Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Distributed Computing

Fast Route Finding

Semester Thesis

Karolis Martinkus
mkarolis@ethz.ch

Distributed Computing Group
Computer Engineering and Networks Laboratory
ETH Zürich

Supervisors:
Aryaz Eghbali
Prof. Dr. Roger Wattenhofer

January 14, 2020

Abstract

In recent years various algorithms for shortest path queries where proposed. Out of all the proposed methods Hub Labelling (variant of 2-hop labelling) stood out for its superb query performance. Hub Labelling algorithms compute label for each vertex that holds all hubs through which any destination in the graph can be reached from that vertex. We implement a state of the art Hub Labelling framework proposed by Delling et at. [1]. We modify this algorithm to allow for recursive shortest path recovery that is entirely label based. We use this algorithm to study where hubs are placed on the road networks and how this placement correlates with real world road usage data, namely taxi trips. We discover that the most important hubs selected by the algorithm do not correspond well to the real world data. However, time graph hubs come closer to it than distance graph hubs. We also look into how well the labelling could be used to recover an alternative path. Our proposed naive method looks promising for short distance alternative paths but needs some further improvement.

Contents

Abstract i

1 Introduction 1

1.1 Contraction Hierarchies . . . 1
1.2 Hub Labelling . . . 2
1.3 Hierarchical Hub Labelling . . . 4
1.4 General framework . . . 5

2 Algorithm 6

2.1 Pruned Labelling . . . 6
2.2 Vertex ordering . . . 7
2.3 Distance and path queries . . . 8
2.3.1 Distance query . . . 8
2.3.2 Path query . . . 8
2.3.3 Optimised distance query for Pruned Labelling and pruned tree sampling . . . 9

3 Data 10

3.1 Graphs . . . 10
3.2 Taxi trips . . . 11
3.3 Taxi traces . . . 12
3.3.1 Fast map matching . . . 12

4 Results 14

4.1 Importance of a good ordering . . . 14
4.2 Scaling to larger graphs . . . 15
4.3 The impact of graph metric . . . 16
4.4 Unequal importance of shortest paths . . . 20
4.5 Hub importance compared to real world traffic . . . 22

4.6 Performance of recursive shortest path queries 25
4.7 Recovering alternative path from the labelling 25

5 Related work **33**

6 Conclusions **34**

7 Future work **35**

Bibliography **36**

Chapter 1

Introduction

Answering shortest path queries is important on wide range of tasks. Probably one of the most common ones is shortest path finding on a road network. An obvious baseline for query performance is Dijkstra's algorithm. Although it is efficient in general case, when answering long distance queries on continent sized road networks it can take seconds. To reach better performance variety of algorithms were proposed. The one's focusing on road networks specifically (i.e. Contraction Hierarchies [2], Transit Node Routing [3, 4], Pruned Highway Labelling [5], Hub Labelling [6]) tend to utilise the fact that shortest paths over long distances use the same few connections - highways. As highlighted by Bast, Delling, Goldberg, Müller-Hannemann, Pajor, Sanders, Wagner and Werneck in their *Route planning in transportation networks* survey [7] Hub Labelling algorithms offer the best query performance. We will now briefly discuss the history of Hub Labelling algorithms. It is known that Hub Labelling and its Hierarchical Hub Labelling variant that we will be discussing here are NP-Complete as shown correspondingly by Mathias Weller [8] and Babenko, Goldberg, Kaplan, Savchenko, and Weller [9]. While there was previous work on Hub Labelling algorithms, for example $O(log(n))$ optimal hub labelling by Cohen, Halperin, Kaplan and Zwick [10] they were computationally infeasible on larger graphs (i.e. algorithm by Cohen et al. requires precomputing all pairs shortest paths). Here we will be focusing on more performant approaches to Hub Labelling.

1.1 Contraction Hierarchies

Contraction Hierarchies [2] as proposed by Geisberger, Sanders, Schultes, and Delling is a popular shortest path algorithm that uses pre-processing to make shortest path queries orders of magnitude faster than Dijkstra on road networks. This algorithm depends on introducing shortcuts to the graph during pre-processing. Shortcut is a new edge *edge(u,v)* that has the length equal to the distance *dist(u,v)* between v and u in the original graph. Intuition is that these shortcuts are also intended to utilise the fact that long distance paths use similar segments (highways) and these can be replaced by shortcuts. In Contraction

Hierarchies nodes are processed in a specific (total) order and when a node is processed it is removed from the graph and shortcuts are introduced in the graph between its neighbours for which the shortest path between them used the now removed vertex. Vertices are ordered by their importance in the graph. Least important vertices are contracted first. Importance is estimated using linear combination of difference in number of edges if this vertex was replaced by shortcuts (shortcuted) and number of already contracted neighbours. This estimate can be improved using some global importance measures i.e. betweenness centrality. After contracting all the $n-1$ vertices we have our set of shortcuts E^+. We then make two pre-processed graph instances: upward graph $G^\uparrow = (V, E \cup E^\uparrow)$ that includes edges and shortcuts from lower importance vertices to higher importance vertices $E^\uparrow = \{(u,v) \in E \cup E^+ : \mathit{importance}(u) < \mathit{importance}(v)\}$ and a downward graph $G^\uparrow = (V, E \cup E^\downarrow)$ that includes shortcuts and edges from higher importance vertices to lower importance vertices $E^\downarrow = \{(u,v) \in E \cup E^+ : \mathit{importance}(u) > \mathit{importance}(v)\}$.

To answer a s-t query a modified bidirectional Dijkstra shortest path search is performed. Forward search is performed in $G^\uparrow$ and a backward search is performed in $G^\downarrow$. Geisberger et al. [2] showed that *iff* there exists a shortest path between s and t in the original graph, then both searches will eventually reach node u that has the highest importance of all the nodes in the shortest s-t path. Then the path (distance) can be recovered by concatenating (summing) forward and backward search results. As discussed in the survey by Bast et al. [7] Contraction Hierarchies (CH) has very good query time while only requiring short pre-processing. Note that CH always returns the correct shortest path, but by finding better ordering one can reduce the pre-processing and query time as well as number of shortcuts introduced dramatically.

1.2 Hub Labelling

Abraham, Fiat, Goldberg and Werneck [11] proposed a concept of highway dimensions to theoretically explain the good performance achieved by Contraction Hierarchies and other algorithms (Transit Node Routing [3, 4] and Reach [12]).

Definition 1.1 (Highway dimension)**.** Given a weighted graph $G = (V, E)$, its highway dimension is the smallest integer h such that

$$\forall r \in \Re^+, \forall u \in V, \exists S \subseteq B_{u,4r}, |S| \leq h \text{ such that}$$
$$\forall v, w \in B_{u,4r}, P(v,w) \cap S \neq \emptyset \text{ if } |P(v,w)| > r \text{ and } P(v,w) \subseteq B_{u,4r}$$

Definition 1.1 states that for every r and every ball $B_{u,4r}$ of radius $4r$ a small set of vertices (at most size h) covers all shortest paths of length greater than r that are inside the ball. This directly implies that there should be a shortest

path cover that for each vertex would cover all the paths originating at it and the intersection of such cover and the ball around the vertex would be small if h is small. See [11] for a more detailed proof. Abraham et al. [11] conjecture that for road networks h is small. Abraham et al. [11] showed that Contraction Hierarchies run in $O((hlog(h)log(D))^2)$. Also the theory they developed implied that there should exist a hub labelling of size $O(hlog(h)log(D))$. Where h is the highway dimension of the graph and D is the diameter. We will see in the next chapter that distance query runs in linear time in the label size. This makes Hub Labelling faster than Contraction Hierarchies with theoretical guarantees.

In a follow up work Abraham, Delling, Goldberg and Werneck [6] developed an implementation of Hub Labelling that as predicted by the theoretical work of Abraham et al. [11] outperformed previous methods. Pre-processing stage computes two labels for each vertex v. A forward label L_f that holds hubs on all outgoing shortest paths and a reverse label L_r that holds hubs on all incoming shortest paths. These labels should have a *cover property*: for any two vertices s, t $L_f(s) \cap L_r(t)$ contains a vertex u on a $s-t$ shortest path. Also distance to the hub recorded in the label ($dist_{L_f(s)}(u)$ and $dist_{L_r(t)}(u)$) matches shortest path distance from s to u and from u to t. Recovering shortest path distance is then straight forward. We just need to find the hub shared between the vertices that minimizes the total distance. *Cover property* guarantees that that there exists a hub on the shortest path between any two vertices.

Abraham et al. [6] propose an implementation that efficiently constructs such a labelling by making use of Contraction Hierarchies pre-processing. They use a relaxation of labels that they call *superlabels*. Distance recorded in *superlabels* does not need to be equal to the real distance. It is only enforced that $dist_{L_v)}(w) \geq dist(v, w)$. The cover property for *superlabels* defines that for the shortest path between s and t distances $dist_{L_f(s)}(u)$ and $dist_{L_r(t)}(u)$ to the shared hub u on the shortest path must match the exact distances. Because of this query results stay valid. The labelling for v is then defined as all the vertices visited by Contraction Hierarchies search originating at v for $L_f(v)$ and all the vertices visited by Contraction Hierarchies search terminating at v for $L_r(v)$. As shown by Geisberger et al. [2] the Contraction Hierarchies search guarantees that the vertex on the shortest path to (from) any destination will be visited. Abraham et al. [6] found that labels created in such a manner on the road network of Western Europe have on average 500 hubs in them. To reduce the label size further they prune the *superlabels* by deleting vertices w from a label $L_f(v)$ with $dist_{L_f(v)}(w) > dist(v, w)$. The same can be done for the reverse labels. As pruning does not delete hubs from the label that have exact distance, the cover property remains satisfied, and labelling is still valid.

1.3 Hierarchical Hub Labelling

Abraham, Delling, Goldberg, and Werneck [13] introduced the notion of Hierarchical Hub Labelling and improved the previous algorithm by considering a different way to order vertices.

We say that the labelling is hierarchical if $v \prec w$ when $w \in L_f(v) \cup L_r(v)$ is a partial order. It is shown by Abraham et al. [13] that a hierarchical label can always be transformed to a canonical label. In canonical labelling for every shortest s-t path only the highest ranked vertex v along the shortest s-t path is included in the forward label of s and the backward label of t. This implies that $r(v) \geq r(s)$ and $r(v) \geq r(t)$ and that ordering implied by canonical labelling is consistent with some total order r. Hierarchical labelling can be refined to a canonical labelling by pruning each label as follows: for every vertex $w \in L_f(v)$ we keep it only if it is the maximum ranked vertex along the v-w shortest path. To check this we can run a HL query from v to w. If there is a maximum ranked vertex $u \neq w$ it should be present in both $L_f(v)$ and $L_r(w)$ per label construction. Label construction is discussed below.

