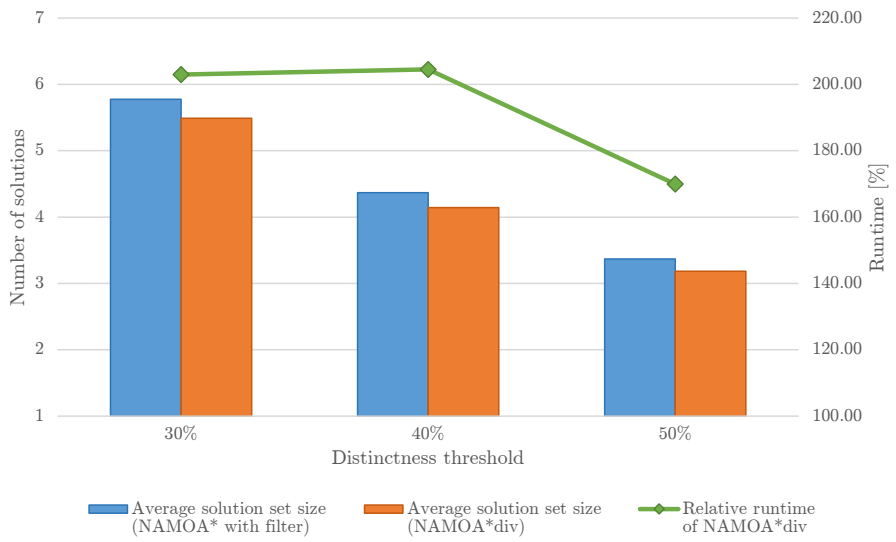



Figure 6.3: A comparison of the average size of the solution set yielded by the two diversity-oriented algorithms using various distinctness thresholds. The line indicates the relative runtime increase of NAMOA*$_{\text{div}}$ compared to its NAMOA* counterpart using the post-processing filter and the same threshold. The values are averaged over the 50 PRG instances.

of 40%. When we compare the plot in Figure 6.3 with that in Figure 6.1, however, we can see that the absolute number of solutions found for the PRG problem instances is generally higher, so now even the use of the threshold value of 50% might be considered reasonable. Moreover, the relative runtime increase drops significantly when switching from the threshold of 40% to the stricter 50%.

The NAMOA*$_{\text{div}}$ search nevertheless remains up to two times slower on average than the standard NAMOA* with the post-processing diversity filter. A few exceptions occurred in our experiments, where certain problem instances were solved faster using NAMOA*$_{\text{div}}$, such as the instances number 10 and 26 (see Table A.6 to review the precise runtimes). In the latter, using the threshold of 50% results in a more than four-times-faster performance (24.84 vs. 105.52 seconds), which is definitely more than just a marginal improvement. We should also remark that it is not at the expense of the size of the solution set, as its size is 4 in both variants (Table A.4). However, these cases are rare and are therefore considered statistical outliers.

## 6.2 Diversity Priority Queue

The most fundamental change in the NAMOA* algorithm that we performed in order to transform it into NAMOA*$_{\text{div}}$ was the replacement of the priority queue. Substituting the heap, which merely maintained the lexicographic minimum among the labels, with the diversity priority queue described in Section 4.4.2 automatically increased the computational

Table 6.5: An example of the diversity priority queue with three labels.

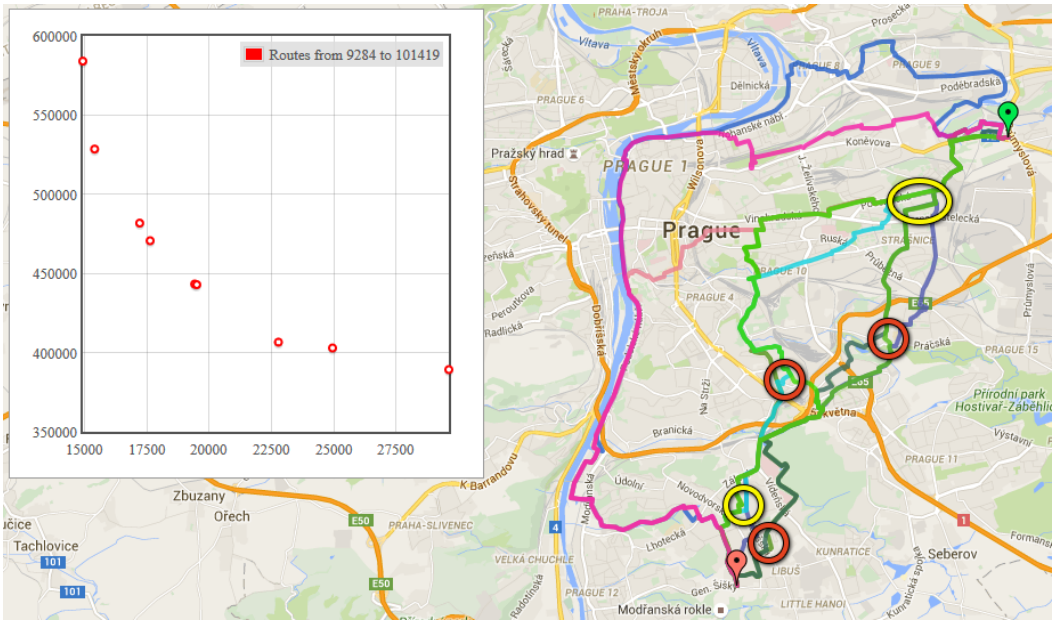
| Label | Cost estimates | Distinctness | Dominant labels | Dominated labels |
|---|---|---|---|---|
| $L_a$ | [3, 12] | 0.6 (60%) | | |
| $L_b$ | [5, 10] | 0.5 (50%) | | $L_c$ |
| $L_c$ | [6, 11] | 0.8 (80%) | $L_b$ | |

complexity of the search. Without this change the search could not become diversity-aware while retaining its optimality. We therefore tried our best to optimize the new data structure so that it does not slow down the search substantially even before introducing the distinctness computations into it.

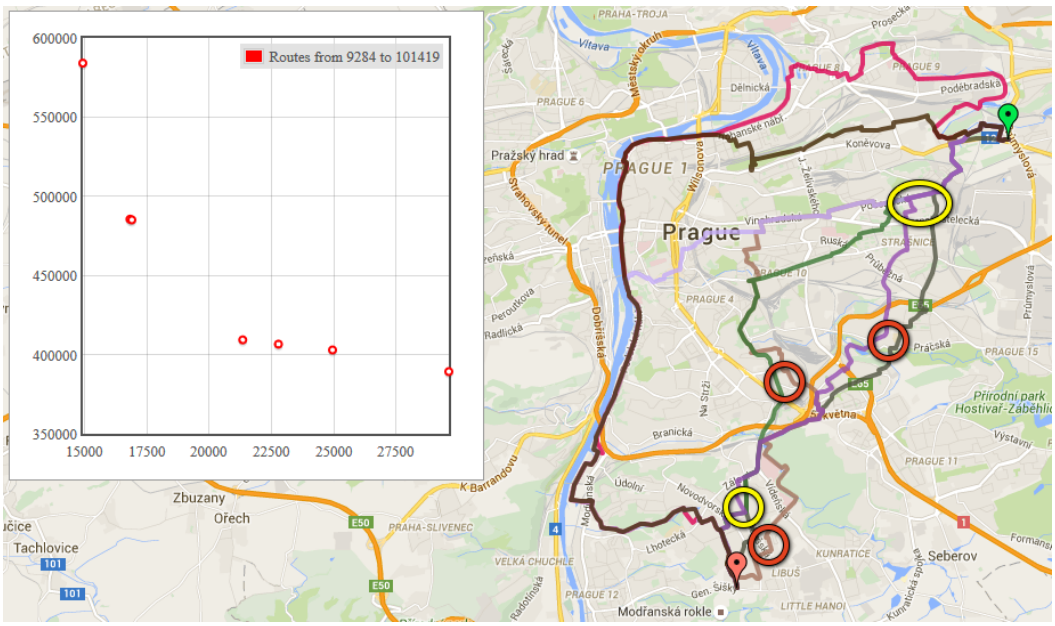
We tested the performance of the priority queue on the 50 NYC problem instances by using it (with a lexicographic sorting of the non-dominated labels) in the basic NAMOA* instead of the default one. The results are summarized in Table A.3 along with those of the other algorithms (see the column "new $Q$" under NAMOA*). When the relative runtime increase compared to the basic NAMOA* is calculated individually for each instance and then averaged, we get the value of 175.27%, which means it is less than two times slower. This is a decent performance considering the radical change of the way the priority queue works.

