Finding The Most Preferred Path

Dimitris Sacharidis
Institute of Software Technology and Interactive Systems
Technische Universität Wien, Austria
dimitris@ec.tuwien.ac.at

Panagiotis Bouros
Department of Computer Science
Aarhus University, Denmark
pbour@cs.au.dk

Theodoros Chondrogiannis
Department of Computer Science
Free University of Bozen-Bolzano, Italy
tchond@inf.unibz.it

ABSTRACT

Consider a road network, and let the preferred subnet consist of the roads a driver is more acquainted to and hence tends to follow. In this paper, we study the problem of finding the most preferred path between two network nodes; we consider two variants of this problem. We first target the *Most Preferred Unrestricted Path* (MPUP) that has the lowest traveling time in the non-preferred subnet; this problem was introduced in the literature as identifying the safest path though safe zones. As MPUP imposes no constraints on the total traveling time, we then introduce the *Most Preferred Near Shortest Path* (MPNSP) that has the lowest traveling time in the non-preferred subnet among all paths which are not much slower than the shortest path. We focus on the efficient evaluation of both problems by proposing solutions with simple pre-processing steps. An extensive evaluation demonstrates the efficiency of our techniques compared to the existing method for MPUP and to the state-of-the-art on computing multi-criteria shortest paths for MPNSP.

CCS CONCEPTS

• **Information systems → Spatial-temporal systems**;

KEYWORDS

Route Planning, Road Networks, Query Services, Shortest Path, Near-Shortest Path, Multi-criteria Shortest Path

1 INTRODUCTION

The proliferation of navigation devices, such as smartphones with GPS receivers, has renewed the interest in algorithms for obtaining optimal routing (driving or walking) directions. Conventional routing operates under the assumption that traveling time or distance is the most important optimization objective. Hence, a plethora of methods have been proposed that answer shortest path queries in almost constant time, even for continental sized networks; [5, 41] offer a complete survey and an experimental evaluation. In practice however, there exist a number of hard-to-formalize factors that affect people's routing decisions and so, reaching the destination as fast as possible is not necessarily the optimal way of moving. In an effort to deliver personalized routing, a number of research work has focused on extracting moving habits and patterns or popular paths from historical trajectory data, e.g., [12, 13, 17, 30, 32, 42].

In this work, we investigate the computation of the most preferred way of moving between two locations under a setting that has received very little attention in the past. Given a road network, there exist parts of the network that a person is more familiar with or is more interested in moving through in practice; we call this the *preferred* (sub-)network. Essentially, the objective of optimal routing is no longer to reach the destination as fast as possible but to travel as much as possible inside the preferred network.

Motivating scenarios. Every person has some routes or paths on the road network for commuting regularly to work, taking children to school, going to the market or visiting friends. In need of reaching a part of the city for the first time, it is fair to assume that a person would prefer to drive along familiar roads, whenever possible. Such behavior comes very natural as drivers tend not to stray from known paths, and are usually reluctant to explore alternative ways, e.g., out of fear of getting lost. As another scenario, consider a visitor to a city focusing on particular neighborhoods due to their interesting venues and sights or simply because they are safe. It is reasonable to assume that during sightseeing or even when trying to reach a particular city location, this person would like to travel as mush as possible through these neighborhoods. Finally, a person riding a bike for his everyday commuting would find great value in driving as much as possible through parts of the city where dedicated bike lanes are available.

Contributions. We study two variants of the most preferred routing. First, we try to minimize solely the time spent outside the preferred network formulating the *Most Preferred Unrestricted Path* (MPUP). Essentially, computing MPUP resembles a shortest path problem where the cost of moving inside the components of the preferred graph is zero. This problem was introduced in [3, 4] as the safest path via safe zones for Euclidean spaces but also studied for road networks; to the best of our knowledge, this is the most relevant work to ours. The authors proposed the HyperEdges algorithm which employs a dense *hypergraph* with every component of the preferred network serving as a node. However, the algorithm struggles as it traverses not only this hypergraph, but also performs two single-source all-targets shortest path searches on the road network. Further, the expensive offline pre-processing step of constructing the hypergraph, renders this solution inefficient in the presence of updates. To deal with the weakness of HyperEdges, we design a novel approach based on simple pre-processing that compresses the road network and on a single online shortest path search on this compressed network to compute MPUP. Our experiments demonstrate the advantage of our method with respect to both the offline pre-processing and the online computation of MPUP.