The authors propose a new Top-Down vertex ordering algorithm. For simple shortest path covering version each vertex v maintains the set of uncovered shortest paths that contain v. In each iteration a vertex in most uncovered paths is picked, and the sets of uncovered paths are updated. This is repeated until all paths are covered. Authors found that a weighted covering version incurs only small computational overhead and improves label quality. In the weighted covering version all vertex priorities (number of shortest paths that would get covered) are weighted by the size of sets consisting of the first and last vertices on the paths that would be covered by picking v. This can be considered label greedy as we maximise the ratio between number of shortest paths covered and number of labels that are updated. While the unweighted version would be path greedy (aims to cover the most uncovered paths).

To efficiently track how many shortest paths would be covered if v was picked authors suggest to store a shortest path tree for each vertex. This tree represents all uncovered shortest paths starting at that vertex. When vertex v is picked, in all the trees sub-tree rooted at v is removed (parent pointers are replaced with *NULL*). While removing each sub-tree we also update the counter of uncovered shortest paths for each of the children in the sub-tree. As we will see later, this is similar to the procedure used in the algorithm we implemented [1]. For computational reasons authors propose to start with the Contraction Hierarchies ordering and then re-order top x vertices or to apply range optimisation multiple times. Range optimisation only reorders vertices v that are in a certain range $a < r(v) < b$ in the order r at the time. First vertices up to a in the order are shortcuted, then Dijkstra's algorithm is run to compute all shortest paths that are not covered by vertices in order higher than b. The whole order r is split in

Figure 1.1: Hierarchical labelling algorithm framework [1].

multiple overlapping ranges. Algorithm is performed on each range in sequence. Multiple applications of this allow to further refine the ordering.

To generate labels authors introduce a recursive procedure. In each iteration of this procedure the lowest ranked unprocessed vertex v in the order is shortcuted. This continues till only one vertex s is left in the graph. It receives a trivial label of $L_f(s) = L_r(s) = (s, 0)$. Then going back through the order each of the vertices receive a label that is a concatenation of the labels of the vertices it was neighbouring when it was processed (shortcuted). Distance to the corresponding neighbour is added to the hub distances coming from that neighbour as well as a new hub representing that neighbour is created. If a hub is present in the labels of multiple neighbours lowest total distance to that hub is recorded in the current vertex's label. This whole process bears some similarity to the Pruned Labelling algorithm [14] we will discuss in the next chapter.

This algorithm proved to produce around 8% smaller labels than the previous algorithm and ran considerably faster.

1.4 General framework

As can be seen from Hierarchical Hub Labelling, and as highlighted by Delling, Goldberg, Pajor and Werneck [1] hierarchical labelling algorithms follow a certain framework which can be seen in Figure 1.1. First an ordering is computed for all the vertices. Then it is used to compute a labelling for exact distance queries. Optionally the labelling can then be further compressed. In this work we do not investigate the various compression techniques, but an interested reader can refer to [15, 1] for compression techniques that offer order of magnitude lower memory requirements in exchange for slower queries. Also, as proposed by Abraham et al. [11] there are various memory compression techniques that do not impact performance. Such as re-labelling vertices so that vertices that appear in the most labels receive the smallest IDs and then using fewer bits to represent their integer IDs (i.e. 8-bit INT for the first 256 IDs).

In the next chapter we will discuss the algorithm proposed by Delling et at. [1] that we implemented for our experiments in more detail.

Chapter 2

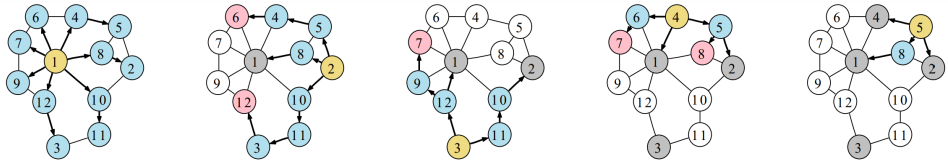
Algorithm

Here we discuss the algorithm we chose to implement for our experiments. It was proposed by Delling et at. [1] and follows the mentioned framework. To produce labelling from a given order Pruned Labelling (PL) algorithm proposed by Akiba, Iwata and Yoshida [14] is used as a sub-routine. Vertex ordering is computed online, but is ideologically similar to the non-weighted (path greedy) ordering scheme of Hierarchical Hub Labelling. This algorithm currently does not have an open source implementation. It was chosen because it offers superb performance for a single core implementation and appears easier to modify than previously mentioned algorithms. It is a cleaner and more distilled version of what we have seen previously.

2.1 Pruned Labelling

Pruned Labelling (PL) was proposed by Akiba et al. [14]. It was originally proposed for unweighted graphs, but it easily extends to weighted ones.

As originally proposed PL runs pruned BFS from vertices in the order $v_1, v_2, ..., v_n$. Algorithm starts with empty labels L_f and L_r. At each step i BFS search is run from a vertex v_i. When BFS_i visits a node u we use current partial labels $L_f(v_i)$ and $L_r(u)$ to compute the best currently known shortest path in the labels. If the distance of the shortest path from the labels is lower or equal to current $dist(v_i, u)$ we prune the BFS and do not check any outgoing edges of u. If the current distance is smaller than the one recovered from the labels the root of the BFS tree (current hub v_i) is added to the reverse label $L_r(u)$ of the visited node with appropriate distance and the BFS proceeds as usual. The reverse BFS is also run from v_i to update the forward labels of the nodes $L_f(u)$ that operates in the same way. The progression of the algorithm on a bidirectional graph can be seen in Figure 2.1 as provided by Akiba et al. [14]. In our implementation we use Dijkstra's algorithm instead of the BFS, but prunning works in the same way. Akiba et al. [14] show that PL is correct and produces minimal labelling. Meaning if any hub would be removed from any label we would lose the cover



(a) First BFS from vertex 1. We visited all the vertices.

(b) Second BFS from vertex 2. We did not add labels to five vertices.

(c) Third BFS from vertex 3. We only visited the lower half of the vertices.

(d) Fourth BFS from vertex 4. This time we only visited the higher half.

(e) Fifth BFS from vertex 5. The search space was even smaller.

Figure 2.1: Examples of pruned BFSs. Yellow vertices denote current roots, blue vertices were visited and labelled, red vertices were visited but pruned, and grey vertices were already used as roots [14].

property. Delling et al. [1] show that PL produces hierarchical and canonical labelling. Which should be fairly obvious from the pruning algorithm.

To improve performance Akiba et al. [14] suggest to not re-initialise each BFS tree as that takes $O(n)$ and BFS trees get much smaller as the algorithm progresses. In our implementation distance arrays used while performing Dijkstra's algorithm are also not re-created. Instead we track which vertices where visited and re-set the corresponding cells of the array after Dijkstra's algorithm terminates. In our experience this offers a substantial performance boost on large graphs.

2.2 Vertex ordering

Delling et al. [1] propose a new sampling based ordering scheme called SamPG. It is based on the unweighted top-down ordering we saw in Hierarchical Hub Labelling [13]. It works as follows: initially we sample $k << n$ vertices uniformly at random and build shortest path trees from them. These trees are then used to estimate how many shortest paths a particular vertex would cover. Of course after selecting first few vertices we would get very unreliable estimates, as few uncovered shortest paths would remain in the sampled trees. This is solved by continuously sampling more shortest path trees from random roots after each Pruned Labelling iteration. The sampled trees are pruned in the same way as during Pruned Labelling iterations. Meaning shortest paths already covered by current hubs are pruned. Time spent on the tree sampling and the labelling is balanced by tracking how many vertices were visited in total during labelling operations c_l and how many vertices were visited during tree sampling c_t (original k trees are free). Trees are only sampled until $c_t > c_l$ or number of vertices in the existing trees exceeds $10kn$.

While total number of descendants in sampled trees is a straight forward estimate of vertex importance it is not very robust. The importance of vertices that are near the root of a randomly sampled tree is overestimated in such case. To

remedy this, the authors [1] proposed to maintain c buckets of trees and counters $\sigma_1(v), \sigma_2(v), ..., \sigma_c(v)$. Where a counter $\sigma_i(v)$ is a count of descendants of v over all trees t_i such that $i = (j \mod c)$. Then robustness of estimation can be increased by discarding the maximal counter for v and summing all others to obtain importance estimate for v. Ties are broken by considering total sums without discarding the maximum counter. Authors found that $c = 16$ and discarding two biggest counters before summing them up works best. In our implementation we used the same procedure and values.

The trees are initially stored in memory as n sized arrays. Delling et al. [1] propose to switch to a hash table representation when a tree contains less than $n/8$ vertices.

At each iteration of the algorithm after picking a vertex v that is going to become a new hub we have to update all the trees and counters removing all of the children of v. In the beginning to determine v we scan over all vertices computing their importance and choose the most important one. Then we iterate over all trees t_i and and remove v and its descendants from t_i and update all counters accordingly. However, as soon as we pick v that covers less than $n/8$ shortest paths we begin using a reverse index that for each vertex stores a pointer to the trees that contain it. We also build a max-heap of vertices using their importance, which allows us to more efficiently retrieve next hub v in subsequent interactions. Next, we switch tree representation from n sized array to a hash table for all of the trees. Original authors [1] suggest to switch the representation of each tree dynamically.

2.3 Distance and path queries

2.3.1 Distance query

To efficiently return the distance $dist(v, u)$ between two vertices v and u we have to make sure that the labels are sorted by vertex ID. Then we can scan both labels $L_f(v)$ and $L_r(u)$ only once to find a matching hub with minimum distance. So query time will be linear in the label size $O(|L_f(v_i)| + |L_r(u)|)$ as discussed by Abraham et al. [6].

2.3.2 Path query

We elected to perform a recursive distance query to retrieve the full path. Meaning when we query distance and find the shared hub j in the labels $L_f(v)$ and $L_r(u)$ that provides minimum total distance we then perform a distance query from v to j and from j to u. Then we concatenate all the hubs that were found during the recursive procedure into a path. However, because of Pruned Labelling

construction it might happen so that u is directly present in the label $L_f(v)$ even if they are not neighbours given that u is a very important node and vice versa. In this case the recursive algorithm will not find a next hop to recurse on. To solve this we elected to extend the Pruned Labelling scheme so that it includes the current hub into a label of a vertex if the current distance is equal to the best recorded distance in the label. In such a case the neighbours of this vertex are not explored. This ensures that we will always have the next hop recorded in the labels. In our experiments this increased the label size by around 10%. But this way we ensure that path queries are linear in the path length times the label size up to a constant factor. With the modified labels we can recurse till we find no shared hub u. Then this means that the vertices are neighbours. This label usage to check if they are neighbours allows us to return paths using just labels without the original graph.

Of course there are other alternatives to recover the full path without modifying the labels. For example one could use a guided Dijkstra's algorithm that at each step would proceed to the neighbour that has the lowest distance to the target. However, this approach would be slower as it would require us to compute distances to the target for the neighbours of the whole path.