Once a label is extracted from the diversity priority queue, it is guaranteed to be non-dominated, which satisfies the condition of NAMOA* to yield only Pareto optimal solutions. However, the priority queue does not guarantee that the solutions are indeed discovered in a descending order of distinctness, nor that the first solution found (after the single-criterion optimal solutions have been established) is the most distinct one, the second solution is the second most distinct one, etc. It rather performs a good approximation. The source of this imperfection is the label shadowing taking place in the queue.

To illustrate it on an example, let us suppose we have three labels in the queue, as defined in Table 6.5. $L_a$ and $L_b$ comprise the heap of non-dominated labels with $L_a$ at the top due to its greater distinctness, while $L_c$ is in the set of dominated labels. May $L_a$ be assigned to the destination node and thus provide a new solution when extracted. We can see that $L_c$ was not dominated by $L_a$, but shadowed by $L_b$, which dominated $L_c$ at the time of $L_a$'s extraction. $L_c$ could later be extended into a destination label with unchanged cost estimates and a slightly lower distinctness, say, 0.7 (after the update taking the newly added path solution into account), which is still greater than that of $L_a$ when it was extracted. So, when $L_c$ finally gets extracted from the queue, the solution path it represents is more distinct than the previous solution path. As a consequence, a poorly chosen distinctness threshold used in the search might prevent some sufficiently distinct paths from making it into the solution set. In this example, such a threshold could be 0.65, which would cause the search to terminate as soon as $L_a$ (representing a solution path with a distinctness lower than 0.65) was extracted, not giving $L_c$ a chance to propagate its eventually more distinct solution path.




(a) A diverse subset of 9 Pareto optimal solutions found by the NAMOA* with the filter and with the distinctness threshold set to 40%.




(b) A diverse subset of 7 Pareto optimal solutions found by NAMOA*$_{\text{div}}$ with the distinctness threshold set to 40%.

Figure 6.4: An example of possible differences between the solution sets of the two diversity-oriented algorithms. The red circles indicate road segments that are not covered by the solution set of NAMOA*$_{\text{div}}$ and the yellow ones mark areas with a variation.

Alternatively, the label shadowing can have the effect of neglecting a potential solution path with a superior distinctness when the path is similar to a previous solution path. Let us suppose that, in the example above, the path represented by $L_c$ shares a great portion with that of $L_a$, which is extracted and added to the solutions. Then, once $L_c$ becomes non-dominated and its distinctness is updated, it drops significantly lower, say, to 0.2, because of the similar path now in the solution set. Whether $L_c$ is a destination label itself or it would be extended to a destination label with a not significantly lower distinctness, it would have provided a more distinct solution at the time when $L_a$ was extracted from the queue. Instead, it will most likely not be included in the solution set.

This behavior probably cannot be remedied without a significant performance hit on the priority queue. In order to keep track of such non-dominated relationships among the labels, the label being inserted into the queue would have to have the dominance evaluated against virtually every other label. And even then, how exactly would the strategy change in the extraction procedure? The label $L_b$ might be extracted before $L_a$ despite being less distinct because it dominates label $L_c$ that precedes $L_a$ in distinctness and, at the same time, it is not dominated by it. But then, let us consider an extended scenario where there is an additional label $L_d$ with the cost estimates of [4, 13] and the distinctness of 0.9, which is dominated by $L_a$. We would be required to apply the same double-checking to $L_b$ only to learn that $L_a$ dominates the label $L_d$, which is not dominated by $L_b$ and has a greater distinctness. In fact, the distinctness of $L_d$ is even greater than that of $L_c$, which would probably resolve the conflict in favor of $L_a$ (keep in mind that only a non-dominated label is allowed to be extracted, hence, only $L_a$ and $L_b$ are in question). And this would take us back to $L_a$ being extracted before $L_c$. There does not seem to be a deterministic strategy that would lead to always extracting a label like $L_c$ before a label like $L_a$ from the example and thus ensure that the Pareto optimal solutions are discovered in a proper descending order.

All things considered, the effects do not necessarily have to be viewed as a defect of the diversity priority queue, for they may enable solutions with a higher distinctness to be found next. For instance, instead of the first three solutions having the distinctness values of 0.8, 0.4 and 0.3, they could be 0.6, 0.55 and 0.5, as the distinctness of the future solutions depends on the already retrieved solutions. Depending on the distinctness threshold, the difference could be one versus three sufficiently distinct solutions making it to the solution set (that would be the case of a threshold value lower than 0.5 but greater than 0.4), in addition to the default solutions. This is the reason for NAMOA$^*_{\text{div}}$ occasionally finding more solutions that satisfy the distinctness threshold than the greedy filter would pick from the Pareto set (see, for example, the problem instance number 26 in Table A.1).

In the end, it is debatable which of the two scenarios is more favorable. There is an example in Figure 6.4 that illustrates possible differences between the solution sets found by the two different methods. The solution space plots show that some of the solutions are shared, while others are unique for each of the sets. Although there are fewer solutions in Figure 6.4b, the roads and streets covered by the solution routes are almost the same as those in Figure 6.4a, with only a few minor discrepancies. In any case, the solutions offered are still Pareto optimal, only a different subset.

# Chapter 7

# Conclusion

Throughout this thesis we gradually took steps toward creating a multi-criteria search algorithm that intrinsically favors solutions that are geographically distinct. We started with the determination of an appropriate metric for comparing individual paths and evaluating to what degree they differ from each other. After an analysis of several approaches, we deemed a metric based on the calculation of the cumulative edge length difference of two paths the most suitable. It satisfied the requirements for the subsequent integration into the search-algorithm and, at the same time, did not give up any important properties for use with real-world road networks.

We then described the state-of-the-art exact algorithm for solving the multi-criteria shortest path problem – NAMOA*. It served both as the foundation for the construction of our diversity-aware extension NAMOA*$_{\text{div}}$ and as a reference for the evaluation of the extension. We detailed the mechanics of a new priority queue that we designed specifically for the purposes of directing the search toward diverse results while retaining the optimality of the solutions. Combined with the selected difference metric to estimate how a path will contribute to the results' diversity, it enables NAMOA*$_{\text{div}}$ to decide accordingly which paths to explore.

To evaluate the solution quality and the performance of the new algorithm, we additionally implemented a diversity filter to the standard NAMOA*, which corresponds to the common diversifying approach. With a proper setting of the diversity parameter, NAMOA*$_{\text{div}}$ achieved similar results to those of the filtered NAMOA*. The number of diverse solutions offered varied to a certain degree between these two approaches, but in general they covered more or less matching road and street segments. We also verified that NAMOA*$_{\text{div}}$ indeed finds a subset of the Pareto optimal set of solutions of the exact NAMOA*, although it was often a different subset than that obtained by simply filtering the entire solution set.

Analyzing the runtimes, we found out that the NAMOA*$_{\text{div}}$ is lacking in performance compared to the exact search with a filter applied at the end. The new algorithm was on average up to two times slower in solving the same problem instances, although there were some instances that were solved by NAMOA*$_{\text{div}}$ substantially faster. The slowdown was more pronounced when using the destination and comfort criteria, as opposed to the destination-time combination, due to the correlation difference of the criterion pairs. This observation renders the NAMOA*$_{\text{div}}$ algorithm inferior in its current implementation, but the performance gap is not so drastic that it would make future tweaking efforts worthless.

Perhaps even a category of problem instances gets identified that consistently achieves better results with the diversity-aware algorithm.

As we have pointed out several times, solving the multi-criteria shortest path problem is a difficult task, and current exact algorithms are relatively efficient only on moderately sized road networks. We tested the search algorithms on graphs with up to 700,000 edges, one representing the city of Prague and another representing New York City. Most of the problem instances were solved within the period of a few seconds, which is a decent result. However, we need to keep in mind that the runtimes are relatively low thanks to restricting ourselves to using only two criteria.

In order to improve the performance of the multi-criteria search, there currently does not seem to be a feasible way other than by applying approximations if we are willing to sacrifice the optimality of the solutions. An example of such an approximation could be the relaxation of the vector dominance checking by using epsilon-dominance instead. A parallelization of the search algorithms may bring a certain performance boost if we insist on exact solutions. In fact, there is more room for parallelization of NAMOA$^*_{\text{div}}$, as it performs batch operations such as the update of path differences when a new solution is discovered.