SIGSPATIAL'17, November 7–10, 2017, Los Angeles Area, CA, USA

ACM ISBN 978-1-4503-5490-5/17/11…$15.00
https://doi.org/10.1145/3139958.3140029

Under the aforementioned setting, a recommended path may have arbitrary total time as the cost of traveling inside the preferred network is ignored. We next study a more practical scenario where one still prefers to move as little as possible outside the preferred network but at the same time, can only afford a specific increase of the total travel time. For this purpose, we introduce the problem of computing the *Most Preferred Near Shortest Path* (MPNSP); an early study on the problem was conducted in [8]. Note that our MPNSP differs from the safest path via preferred zones (SPPZ) also proposed in [3, 4], but not studied for road networks. Essentially, SPPZ looks for a path that minimizes a linear combination of the time spent inside and outside the preferred network; however, the recommended path may still be arbitrary long and potentially of little value for the user. A more distinguishing feature of MPNSP over SPPZ is that the former looks for the best among *sub-optimal* (i.e., near shortest) paths and thus standard shortest path algorithms are not applicable, while the latter essentially looks for optimal paths after adapting the weights along edges and thus can be accelerated using standard shortest path techniques.

As the quality of a path in the MPNSP problem is measured by two criteria, it is more closely related to finding the set of pareto-optimal paths or *path skyline*. In Section 4.1, we describe a baseline solution to MPNSP, which employs the state-of-the-art path skyline algorithm [27] optimized by pareto-prep [36]. However, as we discuss in Section 4 and verify in Section 5, such a path skyline based approach cannot take advantage of the special characteristics of MPNSP. To this end, we propose a novel algorithm termed ALGO-U which makes some important observations regarding the nature of the problem. This allows ALGO-U to compute and progressively refine an upper bound of the solution and hence, dramatically reducing the search space as shown by our experimental analysis.

Outline. The remainder of this paper is organized as follows. Section 2 introduces notation and defines two variants of the most preferred path. Then, Section 3 addresses the efficient computation of MPUP while Section 4 targets MPNSP. Section 5 presents our experimental evaluation of our methodology for both problems. Finally, Section 6 discusses related work and Section 7 concludes this work.

2 PRELIMINARIES

Section 2.1 presents the necessary background and notation while Section 2.2 formally introduces the problems at hand.

2.1 Notation

First, we define the graph representation of a road network.[1]

Definition 2.1 (Network). The *network* is a undirected weighted graph $G(N, E)$ where N is a set of nodes that represent the road intersections and $E \subset N \times N$ is a set of edges that represent the road segments. Each edge (i, j) is associated with a weight $w_{i,j} \in \mathbb{R}^+$ that captures its traveling time.

A *path* p on the network is a finite sequence of edges which visits a network node at most once. The *total time* T_p of path p is the sum of weights of its edges. The *shortest path* from a source node s to a target node t is a path that has the minimum total time among all possible paths from s to t. This minimum total time from s to t is called the *distance* $d(s, t)$.

[1]For simplicity, we represent a road network as a undirected graph but our ideas can also be applied in case of a directed graph.

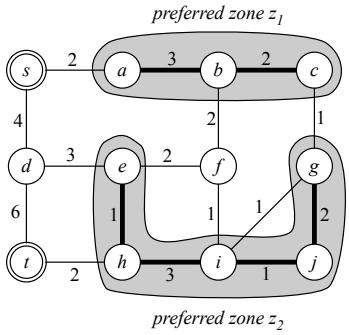


Figure 1: An example of a network G; preferred edges are shown in bold.

We next define the preferred network, which is a subgraph of G.

Definition 2.2 (Preferred Network). The *preferred network* $G_P(N_P, E_P)$ is a subgraph of G where $N_P \subseteq N$, and $E_P \subseteq N_P \times N_P \subseteq E$.

We refer to the edges in G_P as *preferred edges*. The preferred network can be viewed as a set of connected components, which we call *preferred zones*. A node in the preferred graph whose incident edges are all preferred is called a *preferred node*. Not all nodes in the preferred graph are preferred; a node in G_P that is not preferred is called a *border node*. A border node has an incident edge in G which is not contained in G_P. In analogy, we refer to the edges not in E_P as *unpreferred*, and the subgraph of G that contains all such edges as the *unpreferred network*.

All paths considered in this work are on the complete network G. For our purposes, we need to associate a path with the cost of traversing unpreferred edges. Given a path p, its *unpreferred time* U_p is the sum of weights of its unpreferred edges.

Example 2.3. Figure 1 shows a network G of 12 nodes, where the preferred edges are shown in bold. The numbers along each edge represent their weights. The preferred network consists of two zones z_1 and z_2 shown shaded in the figure.