2.3.3 Optimised distance query for Pruned Labelling and pruned tree sampling

Akiba et al. [14] also proposed a faster way to compute distances during Pruned Labelling iterations that we also use for pruning sampled trees. Normal distance query algorithm is linear in sum of label lengths $O(|L_f(v_i)| + |L_r(u)|)$. We can instead pre-compute an array T of length n where $T[j] = dist_{L_f(v_i)}(j)$ for all $j \in L_f(v_i)$ and $T[j] = \inf$ otherwise. Then distance query between v_i and u can be answered if for all l in $L_f(u)$ we compute $T[l] + dist_{L_r(u)}(l)$ and return the minimum. This algorithm runs in $O(|L_r(u)|)$ for repeated queries from v_i.

Chapter 3

Data

3.1 Graphs

We perform our tests on graphs[1] made for 9th DIMACS Implementation Challenge[16]. Namely the US TIGER[2] graphs that are undirected and sometimes have gaps in their road network and directed Western Europe road graph that was provided by PTV Planung Transport Verkehr AG[3] to the challenge participants. This Western Europe graph does not include Ireland and Czech Republic. Details about the graphs we used can be found in Table 3.1. All graphs come with node coordinates and have a distance and a time based versions abbreviated as "-t" and "-d" respectively.

Abbreviation	Description	Type	Vertices	Edges
NY	New York City	Undirected	264,346	733,846
BAY	San Francisco Bay Area	Undirected	321,270	800,172
COL	Colorado	Undirected	435,666	1,057,066
FLA	Florida	Undirected	1,070,376	2,712,798
NW	Northwest USA	Undirected	1,207,945	2,840,208
NE	Northeast USA	Undirected	1,524,453	3,897,636
CAL	California and Nevada	Undirected	1,890,815	4,657,742
LKS	Great Lakes region	Undirected	2,758,119	6,885,658
E	Eastern USA	Undirected	3,598,623	8,778,114
W	Western USA	Undirected	6,262,104	15,248,146
CTR	Central USA	Undirected	14,081,816	34,292,496
EU	Western Europe	Directed	18,010,173	42,560,279

Table 3.1: Graphs used in our experiments

In addition to these graphs we have extracted a graph for Porto, Portugal out of the Western Europe graph that has 19,999 vertices and 46,959 edges.

[1] http://users.diag.uniroma1.it/challenge9/download.shtml
[2] https://www.census.gov/programs-surveys/geography.html
[3] https://i11www.iti.kit.edu/resources/roadgraphs.php

Figure 3.1: Porto graph extracted from PTV Western Europe graph.

This graph is used to compare taxi traces we have to the hubs produced by Hub Labelling. Taxi traces are discussed in the third subsection of this chapter. We kept all nodes between the coordinates of (41.5 -8.250) and (41.0 -8.750) as this is where majority of the taxi traces we had were located. We also include neighbours of these nodes. The extracted graph can be seen in Figure 3.1.

3.2 Taxi trips

In all the Hub Labelling algorithms it is assumed that all shortest paths are made equal. In reality it is possible some shortest paths are used much more than others (their destinations are more popular). To check this assumption we use records of taxi trips that have start and end point coordinates for each trip. We then compute shortest paths on the graph for those trips. More details will be provided in the next chapter.

We use regulatory logs submitted to and published by NYC Taxi and Limousine Commission [4]. All Yellow and Green cabs in New York City have to submit their trip logs. Up to 2016 these logs contained trip start and end coordinates. For our research we downloaded logs for January 2015 (14 million trips). We

[4] https://www1.nyc.gov/site/tlc/about/tlc-trip-record-data.page

then filtered out trips that took place on the third week of January (2015-01-19 to 2015-01-25) that was quite uneventful. This resulted in 4,6 million trips. Out of those we randomly extracted 100,000 trips and mapped each start and end point to the closest vertex on our graph using haversine distance.

3.3 Taxi traces

We use taxi trace dataset[5] provided by Moreira et al. [17]. This dataset includes taxi traces during all occupied taxi trips taken from 2013-07-01 to 2014-06-30 in any of the 442 taxis operating in Porto, Portugal. All trips that had missing points in their GPS traces were marked by the creators of the dataset. We discarded such trips, as well as any trips that only had one GPS point associated with them. This results in 1.7 million clean trips. Out of those we randomly sample 340 thousand (20 %) and map them to the graph we extracted for Porto from the PTV Western Europe graph as described in the previous section.

3.3.1 Fast map matching

To map the taxi traces to the graph we use the algorithm by Yang and Gidofalvi [18]. It has an open source implementation[6]. It is claimed by the authors that the algorithm can match up to 45,000 points/s. The algorithm is based on the Hidden Markov Model approach that is now a classic approach to map matching since Newson and Krumm [19] proposed it. The HMM algorithm treats GPS point emission probabilities as Gaussian distributed around the true location of the car and uses the variance as a free parameter. Transition probabilities are estimated as a ratio between shortest path to the candidate point on the candidate edge and euclidean distance to that point (from the current locations). Candidate points are selected by mapping the GPS point to the nearest point on the candidate edge. Understandably only the candidates that are not too far away from the GPS points receive a non zero probability. After the HMM is modelled the Viterbi algorithm is used to compute the most likely trajectory. The most expensive part of this tends to be the computation of the shortest paths between the current point and the candidate point [18]. The Fast Map Matching (FMM) algorithm [18] tries to alleviate this problem by precomputing all of the shortest paths on the graph that have a distance of at most Δ. We set $\Delta = 5$km. This makes repeated path mapping on the same graph much faster. If the shortest path to a candidate point is not found in the precomputed hash table (is longer than Δ) Dijkstra's algorithm takes over. For other parameters we set GPS error (σ for emission probability) to 50 m, candidates to consider to

[5] https://archive.ics.uci.edu/ml/datasets/Taxi+Service+Trajectory+-+Prediction+Challenge,+ECML+PKDD+2015

[6] https://github.com/cyang-kth/fmm

Figure 3.2: GPS trace and its matched path. Yellow dots mark the raw GPS trace, red dots mark vertices along the matched path.

8, candidate search radius to 300 m and no direction change penalty. You can see GPS trace and its matched path in Figure 3.2.

Chapter 4

Results

4.1 Importance of a good ordering

First we make sure good vertex ordering is as important for label size and runtime as assumed. For this we first ran original Pruned Labelling algorithm as proposed by Akiba et al. [14]. It orders vertices by their degree. It is know that this ordering does not perform well on some graphs. In particular road networks [14, 1]. As can be seen in Table 4.1, SamPG order indeed performs much better on the road graphs. Here for SamPG we initially sampled 1000 shortest path trees.

Graph	Vertex order	Preprocessing time	Average label size
NY-d	Degree	497 s	704
NY-d	SamPG	82 s	71
NY-t	Degree	278 s	556
NY-t	SamPG	58 s	46

Table 4.1: Performance based on ordering

As mentioned in the previous section for SamPG ordering we choose how many trees are initially sampled. Delling et al. [1] do not specify how this parameter was chosen in their study. We perform a small study to determine what influence this parameter has to the label size (and the preprocessing time). As can be seen in Table 4.2, this parameter does have an impact on label size, but not a large one in most cases. Unless we sample way too few trees. This is most visible when we only sample 10 and 50 trees for the time graph. Here we can also see that the algorithm's preprocessing time increases superlinearly with the label size, when the ordering is bad. As well as the fact that distance graphs appear to be more resilient to sampling too few trees initially. In this table we didn't include the time to sample the initial trees in the preprocessing time. Sampling 1000 trees takes 38 s in the time graph and 48 s in the distance graph.

Graph	Initially sampled trees	Preprocessing time	Average label size
NY-d	10	143 s	120
NY-d	50	73 s	78
NY-d	100	54 s	76
NY-d	500	66 s	66
NY-d	1,000	58 s	71
NY-d	3,000	148 s	70
NY-d	5,000	216 s	70
NY-d	10,000	376 s	69
NY-d	20,000	670 s	68
NY-t	10	1318 s	458
NY-t	50	234 s	139
NY-t	100	60 s	70
NY-t	500	47 s	47
NY-t	1,000	58 s	46
NY-t	3,000	139 s	45
NY-t	5,000	207 s	45
NY-t	10,000	363 s	44
NY-t	20,000	662 s	44

Table 4.2: Influence of initially sampled trees.

4.2 Scaling to larger graphs

We investigate how the algorithm's runtime and label size scale with the graph size. We always sample 1000 shortest path trees initially. Results can be seen in Table 4.3. We can see that the label size tends to increase with the graph size. However we see that by increasing the graph size 10 times, label size only increases by 10 to 30 %. This is what we expect to see as far away destinations should be accessible trough only a few hubs and most of the hubs in the label come from nearby destinations. Runtime seems to depend on both, the graph size and the label size. We also observe that distance query times are as expected more or less linearly dependent on the label size. We sampled 10,000 random trips (source and target vertices) to estimate the query time. Of course this makes the estimate somewhat noisy.

Compared to Delling et al. [1] on EU-t graph our algorithm takes 1 hour longer (3h 21 min compared to 2h 19 min) to complete preprocessing and produces labels that are 18 % larger (97 compared to 82). This label size increase is partially due to our modified Pruned Labelling algorithm that allows straight forward recursive path query. The difference can also partially come from Delling et al. potentially using more initially sampled shortest path trees. As we have seen in the previous section sampling more trees initially reduces the label size.

Graph	Vertices	Edges	Preprocessing	Label size	Distance query
NY-d	264,346	733,846	1 min 22 s	71	0.9 μs
BAY-d	321,270	800,172	1 min 23 s	53	0.8 μs
COL-d	435,666	1,057,066	2 min 3 s	62	0.9 μs
FLA-d	1,070,376	2,712,798	5 min 57 s	68	1.1μs
NW-d	1,207,945	2,840,208	6 min 50 s	70	1.2 μs
NE-d	1,524,453	3,897,636	12 min 4 s	100	1.4 μs
CAL-d	1,890,815	4,657,742	13 min 1 s	81	1.2 μs
LKS-d	2,758,119	6,885,658	25 min 49 s	106	1.5 μs
E-d	3,598,623	8,778,114	35 min 29 s	112	1.6 μs
W-d	6,262,104	15,248,146	57 min 46 s	103	1.6 μs
CTR-d	14,081,816	34,292,496	4 h 7 min	159	2.2 μs
EU-d	18,010,173	42,560,279	7 h 44 min	376	3.4 μs
NY-t	264,346	733,846	58 s	46	0.8 μs
BAY-t	321,270	800,172	1 min 13 s	39	0.7 μs
COL-t	435,666	1,057,066	1 min 34 s	39	0.8 μs
FLA-t	1,070,376	2,712,798	4 min 49 s	48	0.9 μs
NW-t	1,207,945	2,840,208	5 min 26 s	45	0.9 μs
NE-t	1,524,453	3,897,636	8 min 10 s	55	1.0 μs
CAL-t	1,890,815	4,657,742	9 min 53 s	51	1.0 μs
LKS-t	2,758,119	6,885,658	10 min 1 s	55	1.0 μs
E-t	3,598,623	8,778,114	23 min 7 s	59	1.1 μs
W-t	6,262,104	15,248,146	40 min 10 s	60	1.2 μs
CTR-t	14,081,816	34,292,496	2h 27 min	75	1.5 μs
EU-t	18,010,173	42,560,279	3 h 21 min	97	1.6 μs

Table 4.3: Scaling of the algorithm. Using 1000 initially sampled trees.