The multi-criteria search can definitely find application in route planning nonetheless. The hazmat transportation, for example, does not strictly require a real-time operation, as the routes are typically planned in advance. The cyclists, on the other hand, may be forgiving of the slight divergence from optimality in the results. Reducing the search results to a reasonable number of highly diverse route suggestions, however, often facilitates the ultimate decision making when looking for the most convenient route while taking multiple factors into account.

# Bibliography


[1] *Admiralty Manual of Navigation, Volume 1*, volume 45 of *BR Series*. Stationery Office, 1997.

[2] V. Akgün, E. Erkut, and R. Batta. On finding dissimilar paths. *European Journal of Operational Research*, 121(2):232–246, 2000.

[3] J. Arz, D. Luxen, and P. Sanders. Transit node routing reconsidered. In *Experimental Algorithms*, pages 55–66. Springer, 2013.

[4] H. Bast, D. Delling, A. Goldberg, M. Müller-Hannemann, T. Pajor, P. Sanders, D. Wagner, and R. F. Werneck. Route planning in transportation networks. *arXiv preprint arXiv:1504.05140*, 2015.

[5] H. Bast, S. Funke, and D. Matijevic. Ultrafast shortest-path queries via transit nodes. *The Shortest Path Problem: Ninth DIMACS Implementation Challenge*, 74:175–192, 2009.

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

[7] R. Bauer and D. Delling. SHARC: fast and robust unidirectional routing. *Journal of Experimental Algorithmics (JEA)*, 14:4, 2009.

[8] R. Bauer, D. Delling, P. Sanders, D. Schieferdecker, D. Schultes, and D. Wagner. Combining hierarchical and goal-directed speed-up techniques for Dijkstra's algorithm. *Journal of Experimental Algorithmics (JEA)*, 15:2–3, 2010.

[9] R. Bellman. On a routing problem. Technical report, DTIC Document, 1956.

[10] M. Caramia, S. Giordani, and A. Iovanella. On the selection of k routes in multiobjective hazmat route planning. *IMA Journal of Management Mathematics*, page dpp017, 2009.

[11] E. Cohen, E. Halperin, H. Kaplan, and U. Zwick. Reachability and distance queries via 2-hop labels. *SIAM Journal on Computing*, 32(5):1338–1355, 2003.

[12] P. Dell'Olmo, M. Gentili, and A. Scozzari. On finding dissimilar Pareto-optimal paths. *European Journal of Operational Research*, 162(1):70–82, 2005.

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

[14] L. R. Ford. Network flow theory. 1956.

[15] C. Gavoille, D. Peleg, S. Pérennes, and R. Raz. Distance labeling in graphs. *Journal of Algorithms*, 53(1):85–112, 2004.

[16] R. Geisberger, P. Sanders, D. Schultes, and C. Vetter. Exact routing in large road networks using contraction hierarchies. *Transportation Science*, 46(3):388–404, 2012.

[17] A. V. Goldberg and C. Harrelson. Computing the shortest path: A search meets graph theory. In *Proceedings of the sixteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 156–165. Society for Industrial and Applied Mathematics, 2005.

[18] P. Hansen. Bicriterion path problems. In *Multiple criteria decision making theory and application*, pages 109–127. Springer, 1980.

[19] M. Haqqani, X. Li, and X. Yu. A multi-objective A* search based on non-dominated sorting. In *Simulated Evolution and Learning*, pages 228–238. Springer, 2014.

[20] P. E. Hart, N. J. Nilsson, and B. Raphael. A formal basis for the heuristic determination of minimum cost paths. *Systems Science and Cybernetics, IEEE Transactions on*, 4(2):100–107, 1968.

[21] M. Hilger, E. Köhler, R. H. Möhring, and H. Schilling. Fast point-to-point shortest path computations with arc-flags. *The Shortest Path Problem: Ninth DIMACS Implementation Challenge*, 74:41–72, 2009.

[22] M. J. Kuby. Programming models for facility dispersion: The p-dispersion and maxisum dispersion problems. *Geographical Analysis*, 19(4):315–329, 1987.

[23] U. Lauther. An experimental evaluation of point-to-point shortest path calculation on road networks with precalculated edge-flags. *The Shortest Path Problem: Ninth DIMACS Implementation Challenge*, 74:19–40, 2006.

[24] M. Levandowsky and D. Winter. Distance between sets. *Nature*, 234(5323):34–35, 1971.

[25] E. Machuca and L. Mandow. Multiobjective heuristic search in road maps. *Expert Systems with Applications*, 39(7):6435–6445, 2012.

[26] E. Machuca, L. Mandow, J. P. de la Cruz, and A. Ruiz-Sepúlveda. A comparison of heuristic best-first algorithms for bicriterion shortest path problems. *European Journal of Operational Research*, 217(1):44–53, 2012.

[27] G. Mali, P. Michail, and C. Zaroliagis. Faster multiobjective heuristic search in road maps. In *Proceedings ICT*, volume 3, pages 67–72, 2012.

[28] L. Mandow and J. L. P. De La Cruz. Multiobjective A* search with consistent heuristics. *Journal of the ACM (JACM)*, 57(5):27, 2010.

[29] L. Mandow and J. P. De la Cruz. A new approach to multiobjective A* search. In *IJCAI*, pages 218–223. Citeseer, 2005.

[30] L. Mandow and J. P. De La Cruz. A memory-efficient search strategy for multiobjective shortest path problems. In *KI 2009: Advances in Artificial Intelligence*, pages 25–32. Springer, 2009.

[31] R. Martí, J. L. G. Velarde, and A. Duarte. Heuristics for the bi-objective path dissimilarity problem. *Computers & Operations Research*, 36(11):2905–2912, 2009.

[32] E. Q. V. Martins. On a multicriteria shortest path problem. *European Journal of Operational Research*, 16(2):236–245, 1984.

[33] M. Müller-Hannemann and K. Weihe. On the cardinality of the Pareto set in bicriteria shortest path problems. *Annals of Operations Research*, 147(1):269–286, 2006.

[34] J.-L. Pérez de la Cruz, L. Mandow, and E. Machuca. A case of pathology in multiobjective heuristic search. *Journal of Artificial Intelligence Research*, pages 717–732, 2013.

[35] F. J. Pulido, L. Mandow, and J. L. P. de la Cruz. Multiobjective shortest path problems with lexicographic goal-based preferences. *European Journal of Operational Research*, 239(1):89–101, 2014.

[36] A. Raith and M. Ehrgott. A comparison of solution strategies for biobjective shortest path problems. *Computers & Operations Research*, 36(4):1299–1331, 2009.

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

[38] P. Sanders and D. Schultes. Robust, almost constant time shortest-path queries in road networks. *9th DIMACS Implementation Challenge [1]*, 2006.

[39] A. J. Skriver and K. A. Andersen. A label correcting approach for solving bicriterion shortest-path problems. *Computers & Operations Research*, 27(6):507–524, 2000.

[40] B. S. Stewart and C. C. White III. Multiobjective A*. *Journal of the ACM (JACM)*, 38(4):775–814, 1991.

[41] C. T. Tung and K. L. Chew. A multicriteria pareto-optimal path algorithm. *European Journal of Operational Research*, 62(2):203–209, 1992.

# Appendix A

# Experiment Results

In this appendix you will find the complete results of our experiments for all 50 instances on the NYC road network. There are three tables that offer an overview of the numbers of solutions, the numbers of label expansions and the runtimes. Whenever there is a '–' instead of a value in the tables, it means that for that particular problem instance the search took more than 15 minutes and was terminated.