Figure 2 shows six paths on G connecting source s to target t. Each path is plotted as a point in the 2-dimensional total time T_p — unpreferred time U_p plane. For example, path $p_2 = (s, d, e, h, t)$ has total time 10, preferred time 1 (as it travels along the preferred esge (e, h)), and thus unpreferred time 9. Hence p_2 is a point at coordinates 10, 9 in the T_p—U_p plane. Paths p_1 and p_2 have the least total time among all possible s-t paths, and thus are the shortest paths: $T_{p_1} = T_{p_2} = d(s, t) = 10$. □

2.2 Problem Definitions

We now formally define the two variants of identifying the most preferred path. The problem of the *Most Preferred Unrestricted Path* (MPUP) was first introduced in [3, 4] as finding the safest path through safe zones. Here we restate the problem using the terminology of Section 2.1.

Problem 1 (MPUP). *Find a path from source node s to target t that has the least unpreferred time.*

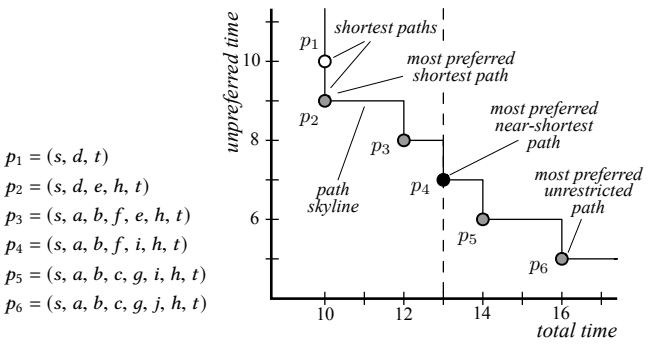


Figure 2: Paths depicted as points under two criteria: total time and unpreferred time.

Intuitively, MPUP directs a driver to travel as mush as possible inside the preferred network or as little as possible inside the unpreferred network. Equivalently, Problem 1 can be seen as a shortest path problem on the complete network G by setting the weights of all preferred edges to zero.

Problem 1 allows for paths that have arbitrary total traveling time while the time spend inside the preferred zones is not taken into account. In view of this, Problem 2 investigates a more practical setting where the total time of the returned path is restricted with respect to the source-to-target distance.

Before introducing this problem, we present the notion of a near-shortest path. Given a parameter $\epsilon \in [0, 1]$, a path p from a node s to t is called *ϵ-near shortest* if its total time is not greater than $(1+\epsilon)$ times the distance of s to t, i.e., $T_p \leq (1+\epsilon) \cdot d(s, t)$. In other words, a near shortest path is a relaxation allowing for a small controlled deviation from the optimal traveling time.[2] We define the problem of identifying the *Most Preferred Near Shortest Path* (MPNSP).

Problem 2 (MPNSP). *Find a ϵ-near shortest path from source node s to target t that has the least unpreferred time.*

Example 2.4. Returning to our example, observe from Figure 2 that path p_6 has the least unpreferred time $U_{p_6} = 5$ among all s-t paths. Therefore, under no restrictions to total time, p_6 is the MPUP.

As discussed the distance between source and target is $d(s, t) = 10$. Now, suppose we are only interested in paths that are at most $\epsilon = 30\%$ longer than the shortest path. In our case, this translates to paths with total time not more than 13. In the T_p—U_p plane of Figure 2, this restriction means that we should only consider paths that lie left of the dashed vertical line at $T = 13$. Among the near shortest paths p_1, p_2, p_3, p_4, observe that p_4 has the least unpreferred time 7 and is this the MPNSP.

It is worth noting that shortest paths p_1, p_2, MPUP p_6 and MPNSP p_4 capture different optimality criteria over total and unpreferred time. The first two optimize total time so that the drive is as short as possible; p_6 optimizes the unpreferred time so that the driver drives as much as possible in the familiar network; while p_4 optimizes the unpreferred time subject to a total time constraint, striking a reasonable balance between driving in the familiar network and getting to the destination fast. □

[2] Although near shortest paths are defined with respect to a relative deviation, e.g., to travel at most 10% longer, it also allows for absolute deviations, e.g., when we want to travel at most 5 minutes longer.

3 COMPUTING MPUP

We first target the efficient evaluation of Problem 1. Section 3.1 briefly revisits the solution proposed in [3, 4] while Section 3.2 details finding the path with the least unpreferred time.

3.1 The HyperEdges Algorithm

In [3, 4], the authors primarily focused on an Euclidean setting for MPUP employing the geometric properties of hyperbolas; however, an adaptation of the proposed solution for spatial road networks was also discussed. Essentially, computing MPUP involves two phases.