4.3 The impact of graph metric

As can be seen from Table 4.2 and Table 4.3 time graphs always yield smaller labels and require shorter pre-processing time (if sufficiently many trees are initially sampled). Intuitively it makes sense as highways should be even more important in the time graph. Thus time graph should have a lower highway dimension. Here we provide a visual inspection of hubs on the time and distance graphs of New York (Figure 4.1) and Western Europe (Figure 4.2 and 4.3). In both cases we see that time graph hubs are placed in more intuitive locations.

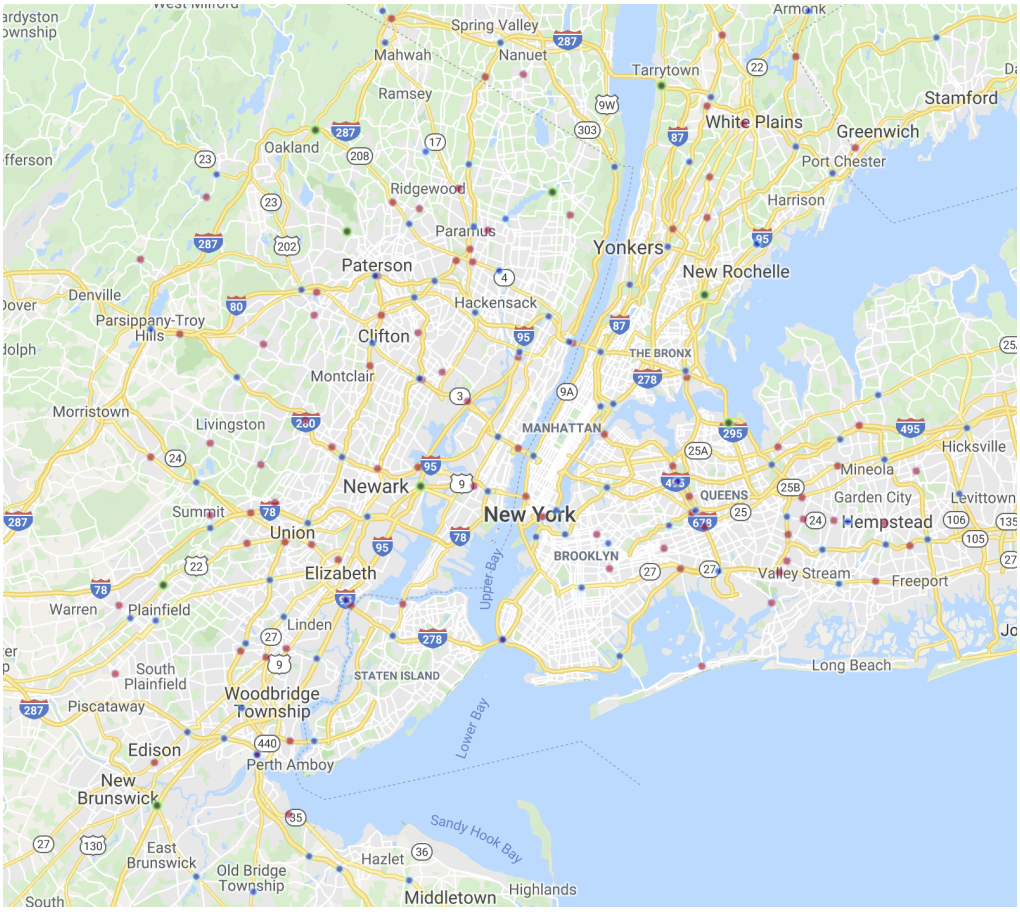


Figure 4.1: 100 most important hubs computed on the New York graph. Blue dots mark the time graph hubs, red dots the distance graph hubs and green dots hubs that coincide between the two graphs.

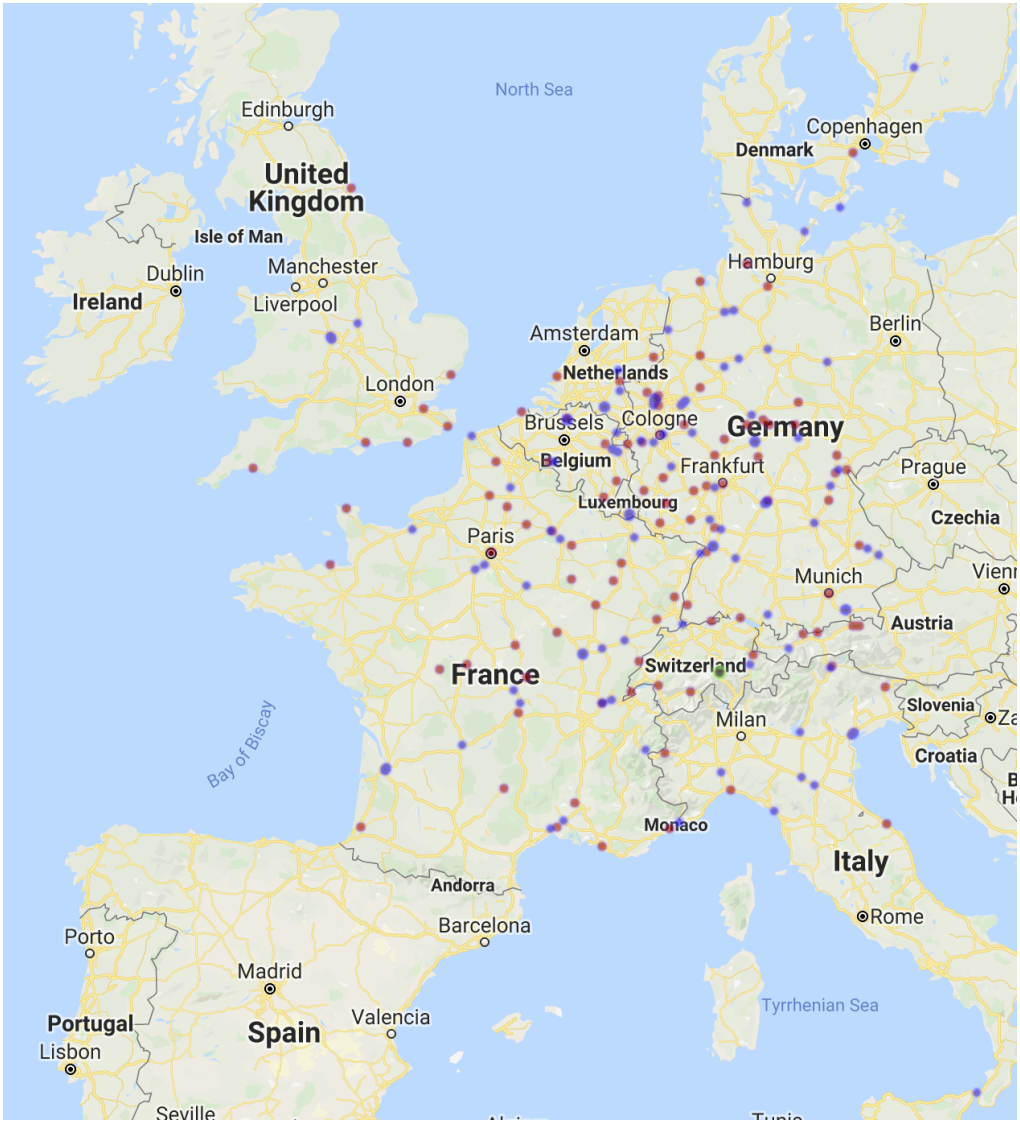


Figure 4.2: 100 most important hubs computed on the Western Europe graph. Blue dots mark the time graph hubs, red dots the distance graph hubs and green dots hubs that coincide between the two graphs (only Gotthard Pass).

Figure 4.3: 10 most important hubs computed on the Western Europe graph. Blue dots mark the time graph hubs, red dots the distance graph hubs and green dots hubs that coincide between the two graphs (only Gotthard Pass).

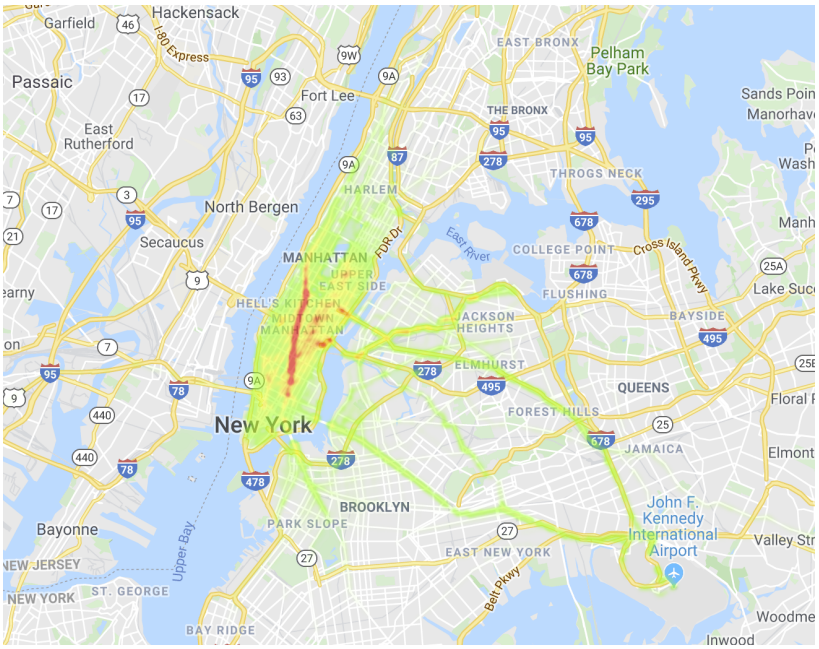


Figure 4.4: Raw taxi trip heatmaps on the New York distance graph.

4.4 Unequal importance of shortest paths

As we have mentioned in the hub labelling algorithm, all shortest paths are assumed to be equally important. In reality it might happen that some destinations (vertices) are much more popular than others. Intuitively it makes sense, a dirt road in the outskirts of a city will see much less traffic than a road in the city centre. To check which hubs would be more important if we weight them by popularity we use New York taxi trip records that provide start and end location of every trip taking place in New York between 2015-01-19 and 2015-01-25. We randomly sampled 100,000 of these trips and used Hub Labelling to compute shortest paths. The raw heatmaps produced by the sampled trips can be seen in Figure 4.4 and Figure 4.5. As can be expected taxi trips heavily prefer the city centre and the airport. Which might not be entirely representative of the true traffic. We should also keep this in mind in the next section. Using this more realistic path weighting we estimate hub importance by greedily picking the hub that covers most yet uncovered taxi trips. This is similar to the path greedy SamPG procedure we use. Comparison with time and distance graph hubs can be seen in Figure 4.6. We can notice that for the top 200 time graph hubs two overlap with the taxi trip hubs. While for the distance graph none do. We also see that the top 10 most important taxi trip hubs between the distance graph and the time graph are similar. With time graph taxi trip hubs showing more preference to the highway nodes.