Table A.1: Sizes of the solution sets using various parameters for the search (NYC dataset).

| | Origin | Dest. | NAMOA* | | | NAMOA* with filter | | | NAMOA*$_{\text{div}}$ | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | authors' | ours | new $Q$ | 30% | 40% | 50% | 30% | 40% | 50% |
| 1 | 33502 | 163335 | 45 | 45 | 45 | 3 | 2 | 2 | 3 | 2 | 2 |
| 2 | 198561 | 195430 | 12 | 12 | 12 | 4 | 3 | 2 | 4 | 3 | 2 |
| 3 | 40851 | 4310 | 344 | 344 | 344 | 5 | 3 | 2 | 4 | 3 | 2 |
| 4 | 19103 | 95503 | 24 | 24 | 24 | 5 | 3 | 2 | 5 | 3 | 2 |
| 5 | 65190 | 57030 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 6 | 172882 | 189944 | 163 | 163 | 163 | 4 | 4 | 4 | 4 | 4 | 4 |
| 7 | 181176 | 151910 | 308 | 308 | 308 | 4 | 3 | 3 | 4 | 3 | 3 |
| 8 | 177414 | 103345 | 122 | 122 | 122 | 4 | 3 | 2 | 3 | 3 | 2 |
| 9 | 186166 | 71968 | 487 | 487 | 487 | 9 | 6 | 5 | 7 | 6 | 3 |
| 10 | 50616 | 76333 | 31 | 31 | 31 | 2 | 2 | 2 | 2 | 2 | 2 |
| 11 | 56699 | 159358 | 401 | 401 | 401 | 6 | 3 | 3 | 4 | 3 | 3 |
| 12 | 103987 | 175817 | 213 | 213 | 213 | 4 | 4 | 2 | 3 | 2 | 2 |
| 13 | 75533 | 165171 | 245 | 245 | 245 | 6 | 3 | 3 | 3 | 3 | 3 |
| 14 | 191865 | 72103 | 346 | 346 | 346 | 9 | 6 | 3 | 7 | 5 | 3 |
| 15 | 35170 | 237017 | 26 | 26 | 26 | 2 | 2 | 2 | 2 | 2 | 2 |
| 16 | 207442 | 156433 | 69 | 69 | 69 | 5 | 4 | 4 | 4 | 4 | 4 |
| 17 | 62306 | 134007 | 78 | 78 | 78 | 4 | 3 | 2 | 3 | 3 | 2 |
| 18 | 58427 | 135252 | 242 | 242 | 242 | 10 | 6 | 5 | 5 | 4 | 4 |
| 19 | 91985 | 200812 | 241 | 241 | 241 | 9 | 6 | 5 | 8 | 4 | 4 |
| 20 | 242644 | 163590 | 156 | 156 | 156 | 2 | 2 | 2 | 2 | 2 | 2 |
| 21 | 40180 | 100359 | 77 | 77 | 77 | 7 | 4 | 3 | 7 | 4 | 3 |
| 22 | 38497 | 207344 | 465 | 465 | 465 | 10 | 7 | 6 | 6 | 4 | 4 |
| 23 | 180834 | 83150 | 814 | 814 | 814 | 10 | 6 | 4 | 9 | 6 | 4 |
| 24 | 129948 | 7003 | 234 | 234 | 234 | 11 | 6 | 4 | 6 | 5 | 3 |
| 25 | 259195 | 173121 | 72 | 72 | 72 | 4 | 3 | 2 | 3 | 3 | 2 |
| 26 | 147806 | 136543 | 371 | 371 | 371 | 12 | 8 | 5 | 13 | 8 | 3 |
| 27 | 179874 | 57536 | 643 | 643 | 643 | 7 | 4 | 4 | 6 | 4 | 4 |
| 28 | 189934 | 31336 | 169 | 169 | 169 | 7 | 5 | 4 | 7 | 4 | 3 |
| 29 | 138263 | 253856 | 11 | 11 | 11 | 3 | 2 | 2 | 3 | 2 | 2 |
| 30 | 246144 | 166336 | 65 | 65 | 65 | 4 | 4 | 2 | 3 | 3 | 2 |
| 31 | 25610 | 143842 | 86 | 86 | 86 | 3 | 2 | 2 | 3 | 2 | 2 |
| 32 | 228779 | 167251 | 162 | 162 | 162 | 3 | 2 | 2 | 3 | 2 | 2 |
| 33 | 78936 | 34136 | 111 | 111 | 111 | 4 | 2 | 2 | 3 | 2 | 2 |
| 34 | 124173 | 138439 | 295 | 295 | 295 | 3 | 3 | 2 | 3 | 3 | 2 |
| 35 | 260563 | 233292 | 36 | 36 | 36 | 3 | 2 | 2 | 3 | 2 | 2 |
| 36 | 193168 | 66816 | 280 | 280 | 280 | 11 | 8 | 5 | 9 | 6 | 3 |
| 37 | 29432 | 29834 | 131 | 131 | 131 | 8 | 6 | 5 | 8 | 6 | 6 |
| 38 | 193241 | 144927 | 787 | 787 | 787 | 10 | 8 | 6 | 8 | 7 | 6 |
| 39 | 161522 | 171446 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 40 | 176910 | 109129 | 164 | 164 | 164 | 3 | 3 | 2 | 3 | 3 | 2 |
| 41 | 251416 | 53900 | 106 | 106 | 106 | 5 | 3 | 3 | 4 | 3 | 3 |
| 42 | 201505 | 262626 | 48 | 48 | 48 | 3 | 3 | 2 | 3 | 3 | 2 |
| 43 | 86937 | 190907 | 632 | 632 | 632 | 4 | 4 | 3 | 5 | 3 | 3 |
| 44 | 35252 | 18638 | 4 | 4 | 4 | 2 | 2 | 2 | 2 | 2 | 2 |
| 45 | 92562 | 65120 | 202 | 202 | 202 | 9 | 6 | 5 | 8 | 5 | 4 |
| 46 | 230423 | 2724 | 66 | 66 | 66 | 2 | 2 | 2 | 2 | 2 | 2 |
| 47 | 17285 | 92411 | 17 | 17 | 17 | 4 | 4 | 3 | 4 | 4 | 3 |
| 48 | 177037 | 199832 | 8 | 8 | 8 | 2 | 2 | 2 | 2 | 2 | 2 |
| 49 | 68330 | 206280 | 270 | 270 | 270 | 5 | 4 | 3 | 5 | 4 | 3 |
| 50 | 61414 | 50367 | 50 | 50 | 50 | 3 | 3 | 3 | 3 | 3 | 3 |
| Average | | | 198.62 | 198.62 | 198.62 | 5.22 | 3.76 | 2.98 | 4.46 | 3.40 | 2.74 |

Table A.2: Numbers of label expansions performed during the search (NYC dataset).