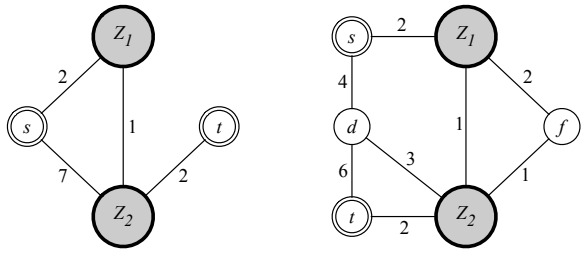
Offline phase. An undirected weighted *hypergraph* G_0 is constructed offline such that each preferred zone serves as a node. A *hyperedge* connecting two nodes in G_0 captures the best way of driving between the corresponding preferred zones. A naïve and thus impractical approach would create a hyperedge for every pair of preferred zones resulting in an extremely dense hypergraph. Instead, the authors of [3, 4] designed a labeling technique which connects two preferred zones only if the shortest path between them on network G (i.e., between a pair of border nodes) does not cross a third preferred zone. In particular, a single-source all-targets shortest path search is initiated on G for each preferred zone, starting from all its border nodes. Every network node encountered during this search is labeled with the *id* of the last zone crossed. The search terminates when all network nodes are labeled and then, the neighboring zones of the examined zone are determined.

Online phase. Given an MPUP query, the hypergraph G_0 is first expanded to include two new preferred zones: the source node s and the target t. For this purpose, a single-source shortest path search on network G is initiated from source node s and another from target t, towards the border nodes of the preferred zones. Then, to determine the most preferred unrestricted path, a shortest path search is performed on the extended hypergraph G_0 between the preferred zones of s, t. Finally, as a special case arises when the path with the least unpreferred time does not cross any preferred zone, a second shortest path search from s to t is required on network G.[3]

Example 3.1. Figure 3a illustrates the approach of [3, 4] using our running example. In the offline phase, the hypergraph connecting the preferred zones is constructed. In our case, there exist only two preferred zones, so the offline hypergraph contains two nodes z_1 and z_2 (representing the two zones) and a single edge connecting them. Using their labeling technique, one can find that this edge has weight 1; this essentially corresponds to the shortest path (c, g) in the complete network between a border node of z_1 and one of z_2.

In the online phase, when the source and target nodes are known, they must also be connected via shortest paths (in the complete network) to zones z_1 and z_2. Observe from Figure 1 that (s, a) (resp. (s, d, e)) is the shortest path connecting the source to zone z_1 (resp. z_2) with a total time of 2 (resp. 7). Hence the weights in the hypergraph of Figure 3a. Using similar reasoning, target t is connected to z_2 via the shortest path (t, h) of total time 2. The shortest path

[3] This last shortest path search can be in fact incorporated to the single-source search which connects the preferred zone of source s to hypergraph G_0.



(a) Hyperedges **(b) Compressed Network**

Figure 3: Approaches for finding the MPUP.

(t, h, f, b) from t to zone z_1 passes via zone z_2 and thus an edge is not created.

Finding the MPUP then translates to finding a shortest path on the hypergraph shown in Figure 3a. This is path (s, z_1, z_2, t) with a length of 5, which expands to path p_6 in the complete graph with unpreferred time 5. □

3.2 The Compressed Network Approach

Despite constructing hypergraph G_0, the HyperEdges algorithm still needs to traverse network G. Overall, the method performs two single-source all-targets shortest path searches on G and a shortest path search on G_0. In fact, contrary to traditionally sparse road network graphs, the hypergraph is expected to be dense which may further impact the efficiency of the online phase. In addition, the HyperEdges algorithm involves an expensive pre-processing step to determine the set of hyperedges that connect the preferred zones. Although this step occurs offline, the resulting hypergraph G_0 needs to be maintained whenever new preferred zones are defined and when existing are dropped or altered; handling these updates requires a number of single-source all-targets shortest path searches on the network.

In view of these shortcomings, we devise a novel solution to MPUP with a simple and inexpensive pre-processing step which employs online a single shortest path search. As the traveling time inside a preferred zone is ignored by Problem 1, the key insight of our approach is to completely exclude from the search the preferred edges of network G. In particular, we first construct offline a *compressed* network G' by reducing every preferred zone to a single zone node; this new zone node has incident edges to every unpreferred node previously connected to a border node of the reduced zone.[4] Given an MPUP query, the path with the least unpreferred time on the original network G can be now computed by a shortest path search from source s to target t on the compressed network G'.

Example 3.2. In the example network of Figure 1, there exist two preferred zones. Thus to construct the compressed network G', depicted in Figure 3b, we create two nodes z_1, z_2 representing the two zones. All unpreferred nodes, s, d, f, t, are also present in the compressed network. Preferred edges are not represented in G'. On the other hand, edges between unpreferred nodes are preserved in the compressed network. Moreover, every edge in the complete network that connects an unpreferred node to a zone's border is replaced in the compressed network by an edge between that node and the zone node. Because node f has two edges (f, e), (f, i) to zone z_2 with weights 2, 1, respectively, we create an edge (f, z_2) in $G'1$ with minimum weight 1.