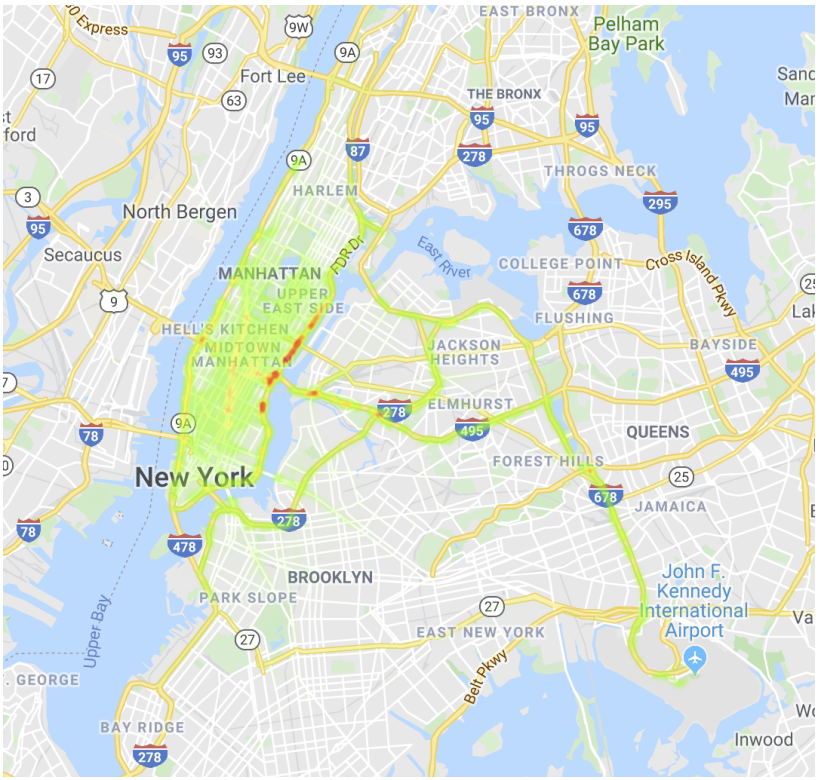
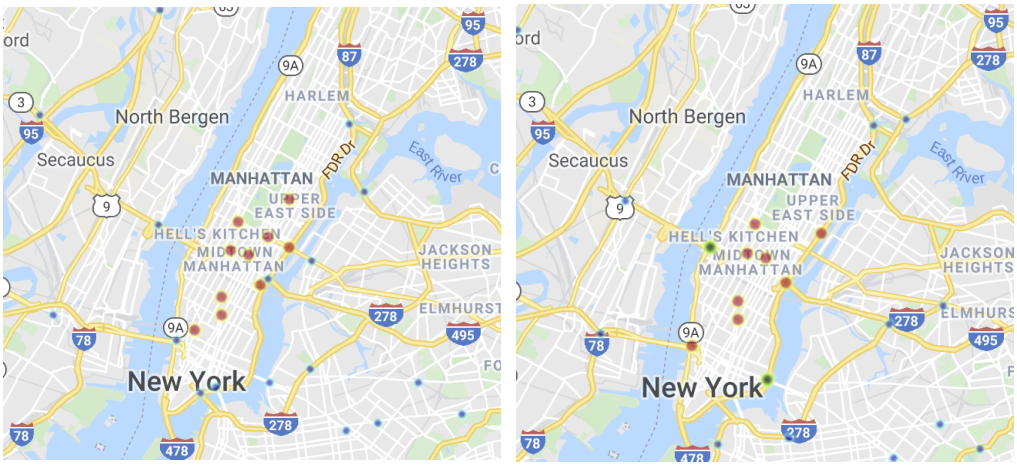


Figure 4.5: Raw taxi trip heatmaps on the New York time graph.



(a) Distance graph (b) Time graph

Figure 4.6: Ten most important hubs in New York time (a) and distance (b) graphs for taxi trips. Blue dots represent top 200 hubs as computed by HL, red dots are 10 most important hubs computed from taxi trips and green dots are overlapping hubs.

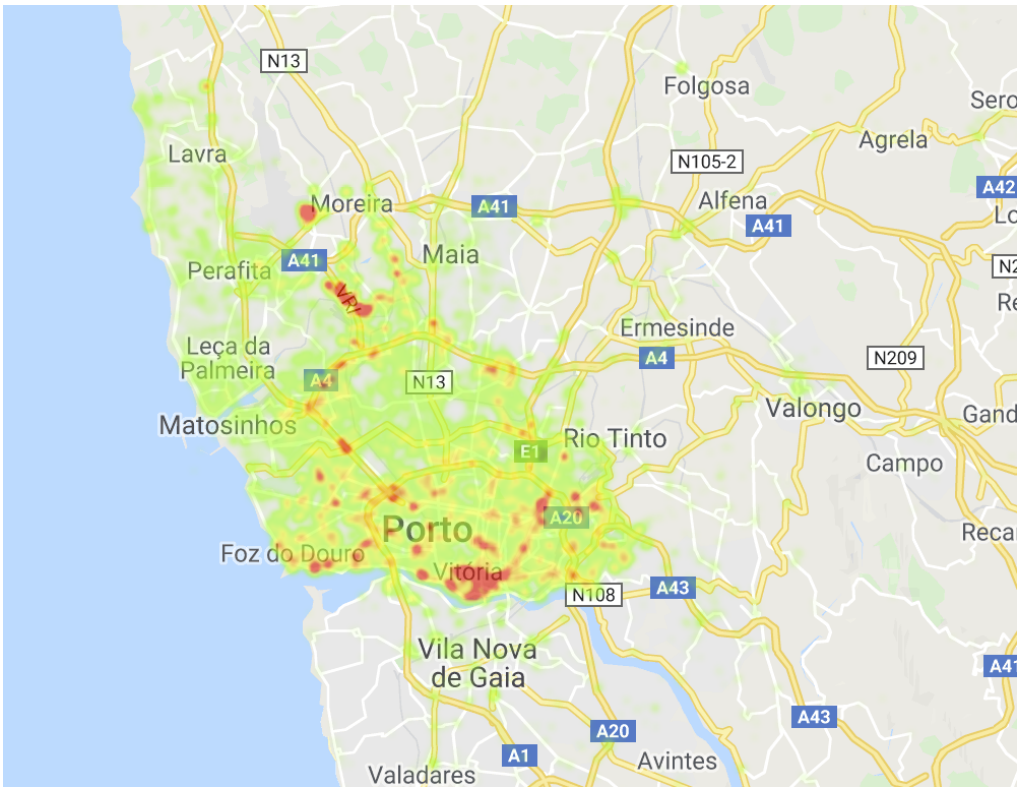


Figure 4.7: Mapped taxi trip trace heatmaps for taxi trips in Porto from 2013-07-01 to 2014-06-30.

4.5 Hub importance compared to real world traffic

We use taxi trip GPS traces from all the taxi cabs active in city of Porto to evaluate which hubs would be most important according to them. As we saw before, taxi trips are unbalanced in the sense that they prefer transportation hubs (i.e. airports and train stations) and city centre to residential areas. But taxi drivers are experienced and can be expected to chose an optimal route. Heatmap of the mapped taxi traces can be seen in Figure 4.7. We computed the most important hubs according to taxi traces in the same way as in the previous section. By greedily picking hubs that cover the most of so far uncovered taxi traces. Comparison with distance and time graph hubs can be seen in Figure 4.8 and Figure 4.9 respectively. In both graphs none of the top 200 HL hubs overlap with top 10 hubs we get from taxi traces. Although time graph hubs seem to be a bit closer to the taxi trace hubs.

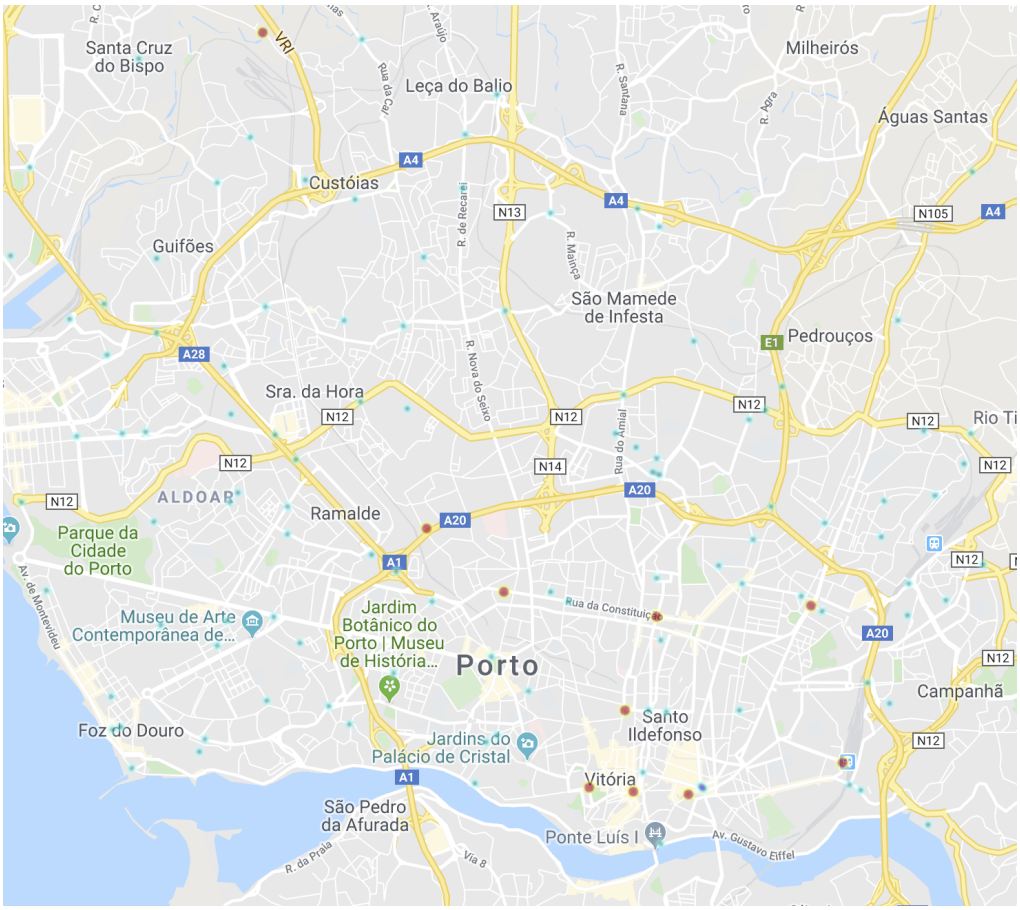


Figure 4.8: Most important hubs in Porto distance graph compared to most important taxi trip trace hubs. Red dots mark 10 most important hubs based on taxi traces and blue dots mark 100 most important distance graph hubs selected by HL.