| | Origin | Dest. | NAMOA* | NAMOA* with filter | | | NAMOA*$_{div}$ | | |
|---|---|---|---|---|---|---|---|---|---|
| | | | | 30% | 40% | 50% | 30% | 40% | 50% |
| 1 | 33502 | 163335 | 10871 | 10871 | 10871 | 10871 | 8469 | 7950 | 7950 |
| 2 | 198561 | 195430 | 774 | 774 | 774 | 774 | 671 | 626 | 609 |
| 3 | 40851 | 4310 | 2354517 | 2354517 | 2354517 | 2354517 | 2227598 | 2081748 | 1929468 |
| 4 | 19103 | 95503 | 5131 | 5131 | 5131 | 5131 | 4702 | 4310 | 4036 |
| 5 | 65190 | 57030 | 19 | 19 | 19 | 19 | 0 | 0 | 0 |
| 6 | 172882 | 189944 | 197523 | 197523 | 197523 | 197523 | 137614 | 137614 | 137614 |
| 7 | 181176 | 151910 | 319665 | 319665 | 319665 | 319665 | 256911 | 235232 | 235232 |
| 8 | 177414 | 103345 | 177192 | 177192 | 177192 | 177192 | 122954 | 122954 | 73959 |
| 9 | 186166 | 71968 | 2399749 | 2399749 | 2399749 | 2399749 | 2315395 | 2181080 | 1387017 |
| 10 | 50616 | 76333 | 10660 | 10660 | 10660 | 10660 | 7312 | 7312 | 7312 |
| 11 | 56699 | 159358 | 432131 | 432131 | 432131 | 432131 | 362128 | 334014 | 334014 |
| 12 | 103987 | 175817 | 475049 | 475049 | 475049 | 475049 | 394340 | 206822 | 206822 |
| 13 | 75533 | 165171 | 565959 | 565959 | 565959 | 565959 | 299034 | 299034 | 299034 |
| 14 | 191865 | 72103 | 539446 | 539446 | 539446 | 539446 | 444455 | 339230 | 285230 |
| 15 | 35170 | 237017 | 7629 | 7629 | 7629 | 7629 | 6634 | 6634 | 6634 |
| 16 | 207442 | 156433 | 105868 | 105868 | 105868 | 105868 | 82771 | 82771 | 82771 |
| 17 | 62306 | 134007 | 132234 | 132234 | 132234 | 132234 | 96627 | 96627 | 77116 |
| 18 | 58427 | 135252 | 319729 | 319729 | 319729 | 319729 | 188356 | 174968 | 174968 |
| 19 | 91985 | 200812 | 423642 | 423642 | 423642 | 423642 | 322516 | 244002 | 244002 |
| 20 | 242644 | 163590 | 30785 | 30785 | 30785 | 30785 | 17956 | 17956 | 17956 |
| 21 | 40180 | 100359 | 22487 | 22487 | 22487 | 22487 | 20174 | 19140 | 19024 |
| 22 | 38497 | 207344 | 586507 | 586507 | 586507 | 586507 | 392156 | 345544 | 345544 |
| 23 | 180834 | 83150 | 3965136 | 3965136 | 3965136 | 3965136 | 3862763 | 3724778 | 3531859 |
| 24 | 129948 | 7003 | 784170 | 784170 | 784170 | 784170 | 754296 | 539752 | 321387 |
| 25 | 259195 | 173121 | 25382 | 25382 | 25382 | 25382 | 19965 | 19965 | 13469 |
| 26 | 147806 | 136543 | 243534 | 243534 | 243534 | 243534 | 234735 | 221592 | 165585 |
| 27 | 179874 | 57536 | 4177079 | 4177079 | 4177079 | 4177079 | 3817582 | 2898984 | 2898984 |
| 28 | 189934 | 31336 | 127754 | 127754 | 127754 | 127754 | 105257 | 102456 | 100990 |
| 29 | 138263 | 253856 | 593 | 593 | 593 | 593 | 463 | 333 | 333 |
| 30 | 246144 | 166336 | 13370 | 13370 | 13370 | 13370 | 7051 | 7051 | 5385 |
| 31 | 25610 | 143842 | 36106 | 36106 | 36106 | 36106 | 33576 | 20387 | 20387 |
| 32 | 228779 | 167251 | 62467 | 62467 | 62467 | 62467 | 44865 | 36057 | 36057 |
| 33 | 78936 | 34136 | 247022 | 247022 | 247022 | 247022 | 66015 | 60146 | 60146 |
| 34 | 124173 | 138439 | 533255 | 533255 | 533255 | 533255 | 308877 | 308877 | 180922 |
| 35 | 260563 | 233292 | 6862 | 6862 | 6862 | 6862 | 5203 | 4690 | 4690 |
| 36 | 193168 | 66816 | 521546 | 521546 | 521546 | 521546 | 420256 | 301999 | 208326 |
| 37 | 29432 | 29834 | 124970 | 124970 | 124970 | 124970 | 114910 | 107934 | 107934 |
| 38 | 193241 | 144927 | 467113 | 467113 | 467113 | 467113 | 270887 | 217913 | 177726 |
| 39 | 161522 | 171446 | 144 | 144 | 144 | 144 | 0 | 0 | 0 |
| 40 | 176910 | 109129 | 79006 | 79006 | 79006 | 79006 | 54022 | 54022 | 44752 |
| 41 | 251416 | 53900 | 92802 | 92802 | 92802 | 92802 | 65449 | 63170 | 63170 |
| 42 | 201505 | 262626 | 22330 | 22330 | 22330 | 22330 | 21532 | 21532 | 20655 |
| 43 | 86937 | 190907 | 4507771 | 4507771 | 4507771 | 4507771 | 4254752 | 4225641 | 4225641 |
| 44 | 35252 | 18638 | 359 | 359 | 359 | 359 | 297 | 297 | 297 |
| 45 | 92562 | 65120 | 189860 | 189860 | 189860 | 189860 | 153579 | 132714 | 113385 |
| 46 | 230423 | 2724 | 11112 | 11112 | 11112 | 11112 | 9039 | 9039 | 9039 |
| 47 | 17285 | 92411 | 8805 | 8805 | 8805 | 8805 | 8315 | 8315 | 7517 |
| 48 | 177037 | 199832 | 1761 | 1761 | 1761 | 1761 | 1508 | 1508 | 1508 |
| 49 | 68330 | 206280 | 409058 | 409058 | 409058 | 409058 | 292512 | 195264 | 158640 |
| 50 | 61414 | 50367 | 17869 | 17869 | 17869 | 17869 | 15449 | 15449 | 15449 |

Table A.3: Runtimes (in seconds) of the algorithms with various parameters (NYC dataset).