The shortest s-t path in G' is (s, z_1, z_2, t) with a length of 5, and corresponds to path p_6 in the complete graph with unpreferred time 5. □

[4] A similar idea is employed for graph reachability where every (strongly) connected component of the (directed) graph is reduced to a single super-node.

3.3 Discussion

Due to performing a single shortest path search, we expect our compressed network solution to always outperform the HyperEdges algorithm in evaluating MPUP queries. We also expect a significantly faster pre-processing phase while maintaining the compressed network G' will be by far more efficient compared to hypegraph G_0 as in practice, we only need to add or remove network edges instead of initiating single-source all-targets shortest path searches.

Both approaches can benefit from a pre-processing technique (e.g., the contraction hierarchies from [18]) that speeds up shortest path search either on the original or the compressed network. We elaborate on this idea in our experimental analysis in Section 5.

4 COMPUTING MPNSP

We next turn our focus to Problem 2. Section 4.1 discusses a baseline solution that builds upon path computation on multi-criteria networks while Section 4.2 details our methodology for finding the ϵ-near shortest path with the least unpreferred time.

4.1 A Path Skyline Based Approach

A straightforward approach is to treat MPNSP as a path-computation problem on a multi-criteria network. In this context, the goal is to find all paths which are optimal according to *any* possible preference function combining the criteria. These paths constitute a *pareto-optimal* set, termed the *path skyline*, borrowing the terminology from the skyline operator literature [7]. Assume paths p, p' from source s to target t; path p *dominates* p' if it is at least as good as p' on all criteria, and strictly better on at least one. The set of all not dominated paths constitutes the path skyline.

Returning to the example of Figure 2, observe that path p_1 is dominated by p_2. The path skyline consists of paths $\{p_2, p_3, p_4, p_5, p_6\}$, shown as filled points along the solid line in the figure. Naturally, the two most preferred paths (unrestricted and near shortest) are in the path skyline. Now, suppose we retrieve the path skyline. Then, as discussed, MPNSP query introduces a threshold $(1 + \epsilon) \cdot d(s, t)$ on the total time axis, depicted by the dashed vertical line. Skyline paths p_5, p_6 to the right of this line do not qualify as ϵ-near shortest paths; their total time is too high compared to $d(s, t)$. Among the skyline paths to the left of the threshold line, p_4 has the least unpreferred time and hence can be returned as the MPNSP.

The ARSC Algorithm. To compute the path skyline, we use the state-of-the-art Advanced Route Skyline Computation (ARSC) algorithm proposed in [27]. Label-correcting ARSC traverses network G in an A* manner until all paths in the skyline are found. When expanding a path p from source s to node n, the algorithm applies two pruning rules to eliminate unpromising paths. The first rule defines a best-case extension of p; if a path contained in the skyline dominates this hypothetical best-case extension of p then path p

is unpromising and hence, pruned. The second pruning rule compares the already computed paths to node n with the currently examined path p; all paths that are dominated are eliminated. Essentially, this rule extends the principle of optimality in shortest path computation to multiple criteria.

Bound Computation with ParetoPrep. To perform an A* traversal of the network and to apply the first pruning rule, ARSC heavily relies on lower bounds for the total and the unpreferred time. For this purpose, the authors in [27] compute offline a Lipschitz reference embedding; lower bounds are then calculated online based on the triangular inequality property. However, studies have shown that the approximation quality of these bounds is insufficient; in fact, computing for each criterion the optimal query-specific bounds through an online single-source shortest path search from target t to all networks nodes, leads to a significant speed-up of the path skyline computation [39]. In this work, we compute the required lower bounds using the ParetoPrep approach proposed in [36]. ParetoPrep computes the optimal lower bounds for the total and unpreferred time by traversing network G only once, handling both criteria at the same time.

4.2 The ALGO-U Algorithm

The ARSC algorithm first extracts the entire path skyline and then identifies the MPNSP among those paths. In this section, we aim for a direct approach that cleverly guides the search towards the MPNSP path. Briefly, the main idea of ALGO-U is to first perform a reverse search, starting from the target and reaching the source, that optimizes for total time, and then a forward search that optimizes for the preferred time. Information stored at the nodes visited during the reverse search is used to guide the forward search.

Before presenting ALGO-U, we introduce some notation and additional definitions.

Labels and Orders. A label λ associated with a path p represents its two costs. It has an entry $\lambda.T$ for the total time of p, and an entry $\lambda.U$ for the unpreferred time of p. We write $\lambda(p)$ to explicitly refer to p's label.

There are two possible lexicographic orders of labels. TU-order, denoted as $<_{TU}$, orders (increasingly) by total time and in case of ties (increasingly) by unpreferred time. That is a label λ is before another λ' if the former has less total time, or equal total time but less unpreferred time. On the other hand, UT-order, denoted as $<_{UT}$, orders by unpreferred time and in case of ties by total time.