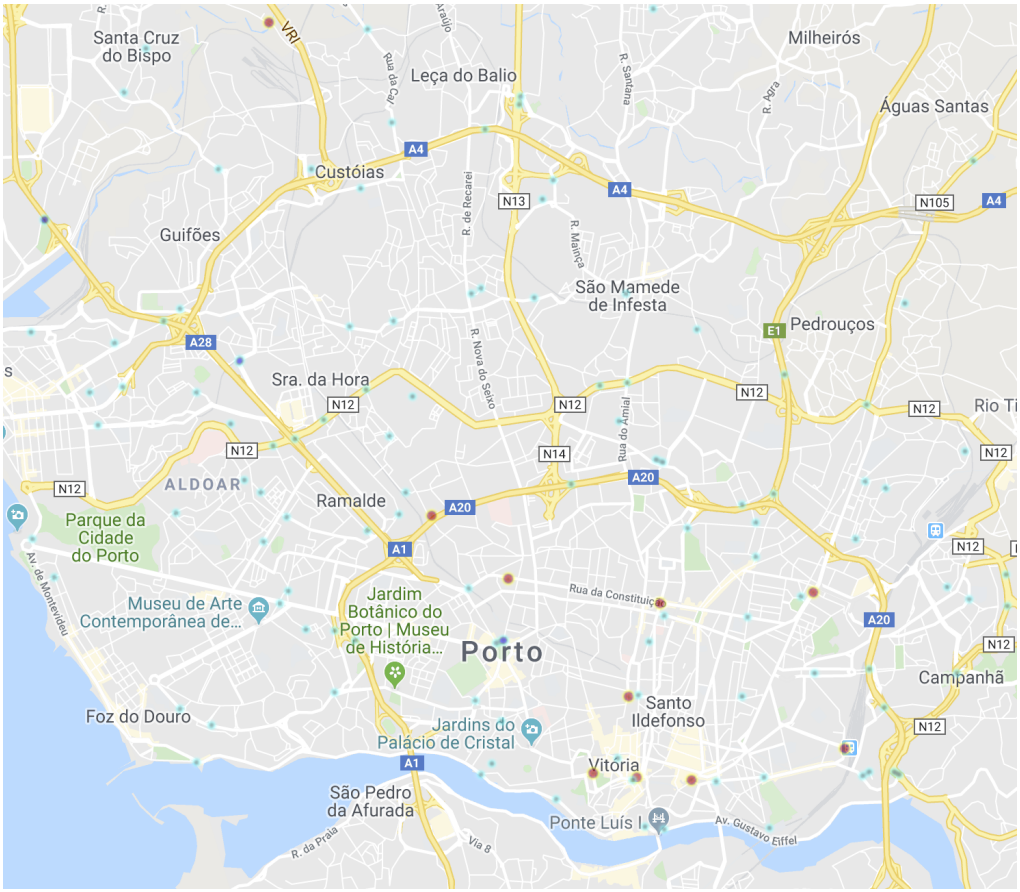


Figure 4.9: Most important hubs in Porto time graph compared to most important taxi trip trace hubs. Red dots mark 10 most important hubs based on taxi traces and blue dots mark 100 most important time graph hubs selected by HL.

4.6 Performance of recursive shortest path queries

Our modified labels always include a hop between any two vertices, unless they are neighbours. We expect our recursive path query algorithm to perform a number of HL queries linear in the path length. Up to a factor of 2 as when the nodes are next to each other one extra HL query is performed to make sure there is no intermediary node and they are in fact neighbours. In Table 4.4 we perform a small study on different length randomly sampled shortest paths. We sample paths randomly until we find 1000 paths of each length. We see that in fact time taken per hop in the path is at most twice the distance query time. The path query times are orders of magnitude faster than Dijkstra. Dijkstra can take more than a second to answer a long path query on the graph of Western Europe.

Graph	Distance query time	Path length	Path query time	Time per hop
NY-d	0.9 μs	≤ 100 hops	85 μs	0.97 μs
NY-d	0.9 μs	$\in (100, 1000)$ hops	507 μs	0.95 μs
NY-d	0.9 μs	≥ 1000 hops	934 μs	0.88 μs
NY-t	0.8 μs	≤ 100 hops	62 μs	0.74 μs
NY-t	0.8 μs	$\in (100, 1000)$ hops	275 μs	0.76 μs
NY-t	0.8 μs	≥ 1000 hops	no paths	no paths
EU-d	3.4 μs	≤ 100 hops	256 μs	3.12 μs
EU-d	3.4 μs	$\in (100, 1000)$ hops	2417 μs	3.21 μs
EU-d	3.4 μs	≥ 1000 hops	12591 μs	3.06 μs
EU-t	1.6 μs	≤ 100 hops	109 μs	1.28 μs
EU-t	1.6 μs	$\in (100, 1000)$ hops	1411 μs	1.42 μs
EU-t	1.6 μs	≥ 1000 hops	3298 μs	1.06 μs

Table 4.4: Shortest path query performance. Path query times are averaged over 1000 random samples.

4.7 Recovering alternative path from the labelling

Potentially interesting application would be to use hub labelling to recover good alternative shortest path. For example in case of an accident on the road or to balance traffic and reduce congestion. While ideally the labelling algorithm would be modified for this use case, we show that even the current implementation returns acceptable alternative shortest paths over city sized distances. This might be partially due to the fact that we introduce extra hubs into the labels to simplify shortest path recovery. First we use New York taxi trip start and end points to generate paths.

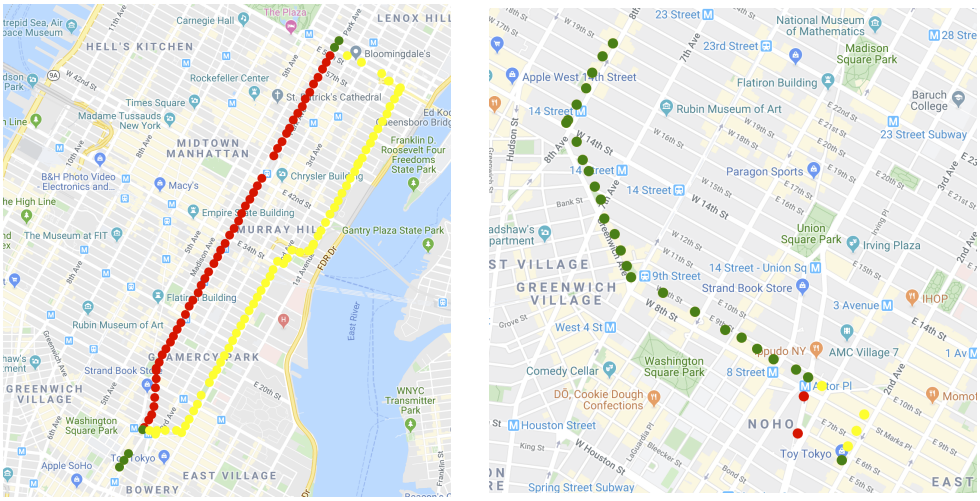
To recover an alternative s-t path from the label while scanning $L_f(s)$ and $L_r(t)$ we return the hub with 2nd best distance (2nd best hub). Then to recover the full path we recurse as usual, taking best distances in all subsequent steps. As a result of this procedure some trips end up with very similar paths while some get a good alternative path (Figure 4.10). If we look at how much the path length increased for the alternative path (fractional stretch) we see that the increase is mostly less than 20 % (Figure 4.12). If we investigate how good the alternative path is in respect to how big of a fraction of its hubs are present in the shortest path (Figure 4.13) we observe that the alternative path uniqueness varies quite uniformly. With more unique paths being slightly more likely for real world taxi trips in New York.

If we look at various length random trips in Western Europe (Figure 4.11) we see that for short trips (at most 100 hops) the behaviour is similar to the New York graph (Figure 4.14 and 4.15). Although we see that paths are less unique in this case (bigger fraction of hubs are shared with the shortest path). For medium length paths the situation is similar, but even bigger fraction of hubs overlap between the alternative and the shortest paths (Figure 4.16 and 4.17). Here we also see that there is a small number of trips for which an alternative path is not found (zeroth column in the histogram in Figure 4.17). In our alternative path recovery scheme this can happen when the shortest path goes over a very important hub in the graph which does not have any alternative due to the hierarchical construction. For long paths (Figure 4.16 and 4.17) we see that this problem becomes more acute. Also more hubs are shared between the alternative and the shortest paths. Although inspecting them visually (Figure 4.11) we can see that they do differ for a substantial distance and could be useful to avoid certain part of the map. In both cases we notice that alternative path distance increase is smaller in the medium length and long paths. This might be related to the fact that they tend to be less unique too.

All the figures provided here are for the corresponding time graphs. The distributions were identical for the distance graphs.

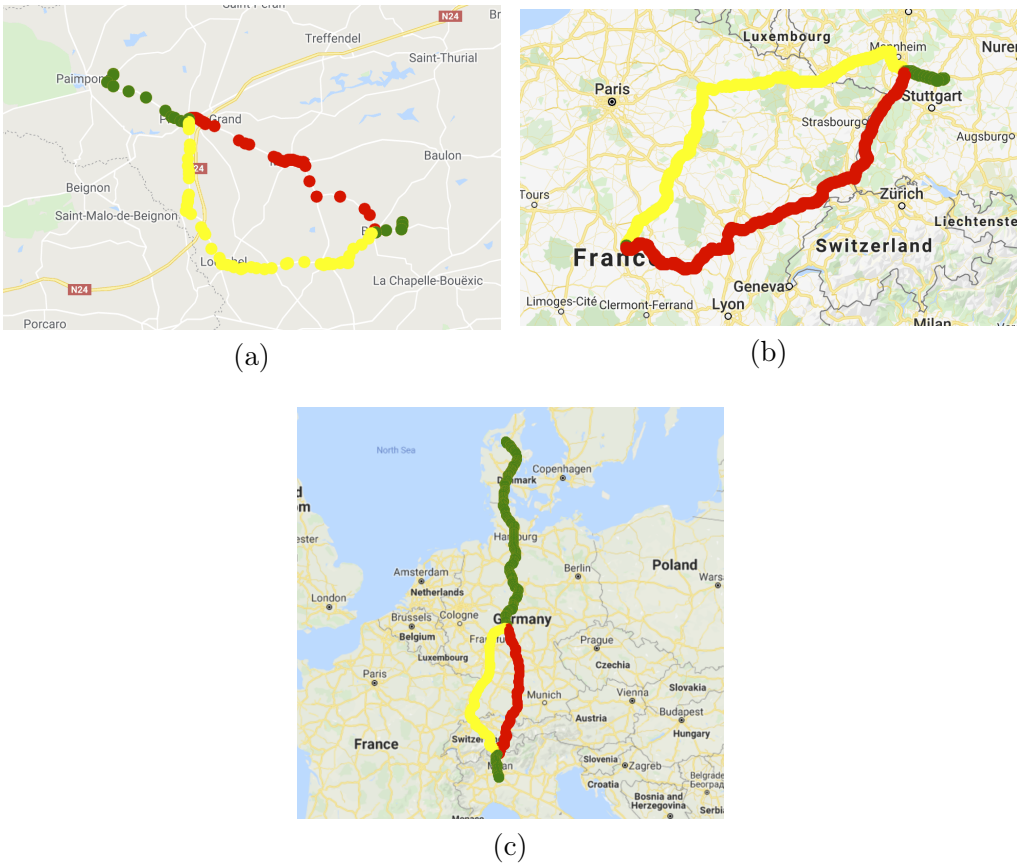
Our alternative path recovery scheme can be modified to route around any vertex in the s-t shortest path. When we detect that an undesirable vertex u will be our next hop we then instead use the alternative path recovery scheme for that portion of the path and choose second best hub from the labels instead of u. If this portion of the path is short we will be in the favourable scenario of alternative path for a sub 100 hop path that we have just seen.