| | Origin | Dest. | NAMOA* | | | NAMOA* with filter | | | NAMOA*$_{\text{div}}$ | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | authors' | ours | new $Q$ | 30% | 40% | 50% | 30% | 40% | 50% |
| 1 | 33502 | 163335 | 9.22 | 0.55 | 0.52 | 0.51 | 0.47 | 0.48 | 0.60 | 0.56 | 0.55 |
| 2 | 198561 | 195430 | 0.34 | 0.06 | 0.07 | 0.06 | 0.06 | 0.06 | 0.06 | 0.07 | 0.04 |
| 3 | 40851 | 4310 | 525.87 | 30.03 | 82.72 | 37.67 | 32.60 | 34.25 | 89.11 | 74.91 | 74.38 |
| 4 | 19103 | 95503 | 4.98 | 0.30 | 0.32 | 0.29 | 0.29 | 0.31 | 0.34 | 0.33 | 0.30 |
| 5 | 65190 | 57030 | 0.09 | 0.05 | 0.05 | 0.04 | 0.05 | 0.05 | 0.05 | 0.04 | 0.06 |
| 6 | 172882 | 189944 | 44.17 | 1.26 | 2.71 | 1.37 | 1.38 | 1.28 | 3.12 | 3.32 | 3.05 |
| 7 | 181176 | 151910 | 89.46 | 2.63 | 4.58 | 2.36 | 2.35 | 2.52 | 8.10 | 5.30 | 5.65 |
| 8 | 177414 | 103345 | 30.72 | 1.50 | 4.02 | 1.40 | 1.39 | 1.64 | 3.16 | 3.06 | 1.99 |
| 9 | 186166 | 71968 | 862.22 | 71.95 | 101.69 | 62.87 | 59.79 | 65.54 | 185.90 | 159.09 | 72.88 |
| 10 | 50616 | 76333 | 7.27 | 0.37 | 0.37 | 0.37 | 0.32 | 0.37 | 0.34 | 0.38 | 0.41 |
| 11 | 56699 | 159358 | 114.02 | 6.36 | 11.33 | 6.48 | 5.61 | 5.95 | 12.09 | 10.36 | 10.50 |
| 12 | 103987 | 175817 | 77.33 | 6.42 | 16.91 | 5.97 | 6.37 | 6.81 | 16.11 | 6.26 | 5.87 |
| 13 | 75533 | 165171 | 129.05 | 6.09 | 11.28 | 5.65 | 5.16 | 5.75 | 11.90 | 9.51 | 12.70 |
| 14 | 191865 | 72103 | 128.17 | 4.07 | 9.82 | 8.65 | 5.17 | 4.97 | 23.76 | 18.23 | 9.85 |
| 15 | 35170 | 237017 | 5.31 | 0.24 | 0.31 | 0.25 | 0.26 | 0.24 | 0.31 | 0.29 | 0.34 |
| 16 | 207442 | 156433 | 15.08 | 0.35 | 0.93 | 0.34 | 0.36 | 0.35 | 1.24 | 1.17 | 1.29 |
| 17 | 62306 | 134007 | 22.05 | 1.26 | 2.65 | 1.31 | 1.42 | 1.18 | 2.05 | 1.88 | 1.59 |
| 18 | 58427 | 135252 | 76.63 | 2.59 | 5.06 | 3.90 | 3.18 | 3.19 | 6.41 | 5.47 | 5.39 |
| 19 | 91985 | 200812 | 97.36 | 3.44 | 9.26 | 4.33 | 3.78 | 4.88 | 15.13 | 6.01 | 6.97 |
| 20 | 242644 | 163590 | 12.83 | 0.62 | 0.79 | 0.58 | 0.73 | 0.62 | 0.72 | 0.70 | 0.72 |
| 21 | 40180 | 100359 | 5.54 | 0.22 | 0.26 | 0.24 | 0.26 | 0.24 | 0.37 | 0.34 | 0.33 |
| 22 | 38497 | 207344 | 199.53 | 7.16 | 11.82 | 7.76 | 6.78 | 9.41 | 16.00 | 12.07 | 13.09 |
| 23 | 180834 | 83150 | 1948.15 | 134.34 | 211.57 | 138.90 | 146.70 | 121.71 | 315.50 | 295.92 | 227.72 |
| 24 | 129948 | 7003 | 135.24 | 6.53 | 24.74 | 8.50 | 7.17 | 6.15 | 32.40 | 16.48 | 7.86 |
| 25 | 259195 | 173121 | 12.87 | 0.77 | 0.86 | 0.84 | 0.84 | 0.94 | 0.98 | 0.94 | 0.79 |
| 26 | 147806 | 136543 | 63.93 | 2.09 | 3.64 | 3.49 | 2.67 | 3.07 | 9.29 | 8.29 | 4.87 |
| 27 | 179874 | 57536 | 1495.73 | 99.53 | 201.64 | 126.75 | 130.80 | 145.94 | 274.87 | 155.75 | 155.14 |
| 28 | 189934 | 31336 | 25.34 | 0.95 | 2.55 | 1.55 | 1.07 | 1.27 | 3.19 | 2.57 | 2.65 |
| 29 | 138263 | 253856 | 0.64 | 0.05 | 0.06 | 0.05 | 0.06 | 0.05 | 0.05 | 0.09 | 0.09 |
| 30 | 246144 | 166336 | 6.84 | 0.34 | 0.43 | 0.38 | 0.40 | 0.36 | 0.41 | 0.46 | 0.38 |
| 31 | 25610 | 143842 | 10.71 | 0.42 | 0.67 | 0.53 | 0.45 | 0.40 | 1.04 | 0.69 | 0.57 |
| 32 | 228779 | 167251 | 18.6 | 0.92 | 1.05 | 0.88 | 0.92 | 0.84 | 1.78 | 1.75 | 1.76 |
| 33 | 78936 | 34136 | 39.86 | 1.64 | 3.66 | 1.58 | 1.52 | 1.41 | 1.89 | 1.43 | 1.56 |
| 34 | 124173 | 138439 | 108.12 | 6.16 | 14.78 | 6.83 | 5.33 | 5.94 | 8.64 | 9.02 | 4.44 |
| 35 | 260563 | 233292 | 3.51 | 0.18 | 0.22 | 0.18 | 0.17 | 0.17 | 0.20 | 0.21 | 0.20 |
| 36 | 193168 | 66816 | 94.74 | 3.82 | 11.15 | 7.21 | 5.05 | 4.09 | 21.55 | 12.09 | 7.89 |
| 37 | 29432 | 29834 | 20.5 | 0.88 | 1.61 | 1.17 | 0.95 | 0.90 | 3.09 | 3.01 | 2.92 |
| 38 | 193241 | 144927 | 209.09 | 8.81 | 12.86 | 16.43 | 14.41 | 11.78 | 18.31 | 11.96 | 7.04 |
| 39 | 161522 | 171446 | 0.8 | 0.06 | 0.05 | 0.05 | 0.06 | 0.06 | 0.05 | 0.05 | 0.05 |
| 40 | 176910 | 109129 | 23.21 | 1.11 | 1.40 | 1.21 | 1.23 | 1.10 | 1.90 | 1.89 | 1.55 |
| 41 | 251416 | 53900 | 19.09 | 0.94 | 1.53 | 0.99 | 0.87 | 1.02 | 1.98 | 1.58 | 1.61 |
| 42 | 201505 | 262626 | 7.32 | 0.38 | 0.53 | 0.35 | 0.35 | 0.32 | 0.54 | 0.54 | 0.56 |
| 43 | 86937 | 190907 | 1389.98 | 118.43 | 194.76 | 102.77 | 105.06 | 114.87 | 234.19 | 280.15 | 237.94 |
| 44 | 35252 | 18638 | 1.53 | 0.09 | 0.09 | 0.09 | 0.09 | 0.09 | 0.09 | 0.09 | 0.09 |
| 45 | 92562 | 65120 | 33.85 | 1.21 | 3.63 | 2.79 | 2.06 | 1.68 | 6.21 | 4.86 | 3.27 |
| 46 | 230423 | 2724 | 4.78 | 0.25 | 0.30 | 0.25 | 0.24 | 0.25 | 0.36 | 0.31 | 0.34 |
| 47 | 17285 | 92411 | 3.35 | 0.15 | 0.27 | 0.19 | 0.15 | 0.15 | 0.21 | 0.20 | 0.21 |
| 48 | 177037 | 199832 | 5.44 | 0.28 | 0.31 | 0.29 | 0.31 | 0.29 | 0.35 | 0.30 | 0.29 |
| 49 | 68330 | 206280 | 114.12 | 3.28 | 6.35 | 3.70 | 3.00 | 2.93 | 11.23 | 5.83 | 4.32 |
| 50 | 61414 | 50367 | 11.32 | 0.61 | 0.79 | 0.63 | 0.63 | 0.58 | 0.76 | 0.75 | 0.87 |
| Average | | | 165.32 | 10.83 | 19.58 | 11.62 | 11.41 | 11.57 | 26.96 | 22.73 | 18.10 |

Table A.4: Sizes of the solution sets using various parameters for the search (PRG dataset).

| | Origin | Dest. | NAMOA* | NAMOA* with filter | | | $NAMOA^*_{div}$ | | |
|---|---|---|---|---|---|---|---|---|---|
| | | | | 30% | 40% | 50% | 30% | 40% | 50% |
| 1 | 103429 | 107165 | 801 | 8 | 5 | 5 | 8 | 4 | 4 |
| 2 | 27625 | 26432 | 732 | 8 | 7 | 5 | 9 | 7 | 5 |
| 3 | 27912 | 89965 | 1397 | 5 | 4 | 3 | 4 | 4 | 3 |
| 4 | 92297 | 36432 | 885 | 7 | 6 | 5 | 7 | 6 | 4 |
| 5 | 18160 | 84281 | 57 | 7 | 6 | 3 | 7 | 6 | 3 |
| 6 | 17458 | 20024 | 188 | 9 | 6 | 4 | 9 | 6 | 4 |
| 7 | 66914 | 9549 | 1774 | 7 | 3 | 2 | 7 | 3 | 2 |
| 8 | 81734 | 105542 | 231 | 4 | 3 | 3 | 5 | 3 | 3 |
| 9 | 13085 | 32485 | 373 | 6 | 5 | 4 | 6 | 5 | 4 |
| 10 | 15649 | 49846 | 1005 | 9 | 5 | 5 | 7 | 5 | 3 |
| 11 | 6236 | 85706 | 214 | 8 | 6 | 6 | 8 | 6 | 6 |
| 12 | 54128 | 26394 | 74 | 6 | 4 | 4 | 6 | 4 | 4 |
| 13 | 107731 | 101041 | 347 | 6 | 6 | 3 | 6 | 5 | 3 |
| 14 | 45195 | 104737 | 430 | 6 | 4 | 3 | 6 | 5 | 2 |
| 15 | 55635 | 105308 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 16 | 38648 | 97599 | 840 | 7 | 4 | 3 | 6 | 4 | 3 |
| 17 | 24386 | 48526 | 23 | 3 | 2 | 2 | 3 | 2 | 2 |
| 18 | 97404 | 73342 | 110 | 5 | 4 | 4 | 5 | 4 | 2 |
| 19 | 76986 | 63946 | 336 | 5 | 4 | 3 | 4 | 4 | 3 |
| 20 | 74057 | 35482 | 82 | 4 | 4 | 3 | 4 | 4 | 3 |
| 21 | 106775 | 52687 | 23 | 4 | 3 | 2 | 3 | 2 | 2 |
| 22 | 35396 | 31745 | 29 | 4 | 3 | 3 | 4 | 3 | 3 |
| 23 | 73824 | 75324 | 1496 | 9 | 6 | 4 | 8 | 5 | 3 |
| 24 | 84789 | 61539 | 45 | 3 | 3 | 2 | 3 | 3 | 2 |
| 25 | 9284 | 101419 | 266 | 11 | 9 | 6 | 9 | 7 | 6 |
| 26 | 54686 | 7840 | 1138 | 8 | 6 | 4 | 7 | 5 | 4 |
| 27 | 346 | 6895 | 115 | 7 | 4 | 3 | 7 | 4 | 3 |
| 28 | 13017 | 10384 | 165 | 6 | 4 | 3 | 5 | 4 | 3 |
| 29 | 12431 | 30085 | 282 | 9 | 7 | 5 | 8 | 5 | 4 |
| 30 | 7979 | 43212 | 436 | 7 | 6 | 4 | 6 | 5 | 4 |
| 31 | 16110 | 7930 | 74 | 3 | 3 | 2 | 3 | 3 | 2 |
| 32 | 70404 | 66089 | 73 | 4 | 3 | 2 | 4 | 3 | 2 |
| 33 | 87069 | 17496 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 34 | 1731 | 106815 | 72 | 3 | 3 | 3 | 3 | 3 | 3 |
| 35 | 47179 | 17472 | 1545 | 9 | 5 | 5 | 8 | 5 | 5 |
| 36 | 38480 | 67828 | 3659 | - | - | - | - | - | - |
| 37 | 72187 | 31209 | 83 | 3 | 2 | 2 | 3 | 2 | 2 |
| 38 | 106609 | 9431 | 353 | 8 | 6 | 4 | 8 | 5 | 4 |
| 39 | 53634 | 41879 | 465 | 7 | 5 | 3 | 5 | 5 | 3 |
| 40 | 2068 | 73787 | 839 | 6 | 4 | 3 | 6 | 4 | 3 |
| 41 | 34583 | 50919 | 355 | 6 | 6 | 5 | 6 | 6 | 5 |
| 42 | 4087 | 48156 | 159 | 8 | 6 | 4 | 8 | 6 | 4 |
| 43 | 108248 | 38197 | 179 | 3 | 3 | 2 | 3 | 3 | 2 |
| 44 | 83952 | 16603 | 9 | 2 | 2 | 2 | 2 | 2 | 2 |
| 45 | 74602 | 15450 | 1002 | 5 | 4 | 3 | 5 | 4 | 3 |
| 46 | 108109 | 51919 | 18 | 3 | 2 | 2 | 3 | 2 | 2 |
| 47 | 26057 | 53072 | 1406 | 7 | 6 | 4 | 6 | 5 | 4 |
| 48 | 7760 | 78446 | 26 | 8 | 6 | 5 | 9 | 6 | 5 |
| 49 | 31095 | 93191 | 341 | 6 | 5 | 4 | 6 | 5 | 4 |
| 50 | 38141 | 60189 | 9 | 2 | 2 | 2 | 2 | 2 | 2 |
| Average | | | 491.26 | 5.78 | 4.37 | 3.37 | 5.49 | 4.14 | 3.18 |