These orders are useful when we need to distinguish among paths having equal total time (or equal unpreferred time). Fix a source s and target t, and let $d(s,t)$ be the distance between them. There can exist different shortest paths from s to t having exactly the same total time $d(s,t)$. In defining the ALGO-U algorithm later, we are interested in the least unpreferred time achieved the shortest paths. Observe, that the s-to-t path ranked first by the TU-order exhibits this optimal unpreferred time among the shortest paths. We refer to this path as a *most preferred shortest path*. Returning to the example of Figure 2, while both p_1 and p_2 are shortest paths, path p_2 is a most preferred shortest path, having lower unpreferred time than p_1 and coming before it in the TU-order.

In analogy, we define a *shortest most preferred (unrestricted) path* as a path (among all source to target paths) with the best label according to the UT-order.

Finally, we say that a label λ *dominates* another λ', denoted as $\lambda \prec \lambda'$ if λ precedes λ' according to both TU-order and UT-order, i.e., $\lambda <_{TU} \lambda'$ and $\lambda <_{UT} \lambda'$.

Pruning Paths. The next lemmas allow us to eliminate paths from consideration during the forward search. They require certain total time and unpreferred time computations, which as we see are computed during the reverse search.

The first lemma prunes paths starting from source s that when extended to reach target t result in paths which exceed the total time threshold of $(1+\epsilon)\cdot d(s,t)$.

Lemma 4.1. *Assume a path p from s to node n, and let $d(s,t)$ denote the distance from s to t, and $d(n,t)$ the distance from n to t. Then, if condition $T_p + d(n,t) > (1+\epsilon)\cdot d(s,t)$ holds, no path extending p to t can be an* MPNSP *solution.*

The next lemma eliminates dominated paths and is also the key pruning rule of the ARSC algorithm.

Lemma 4.2. *Assume two paths p, p' from s to node i. If $\lambda(p) \prec \lambda(p')$ no path extending p' to t can be an* MPNSP *solution.*

The following lemmas consider the two possible optimal extensions of a path ending at node n. The first, called the *TU extension*, is via a most preferred shortest path $p_{TU}(n)$ from n to t, optimizing total time primarily and unpreferred time in case of ties, or equivalently having the minimum TU label. Let $TU(n).T$ and $TU(n).U$ denote the total time and unpreferred time of this path.

The second, called the *UT extension*, is via a shortest most preferred (unrestricted) path $p_{UT}(n)$ from n to t, optimizing unpreferred time primarily and total time in case of ties, or equivalently having the minimum UT label. Let $UT(n).T$ and $UT(n).U$ denote the total time and unpreferred time of that path.

The next lemma computes an upper bound $\overline{U^*}$ to the unpreferred time of the MPNSP solution. It considers the TU and UT extensions discussed previously. If any of them results in a valid path (near shortest), then its unpreferred time upper bounds $\overline{U^*}$.

Lemma 4.3. *Assume a path p from s to node n. Then, the following holds for the unpreferred time U^* of the* MPNSP *solution:*

$$U^* \leq \min \begin{cases} U_p + TU(n).U, & \text{if } T_p + TU(n).T \leq (1+\epsilon)\cdot d(s,t) \\ U_p + UT(n).U, & \text{if } T_p + UT(n).T \leq (1+\epsilon)\cdot d(s,t) \end{cases}$$

Given such an upper bound $\overline{U^*}$, the next lemma prunes any path reaching node n from source s that when UT extended to target t (i.e., with the optimal extension in terms of unpreferred time) results in a suboptimal path, i.e., with unpreferred time greater than the upper bound.

Lemma 4.4. *Assume a path p from s to node n, and let $\overline{U^*}$ be an upper bound of the* MPNSP *solution. Then, if condition $U_p + UT(n).U > \overline{U^*}$ holds, no path extending p to t can be an* MPNSP *solution.*

The last lemma provides a stronger criterion than Lemma 4.2. If the UT extension of a path ending at node n is a near shortest path, then any other path reaching n with worst label in the UT order can only be extended to a path with higher unpreferred time.

Algorithm 1: Algorithm ALGO-U

Input: MPNSP(s, t, ϵ); network G
Output: path $p(s, \ldots, t)$ on G with lowest U_p and $T_p \le \alpha \cdot d(s,t)$
Variables: priority queue Q with entries $(n, p, \lambda, \widehat{\lambda})$ in ascending $\le_{UT}$ order of $\widehat{\lambda}$;
set $\Lambda[n]$ of labels in Q associated with node n;
upper bound $\overline{U}$ on unfamiliar time of the solution