It is left for future work to see if uniqueness and recovery of alternative paths can be improved by a different alternative path recovery algorithm and/or different label construction.



(a) (b)

Figure 4.10: Example of a good (a) and worse (b) alternative path for a taxi trip in NY. Red dots are shortest path vertices, yellow dots are alternative path vertices and green dots are overlapping vertices.



(a) (b)

(c)

Figure 4.11: Example of a short (a) and medium length (b) and long (c) alternative path for a random trip in Western Europe. Red dots are shortest path vertices, yellow dots are alternative path vertices and green dots are overlapping vertices.

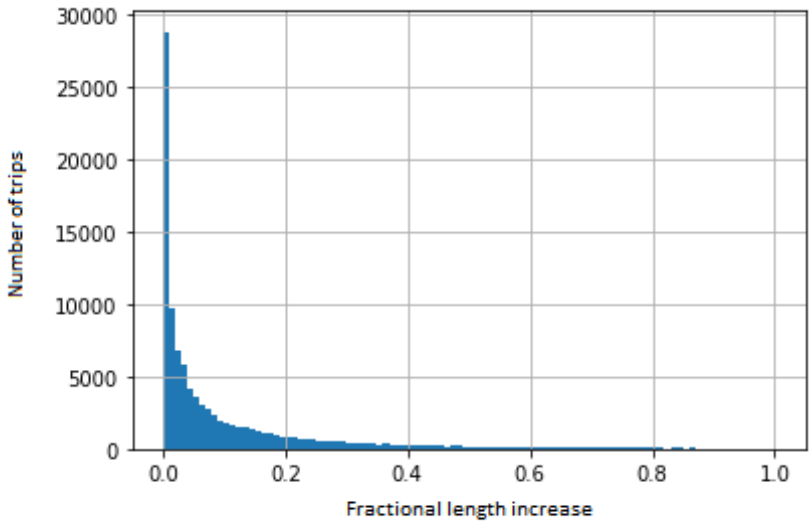


Figure 4.12: NY taxi trip histogram for fraction of length increase in the alternative path compared to the shortest path.

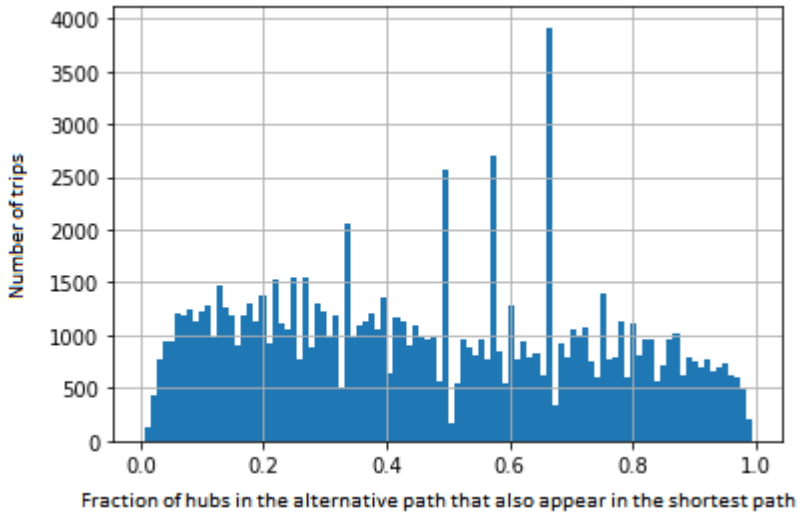


Figure 4.13: NY taxi trip histogram for fraction of hubs in the alternative path that were present in the shortest path.

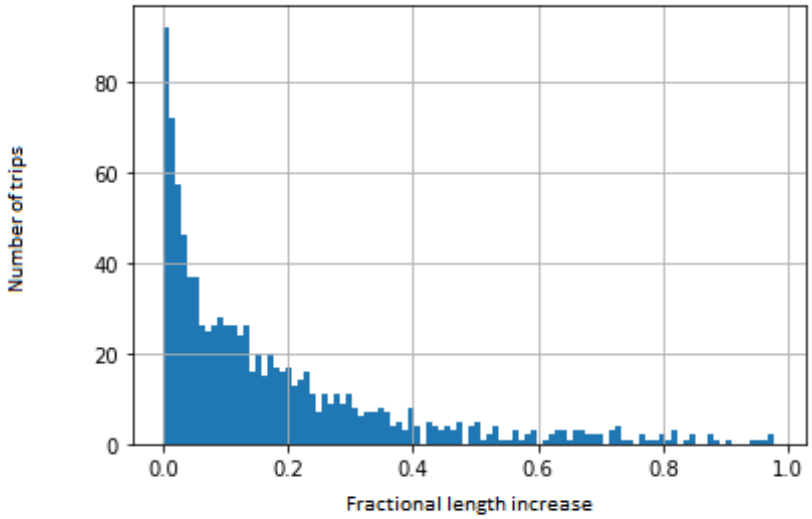


Figure 4.14: EU random short trip (at most 100 hops) histogram for fraction of length increase in the alternative path compared to the shortest path.

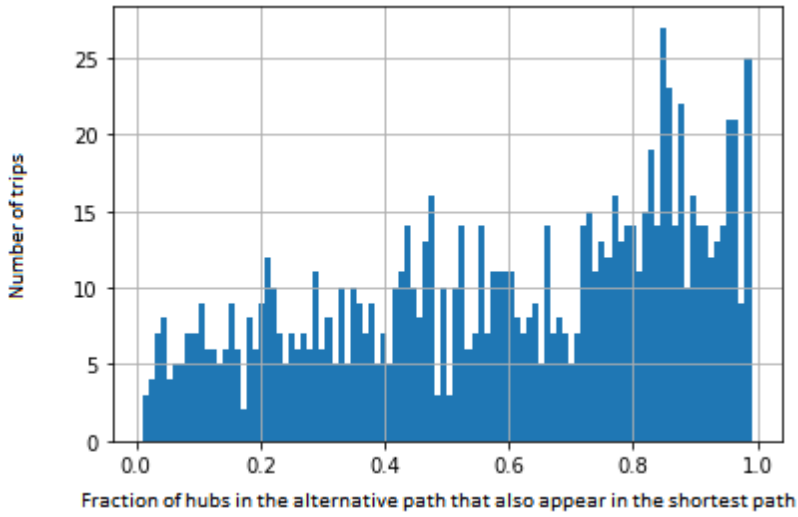


Figure 4.15: EU random short trip (at most 100 hops) histogram for fraction of hubs in the alternative path that were present in the shortest path.

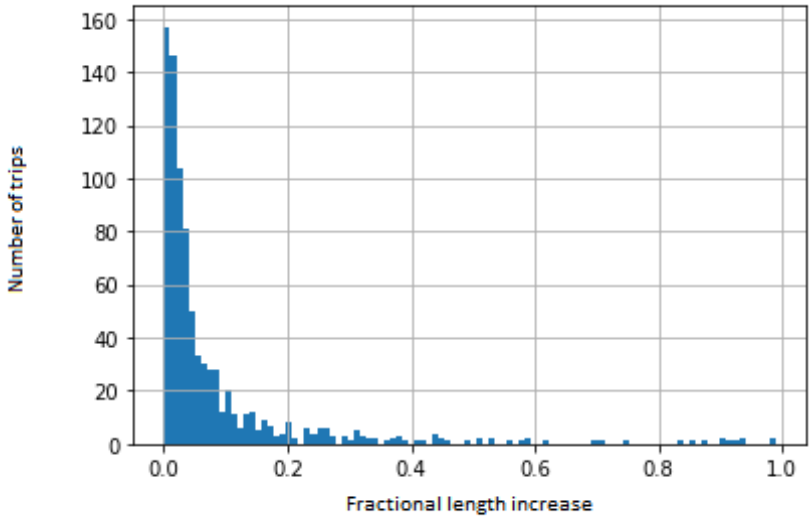


Figure 4.16: EU random medium length trip (between 100 and 1000 hops) histogram for fraction of length increase in the alternative path compared to the shortest path.

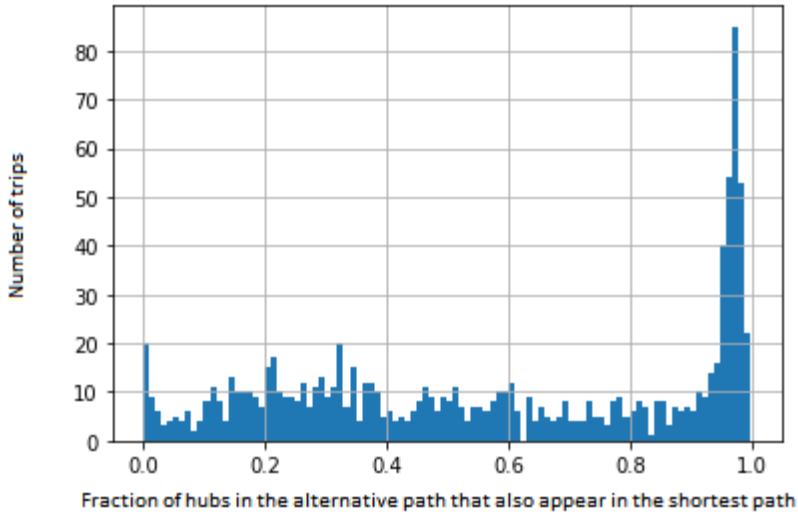


Figure 4.17: EU random medium length trip (between 100 and 1000 hops) histogram for fraction of hubs in the alternative path that were present in the shortest path.