Table A.5: Numbers of label expansions performed during the search (PRG dataset).

| | Origin | Dest. | NAMOA* | NAMOA* with filter | | | NAMOA*$_{div}$ | | |
|---|---|---|---|---|---|---|---|---|---|
| | | | | 30% | 40% | 50% | 30% | 40% | 50% |
| 1 | 103429 | 107165 | 1852113 | 1852113 | 1852113 | 1852113 | 1640893 | 1430479 | 1430479 |
| 2 | 27625 | 26432 | 1856819 | 1856819 | 1856819 | 1856819 | 1800962 | 1772838 | 1603802 |
| 3 | 27912 | 89965 | 6100784 | 6100784 | 6100784 | 6100784 | 5894165 | 5894165 | 4746645 |
| 4 | 92297 | 36432 | 1513010 | 1513010 | 1513010 | 1513010 | 1430572 | 1420528 | 1342375 |
| 5 | 18160 | 84281 | 10318 | 10318 | 10318 | 10318 | 9911 | 9767 | 9487 |
| 6 | 17458 | 20024 | 186636 | 186636 | 186636 | 186636 | 172753 | 141211 | 77403 |
| 7 | 66914 | 9549 | 7741839 | 7741839 | 7741839 | 7741839 | 7477443 | 7339151 | 6994676 |
| 8 | 81734 | 105542 | 203614 | 203614 | 203614 | 203614 | 149203 | 136784 | 136784 |
| 9 | 13085 | 32485 | 160562 | 160562 | 160562 | 160562 | 147397 | 143427 | 129903 |
| 10 | 15649 | 49846 | 1200251 | 1200251 | 1200251 | 1200251 | 862280 | 544262 | 260242 |
| 11 | 6236 | 85706 | 124463 | 124463 | 124463 | 124463 | 115342 | 114212 | 114212 |
| 12 | 54128 | 26394 | 13489 | 13489 | 13489 | 13489 | 11443 | 10619 | 10619 |
| 13 | 107731 | 101041 | 456146 | 456146 | 456146 | 456146 | 438388 | 435761 | 431967 |
| 14 | 45195 | 104737 | 476861 | 476861 | 476861 | 476861 | 218954 | 171675 | 34681 |
| 15 | 55635 | 105308 | 16 | 16 | 16 | 16 | 0 | 0 | 0 |
| 16 | 38648 | 97599 | 3367624 | 3367624 | 3367624 | 3367624 | 2715135 | 2279626 | 1814000 |
| 17 | 24386 | 48526 | 1246 | 1246 | 1246 | 1246 | 1136 | 1024 | 1024 |
| 18 | 97404 | 73342 | 22509 | 22509 | 22509 | 22509 | 20482 | 17312 | 15316 |
| 19 | 76986 | 63946 | 357130 | 357130 | 357130 | 357130 | 276853 | 276853 | 124663 |
| 20 | 74057 | 35482 | 32572 | 32572 | 32572 | 32572 | 30924 | 30924 | 29770 |
| 21 | 106775 | 52687 | 4015 | 4015 | 4015 | 4015 | 2526 | 2470 | 2470 |
| 22 | 35396 | 31745 | 8093 | 8093 | 8093 | 8093 | 7539 | 7407 | 7407 |
| 23 | 73824 | 75324 | 3770577 | 3770577 | 3770577 | 3770577 | 3495357 | 3304800 | 3018644 |
| 24 | 84789 | 61539 | 8598 | 8598 | 8598 | 8598 | 7264 | 7264 | 4852 |
| 25 | 9284 | 101419 | 198252 | 198252 | 198252 | 198252 | 164492 | 141931 | 139304 |
| 26 | 54686 | 7840 | 3428293 | 3428293 | 3428293 | 3428293 | 2268502 | 1382278 | 852984 |
| 27 | 346 | 6895 | 69408 | 69408 | 69408 | 69408 | 61154 | 56677 | 49208 |
| 28 | 13017 | 10384 | 52088 | 52088 | 52088 | 52088 | 39770 | 36299 | 12174 |
| 29 | 12431 | 30085 | 360663 | 360663 | 360663 | 360663 | 307146 | 280084 | 259623 |
| 30 | 7979 | 43212 | 143710 | 143710 | 143710 | 143710 | 108064 | 81981 | 67634 |
| 31 | 16110 | 7930 | 32615 | 32615 | 32615 | 32615 | 16938 | 16938 | 5487 |
| 32 | 70404 | 66089 | 15690 | 15690 | 15690 | 15690 | 15062 | 14865 | 13974 |
| 33 | 87069 | 17496 | 49 | 49 | 49 | 49 | 0 | 0 | 0 |
| 34 | 1731 | 106815 | 11045 | 11045 | 11045 | 11045 | 9458 | 9458 | 9458 |
| 35 | 47179 | 17472 | 3577342 | 3577342 | 3577342 | 3577342 | 3540308 | 3508581 | 3508581 |
| 36 | 38480 | 67828 | - | - | - | - | - | - | - |
| 37 | 72187 | 31209 | 22324 | 22324 | 22324 | 22324 | 16133 | 9842 | 9842 |
| 38 | 106609 | 9431 | 106999 | 106999 | 106999 | 106999 | 99286 | 85811 | 79808 |
| 39 | 53634 | 41879 | 690161 | 690161 | 690161 | 690161 | 510835 | 510835 | 141363 |
| 40 | 2068 | 73787 | 1018480 | 1018480 | 1018480 | 1018480 | 1004731 | 999416 | 988858 |
| 41 | 34583 | 50919 | 787982 | 787982 | 787982 | 787982 | 771170 | 771170 | 744262 |
| 42 | 4087 | 48156 | 127681 | 127681 | 127681 | 127681 | 119788 | 113480 | 82039 |
| 43 | 108248 | 38197 | 26513 | 26513 | 26513 | 26513 | 24875 | 24875 | 20988 |
| 44 | 83952 | 16603 | 671 | 671 | 671 | 671 | 543 | 543 | 543 |
| 45 | 74602 | 15450 | 1914584 | 1914584 | 1914584 | 1914584 | 1595373 | 1567763 | 1529370 |
| 46 | 108109 | 51919 | 475 | 475 | 475 | 475 | 382 | 336 | 336 |
| 47 | 26057 | 53072 | 6814719 | 6814719 | 6814719 | 6814719 | 4658219 | 4118835 | 3335151 |
| 48 | 7760 | 78446 | 2341 | 2341 | 2341 | 2341 | 2069 | 1948 | 1788 |
| 49 | 31095 | 93191 | 414957 | 414957 | 414957 | 414957 | 399919 | 393221 | 374552 |
| 50 | 38141 | 60189 | 769 | 769 | 769 | 769 | 615 | 615 | 615 |