1 $\{UT(n), TU(n)\} \leftarrow$ **execute** ReverseSearch(s, t, ϵ);
2 $\overline{U^*} \leftarrow TU(s).U$;
3 **insert** $(s, (s), \langle 0, 0\rangle, \langle TU(s).T, UT(s).U\rangle)$ in Q;
4 **insert** $\langle 0, 0\rangle$ in $\Lambda(s)$;
5 **while** Q *is not empty* **do**
6 $(n, p, \langle T_p, U_p\rangle, \widehat{\lambda}) \leftarrow$ pop Q;
7 **if** $n = t$ **then**
8 $\overline{U^*} \leftarrow U_p$; $p^* \leftarrow p$; ▷ found solution
9 **break**
10 **if** $T_p + UT(n).T \le (1+\epsilon) \cdot TU(s).T$ **then**
11 **mark** n as *closed* ▷ Lemma 4.5
12 **foreach** $(n, n') \in G$ *such that* n' *is not* closed **do**
13 $p' \leftarrow p \cup (n, n')$; $T_{p'} \leftarrow T_p + w_{n,n'}$; $U_{p'} \leftarrow U_p$;
14 **if** $(n, n') \in G_U$ **then** $U_{p'} \leftarrow U_p + w_{n,n'}$;
15 $\lambda' \leftarrow \langle T_{p'}, U_{p'}\rangle$; ▷ create new label
16 $\widehat{\lambda'} \leftarrow \lambda' + \langle TU(n').T,\ UT(n').U\rangle$; ▷ compute predicted label
17 **if** $T_{p'} + TU(n').T > (1+\epsilon) \cdot TU(s).T$ ***or*** ▷ Lemma 4.1
18 $U_{p'} + UT(n').U > \overline{U}$ ***or*** ▷ Lemma 4.4
19 $\exists \lambda \in \Lambda(n') : \lambda \prec \lambda'$ **then continue**; ▷ Lemma 4.2
20 **else**
21 **insert** $(n', p', \lambda', \widehat{\lambda'})$ in Q;
22 **insert** λ' in $\Lambda(n')$;
23 **if** $T_{p'} + UT(n').T \le (1+\epsilon) \cdot TU(s).T$ **then** ▷ update $\overline{U^*}$ by Lemma 4.3
24 $\overline{U^*} \leftarrow \min\{\overline{U^*}, U_{p'} + UT(n').U\}$;
25 **else**
26 $\overline{U^*} \leftarrow \min\{\overline{U^*}, U_{p'} + TU(n').U\}$;
27 **foreach** $\lambda \in \Lambda[n']$ **do** ▷ remove unpromising labels
28 **if** $\lambda.U + UT(n').U > \overline{U^*}$ ***or*** $\lambda' \prec \lambda$ **then**
29 **remove** λ from $\Lambda(n')$;
30 **remove** entry for λ from Q
31 **return** p^*

Lemma 4.5. *Assume two paths p, p' from s to node n. If the UT extension of p is a near shortest path, i.e., $T_p + UT(n).T \le (1+\epsilon)\cdot d(s,t)$, and $\lambda(p') \ge_{UT} \lambda(p)$, then no path extending p' to t can be an* MPNSP *solution.*

Algorithm Description. In order to apply the previous pruning rules, we need the distance of every node to target (Lemmas 4.1, 4.3, 4.5), and the UT and TU extensions of every node to target (4.3, 4.4, 4.5). Since the distances are equal to the total time of the TU extensions, it turns out we only need to compute the extensions.

In the reverse search, ALGO-U computes the UT and TU extension of every node to target. This entails storing for each node n, four values: the total time $UT(n).T$ and unpreferred time $UT(n).T$ of the UT-optimal (shortest most preferred) path from t to n, and the total time $TU(n).T$ and unpreferred time $TU(n).T$ of the TU-optimal (most preferred shortest) path from t to n. This step can be executed using any standard single-source shortest path algorithm with a small twist: path optimality is defined according to the UT or TU lexicographic order (instead of total or unpreferred time).

Algorithm 1 shows the pseudocode of ALGO-U. The first step is the reverse search procedure (line 1) that computes values $UT(n).T$, $UT(n).U$, $TU(n).T$, $TU(n).U$ for every node n. Note also that an upper bound to the MPNSP solution's unpreferred time can be immediately set as the unpreferred time of the most preferred shortest path from s to t (line 2), and that distance $d(s,t)$ is equal to $TU(s).T$.

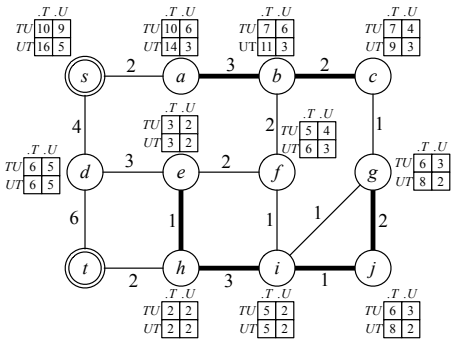


Figure 4: The annotated network resulting from the reverse search of ALGO-U.