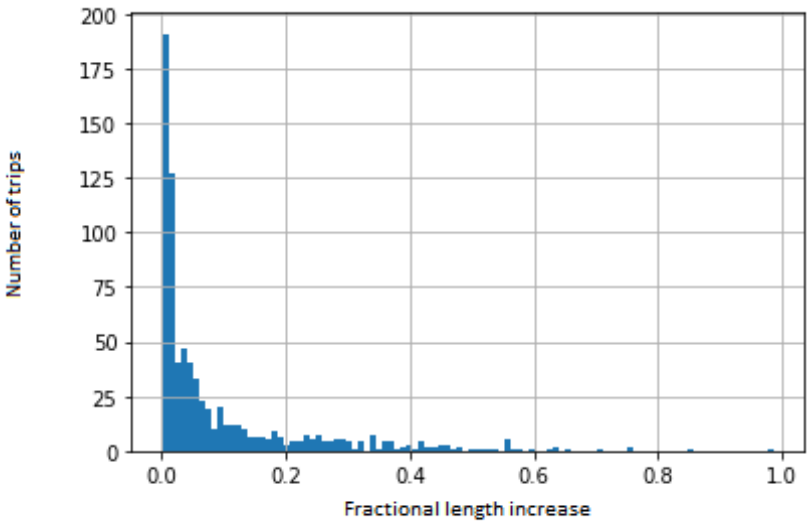


Figure 4.18: EU random long trip (more than 1000 hops) histogram for fraction of length increase in the alternative path compared to the shortest path.

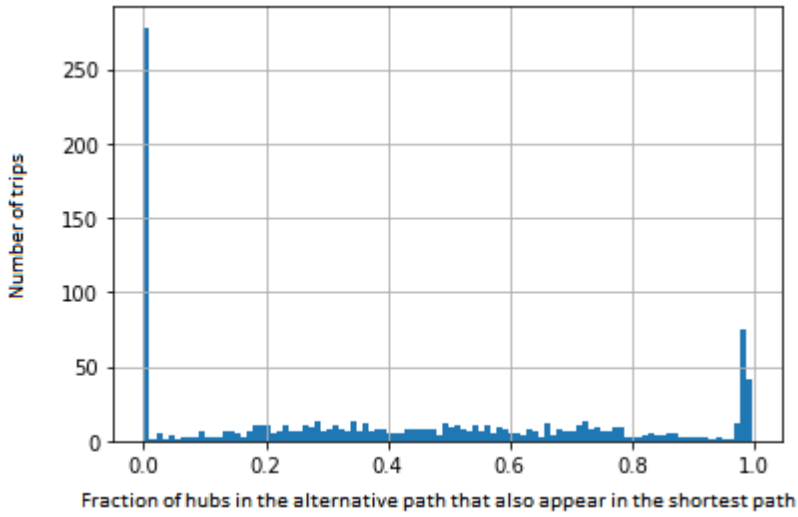


Figure 4.19: EU random long trip (more than 1000 hops) histogram for fraction of hubs in the alternative path that were present in the shortest path.

Chapter 5

Related work

We are unaware of any work comparing Hub Labelling hub priority estimates to real world trip data. However there is an experimental study performed on Hub Labelling by Li, U, Yiu and Kou [20]. There authors mainly compare three vertex ordering techniques. Two of them being degree ordering and betweenness centrality (greedy shortest path cover) ordering. Both of which we have seen here. As well as their newly proposed significant path based ordering. This ordering is much quicker than betweeness centrality ordering, but results in slightly worse labels. They also confirm that degree based ordering performs very poorly on the road graphs. They also compare the effectiveness of label compression techniques proposed by Delling et al. in [1] and [15].

We are also unaware of any work attempting to provide an alternative path based on Hub Labelling. There is work by Abraham, Delling, Goldberg and Werneck [21] proposing fast alternative path finding algorithm with theoretical alternative path optimallity guarantees. Of course it's not as fast as Hub Labelling. Delling, Goldberg, Pajor and Werneck [22] discuss a graph separator based routing engine that among many things can support arbitrary metrics, real-time traffic updates and alternative path finding. The alternative path finding part of this engine was based on the just mentioned work by Abraham et al. [21]. The described routing engine [22] was used in Bing Maps at the time.

Chapter 6

Conclusions

In this work we validated the performance of Hub Labelling algorithm as proposed by Delling et al. [1]. We also demonstrated that shortest path can be recovered efficiently with a slight modification to the labelling algorithm. We experimentally showed that a reasonable alternative path can be recovered from the labels for short to medium distances. This can possibly be improved in future work. Non the less, from our analysis we can see that in most cases Hub Labelling can be used as is to provide an alternative path around one particular hub that we want to avoid (i.e. if there is a road accident). At last we related the hubs selected by Hub Labelling algorithm to real world road usage data. We found that there is a substantial difference. Albeit time graph hub ranking seems to come closer to the real world hub importance. This leads us to ponder if one could come up with a more efficient metric for the graphs that would better match real world hub importance as well as provide smaller labels. As time graphs also tend to yield much smaller labels than distance graphs.

Chapter 7

Future work

Two main directions we see following from this work would be:

- Creating hub labels entirely using real world trip data to investigate if more realistic vertex importance measure would yield smaller labels. As we have seen time based graphs (which reflect real world vertex importance better than distance graphs) yield smaller labels. It would be possible to use the algorithm originally proposed by Cohen et al. [10] for this. One would just need to come up with a good way to amend real world data (i.e. taxi trips) to cover all possible node pairs. So the algorithm by Cohen et al. [10] that uses precomputed all pairs shortest paths and (label) greedily picks hubs that maximise the ratio between new paths covered and number of labels updated would yield valid labels that satisfy the cover property.
- Improving our simple alternative path recovery scheme so that it more reliably returns an alternative path over long distances and possibly offers alternative paths that are more unique and have smaller and better bounded stretch. It should be possible to achieve this by modifying the label construction further.

Bonus future direction not entirely related to this work would be parallelization of the algorithm proposed by Delling et al. [1] that we reimplemented. Because this algorithm offers superb single thread performance, a parallelized version should offer a state of the art algorithm for labelling arbitrary graphs. While it's not immediately obvious how to parallelize the Pruned Labelling algorithm [14], recently Jin, Peng, Wu, Dragan, Agrawal and Ren proposed a parallelized version of it [23]. They also claim improved single thread performance. It might be possible to extend their findings to this Hub Labelling scheme proposed by Delling et al. [1]. At the same time it would be worthwhile to come up with a way to perform label greedy version of the tree SamPG hub importance estimation. As Abraham et al. [13] have shown label greedy ordering offers better performance than path greedy ordering.

Bibliography

[1] D. Delling, A. V. Goldberg, T. Pajor, and R. F. Werneck, "Robust distance queries on massive networks," in *European Symposium on Algorithms*. Springer, 2014, pp. 321–333.

[2] R. Geisberger, P. Sanders, D. Schultes, and D. Delling, "Contraction hierarchies: Faster and simpler hierarchical routing in road networks," in *International Workshop on Experimental and Efficient Algorithms*. Springer, 2008, pp. 319–333.

[3] H. Bast, S. Funke, P. Sanders, and D. Schultes, "Fast routing in road networks with transit nodes," *Science*, vol. 316, no. 5824, pp. 566–566, 2007.

[4] J. Arz, D. Luxen, and P. Sanders, "Transit node routing reconsidered," in *International Symposium on Experimental Algorithms*. Springer, 2013, pp. 55–66.

[5] T. Akiba, Y. Iwata, K.-i. Kawarabayashi, and Y. Kawata, "Fast shortest-path distance queries on road networks by pruned highway labeling," in *2014 Proceedings of the Sixteenth Workshop on Algorithm Engineering and Experiments (ALENEX)*. SIAM, 2014, pp. 147–154.

[6] I. Abraham, D. Delling, A. V. Goldberg, and R. F. Werneck, "A hub-based labeling algorithm for shortest paths in road networks," in *International Symposium on Experimental Algorithms*. Springer, 2011, pp. 230–241.

[7] H. Bast, D. Delling, A. Goldberg, M. Müller-Hannemann, T. Pajor, P. Sanders, D. Wagner, and R. F. Werneck, "Route planning in transportation networks," in *Algorithm engineering*. Springer, 2016, pp. 19–80.

[8] M. Weller, "Optimal hub labeling is np-complete," *arXiv preprint arXiv:1407.8373*, 2014.

[9] M. Babenko, A. V. Goldberg, H. Kaplan, R. Savchenko, and M. Weller, "On the complexity of hub labeling," in *International Symposium on Mathematical Foundations of Computer Science*. Springer, 2015, pp. 62–74.

[10] E. Cohen, E. Halperin, H. Kaplan, and U. Zwick, "Reachability and distance queries via 2-hop labels," *SIAM Journal on Computing*, vol. 32, no. 5, pp. 1338–1355, 2003.

[11] I. Abraham, A. Fiat, A. V. Goldberg, and R. F. Werneck, "Highway dimension, shortest paths, and provably efficient algorithms," in *Proceedings of the twenty-first annual ACM-SIAM symposium on Discrete Algorithms*. Society for Industrial and Applied Mathematics, 2010, pp. 782–793.

[12] A. V. Goldberg, H. Kaplan, and R. F. Werneck, "Reach for a*: Efficient point-to-point shortest path algorithms," in *2006 Proceedings of the Eighth Workshop on Algorithm Engineering and Experiments (ALENEX)*. SIAM, 2006, pp. 129–143.

[13] I. Abraham, D. Delling, A. V. Goldberg, and R. F. Werneck, "Hierarchical hub labelings for shortest paths," in *European Symposium on Algorithms*. Springer, 2012, pp. 24–35.

[14] T. Akiba, Y. Iwata, and Y. Yoshida, "Fast exact shortest-path distance queries on large networks by pruned landmark labeling," in *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*. ACM, 2013, pp. 349–360.

[15] D. Delling, A. V. Goldberg, and R. F. Werneck, "Hub label compression," in *International Symposium on Experimental Algorithms*. Springer, 2013, pp. 18–29.

[16] C. Demetrescu, A. V. Goldberg, and D. S. Johnson, *The Shortest Path Problem: Ninth DIMACS Implementation Challenge*. American Mathematical Soc., 2009, vol. 74.

[17] L. Moreira-Matias, J. Gama, M. Ferreira, J. Mendes-Moreira, and L. Damas, "Predicting taxi–passenger demand using streaming data," *IEEE Transactions on Intelligent Transportation Systems*, vol. 14, no. 3, pp. 1393–1402, 2013.

[18] C. Yang and G. Gidofalvi, "Fast map matching, an algorithm integrating hidden markov model with precomputation," *International Journal of Geographical Information Science*, vol. 32, no. 3, pp. 547–570, 2018.

[19] P. Newson and J. Krumm, "Hidden markov map matching through noise and sparseness," in *Proceedings of the 17th ACM SIGSPATIAL international conference on advances in geographic information systems*. ACM, 2009, pp. 336–343.

[20] Y. Li, L. H. U, M. L. Yiu, and N. M. Kou, "An experimental study on hub labeling based shortest path algorithms," *Proceedings of the VLDB Endowment*, vol. 11, no. 4, pp. 445–457, 2017.

[21] I. Abraham, D. Delling, A. V. Goldberg, and R. F. Werneck, "Alternative routes in road networks," in *International Symposium on Experimental Algorithms*. Springer, 2010, pp. 23–34.

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

[23] R. Jin, Z. Peng, W. Wu, F. Dragan, G. Agrawal, and B. Ren, "Pruned landmark labeling meets vertex centric computation: A surprisingly happy marriage!" *arXiv preprint arXiv:1906.12018*, 2019.