Table A.6: Runtimes (in seconds) of the algorithms with various parameters (PRG dataset).

| | Origin | Dest. | NAMOA* with filter | | | NAMOA*$_{div}$ | | |
|---|---|---|---|---|---|---|---|---|
| | | | 30% | 40% | 50% | 30% | 40% | 50% |
| 1 | 103429 | 107165 | 32.34 | 34.62 | 30.09 | 125.36 | 87.88 | 81.70 |
| 2 | 27625 | 26432 | 33.66 | 34.21 | 29.27 | 89.61 | 72.76 | 58.55 |
| 3 | 27912 | 89965 | 260.38 | 241.53 | 232.64 | 433.68 | 436.79 | 270.74 |
| 4 | 92297 | 36432 | 27.55 | 23.33 | 25.17 | 56.55 | 60.64 | 46.51 |
| 5 | 18160 | 84281 | 0.13 | 0.15 | 0.11 | 0.15 | 0.16 | 0.15 |
| 6 | 17458 | 20024 | 1.55 | 0.95 | 0.96 | 5.23 | 3.33 | 1.21 |
| 7 | 66914 | 9549 | 342.81 | 336.52 | 387.00 | 868.88 | 792.32 | 666.95 |
| 8 | 81734 | 105542 | 0.98 | 0.95 | 1.03 | 2.61 | 2.23 | 2.69 |
| 9 | 13085 | 32485 | 1.43 | 1.45 | 1.18 | 4.51 | 3.78 | 3.02 |
| 10 | 15649 | 49846 | 35.24 | 28.88 | 29.60 | 59.51 | 19.79 | 5.34 |
| 11 | 6236 | 85706 | 1.03 | 0.79 | 0.76 | 2.79 | 2.96 | 2.74 |
| 12 | 54128 | 26394 | 0.16 | 0.15 | 0.13 | 0.19 | 0.23 | 0.17 |
| 13 | 107731 | 101041 | 3.49 | 3.29 | 2.73 | 12.42 | 20.47 | 10.94 |
| 14 | 45195 | 104737 | 4.58 | 3.59 | 3.25 | 7.72 | 4.62 | 0.84 |
| 15 | 55635 | 105308 | 0.02 | 0.01 | 0.01 | 0.01 | 0.01 | 0.01 |
| 16 | 38648 | 97599 | 94.04 | 82.31 | 74.11 | 129.00 | 94.55 | 62.11 |
| 17 | 24386 | 48526 | 0.08 | 0.07 | 0.07 | 0.08 | 0.08 | 0.08 |
| 18 | 97404 | 73342 | 0.27 | 0.28 | 0.24 | 0.48 | 0.38 | 0.36 |
| 19 | 76986 | 63946 | 2.66 | 2.54 | 2.36 | 7.06 | 6.45 | 2.26 |
| 20 | 74057 | 35482 | 0.28 | 0.29 | 0.24 | 0.42 | 0.40 | 0.40 |
| 21 | 106775 | 52687 | 0.15 | 0.14 | 0.13 | 0.12 | 0.12 | 0.11 |
| 22 | 35396 | 31745 | 0.18 | 0.15 | 0.16 | 0.22 | 0.26 | 0.15 |
| 23 | 73824 | 75324 | 196.90 | 163.28 | 150.24 | 666.86 | 522.49 | 387.62 |
| 24 | 84789 | 61539 | 0.15 | 0.15 | 0.16 | 0.16 | 0.27 | 0.13 |
| 25 | 9284 | 101419 | 2.15 | 1.55 | 1.16 | 4.48 | 2.68 | 2.69 |
| 26 | 54686 | 7840 | 114.99 | 100.18 | 105.52 | 126.62 | 45.17 | 24.84 |
| 27 | 346 | 6895 | 0.48 | 0.36 | 0.34 | 0.91 | 0.84 | 0.72 |
| 28 | 13017 | 10384 | 0.40 | 0.31 | 0.32 | 0.60 | 0.54 | 0.24 |
| 29 | 12431 | 30085 | 2.45 | 2.05 | 1.96 | 6.58 | 5.01 | 4.05 |
| 30 | 7979 | 43212 | 1.57 | 1.26 | 1.09 | 2.35 | 1.70 | 1.09 |
| 31 | 16110 | 7930 | 0.27 | 0.29 | 0.33 | 0.35 | 0.35 | 0.26 |
| 32 | 70404 | 66089 | 0.20 | 0.16 | 0.16 | 0.32 | 0.29 | 0.27 |
| 33 | 87069 | 17496 | 0.09 | 0.09 | 0.09 | 0.10 | 0.09 | 0.15 |
| 34 | 1731 | 106815 | 0.15 | 0.14 | 0.12 | 0.14 | 0.15 | 0.15 |
| 35 | 47179 | 17472 | 177.05 | 157.89 | 158.80 | 449.19 | 430.84 | 425.15 |
| 36 | 38480 | 67828 | - | - | - | - | - | - |
| 37 | 72187 | 31209 | 0.21 | 0.19 | 0.19 | 0.34 | 0.24 | 0.32 |
| 38 | 106609 | 9431 | 1.24 | 0.97 | 0.92 | 2.93 | 1.95 | 1.58 |
| 39 | 53634 | 41879 | 8.47 | 7.26 | 7.75 | 16.69 | 16.15 | 3.10 |
| 40 | 2068 | 73787 | 15.03 | 12.65 | 12.80 | 58.04 | 54.38 | 49.44 |
| 41 | 34583 | 50919 | 5.48 | 4.94 | 5.29 | 16.42 | 16.11 | 24.32 |
| 42 | 4087 | 48156 | 0.71 | 0.62 | 0.60 | 2.36 | 1.97 | 1.22 |
| 43 | 108248 | 38197 | 0.26 | 0.24 | 0.27 | 1.13 | 0.95 | 0.78 |
| 44 | 83952 | 16603 | 0.04 | 0.04 | 0.04 | 0.04 | 0.04 | 0.04 |
| 45 | 74602 | 15450 | 36.46 | 36.24 | 41.21 | 111.54 | 101.89 | 95.44 |
| 46 | 108109 | 51919 | 0.02 | 0.02 | 0.02 | 0.02 | 0.02 | 0.02 |
| 47 | 26057 | 53072 | 250.51 | 242.23 | 255.83 | 319.97 | 218.31 | 133.26 |
| 48 | 7760 | 78446 | 0.09 | 0.07 | 0.08 | 0.08 | 0.08 | 0.14 |
| 49 | 31095 | 93191 | 3.06 | 2.66 | 2.62 | 9.99 | 8.50 | 6.69 |
| 50 | 38141 | 60189 | 0.11 | 0.11 | 0.12 | 0.12 | 0.13 | 0.21 |
| Average | | | 33.91 | 31.27 | 32.03 | 73.57 | 62.07 | 48.59 |

# Appendix B

# CD Contents

- `readme.txt` ..... brief description of the CD's contents
- `src`
  - `impl` ..... directory with the implementation source code
    - `src`
      - `main` ..... source codes of the search algorithms
      - `test`
        - `java` ..... source codes of the graph building and the testing routines
        - `resources` ..... road network data
    - `visualization` ..... directory with JavaScript libraries
  - `thesis` ..... directory with the LaTeX source code of the thesis
- `text`
  - `DP_Juraska_Juraj_2016.pdf` ..... thesis text in PDF format