Then, the forward search commences. This uses a priority queue Q containing entries of the form $(n, p, \lambda, \widehat{\lambda})$ representing a path p from s to node n having label λ. Note that it is not necessary to store the entire path along each label; as is common, it suffices to just store the predecessor of n in the path. Predicted label $\widehat{\lambda}$ is a lower bound of the total and unpreferred time required to reach the target t. ALGO-U is a label setting algorithm akin to A*, and at each step dequeues from Q the entry with the first, in UT order, predicted label. However unlike A*, ALGO-U maintains multiple labels per node; these are stored in list $\Lambda(n)$ for node n.

Initially, ALGO-U creates an entry for source s and inserts into the queue (line 3). Moreover it inserts a label for s in its label list $\Lambda(s)$ (line 4). Then, the algorithm proceeds in iteration extracting at each an entry from the queue (lines 5–31) until the queue is depleted or the dequeued entry (line 6) corresponds to the target (lines 7–9). At that point the MPNSP solution is found, as no other path can reach the target with lower unpreferred time. Otherwise, let n be the current node. The algorithm checks if the conditions of Lemma 4.5 apply for node n; if yes it marks node n as closed so as to discard all other paths, not yet discovered, leading to it.

Subsequently, ALGO-U examines each non-closed neighbor n' of node n (lines 12–30). First, it constructs a new path p' extending the current path with edge (n, n') and prepares its entry (lines 13–16). The predicted label for p' is its label incremented by the least possible total time $TU(n').T$ and the least possible preferred time $UT(n').U$ required to reach the target (line 16). The next step is to check whether path p' should be pruned according to Lemmas 4.1, 4.2, and 4.4 (lines 17–19).

If path p' is not pruned (lines 20–30), then its entry and label are inserted in the queue and the label list of n', respectively (lines 21–22). Subsequently, the upper bound $\overline{U^*}$ on the preferred time of the solution is updated (lines 23–26), by considering the UT (line 24) and TU extensions of p' (line 26). The final step in an iteration is to remove entries and labels corresponding to other paths either dominated or due to a tighter $\overline{U^*}$ (lines 27–30).

Example 4.6. We describe ALGO-U on our running example network, also depicted in Figure 4. The first step of ALGO-U is to perform a reverse search from the target t and compute for each node n four values, the total and unpreferred time of the most

preferred shortest path from t, and the total and unpreferred time of the shortest most preferred path, i.e., $TU(n).T$, $TU(n).U$, $UT(n).T$, $UT(n).U$. These values are depicted in the tables near each node in Figure 4. For example for node f, the most preferred shortest path from t is (t, h, e, f) with a total time of 5 and unpreferred time 4; these are the entries in the first row of the table for f. The second row suggests that there exists a slightly longer path (t, h, i, f) of total time 6, which has however lower unpreferred time 3.

Upon execution of the reverse search, ALGO-U establishes that the s-t distance is 10 and thus the total time threshold is set to 13 ($\epsilon = 30\%$). Also it sets the upper bound $\overline{U^*}$ of MPNSP's unpreferred time to 9 which comes from the most preferred shortest path. The first entry enheaped is for the source: $(s, \langle 0, 0\rangle, \langle 10, 5\rangle)$, meaning that the best total and preferred time to reach t via s is 10 and 5, respectively. When this entry is deheaped ALGO-U considers s's neighbors a and d. For the former, an entry $(a, \langle 2, 2\rangle, \langle 12, 5\rangle)$ is created since the unpreferred edge (s, a) has weight 2, and the least possible total time to reach s from a is $TU(a).T = 10$ while the least unpreferred time is $UT(a).U = 3$. This entry cannot be pruned and thus is enheaped. Also $\overline{U^*}$ is updated to 8, because the current path (s, a) can be TU extended to a s-t path that has $2+TU(a).T = 12$ total time, thus is near shortest, and unpreferred time $2 + TU(a).U = 8$. Similarly for neighbor d, an entry $(d, \langle 4, 4\rangle, \langle 10, 9\rangle)$ is enheaped.

The next entry to deheap is the one that has the smallest predicted label in UT order, and that is the entry for node a that has a single neighbor b. Thus an entry $(b, \langle 5, 2\rangle, \langle 12, 5\rangle)$ is enheaped, which is subsequently deheaped. Node b is connected to c and f. The entry for c is $(c, \langle 7, 2\rangle, \langle 14, 5\rangle)$ is pruned by Lemma 4.1 as extending this path towards t results in a total time of at least 14 exceeding the threshold. On the other hand entry $(f, \langle 7, 4\rangle, \langle 12, 7\rangle)$ is enheaped.

The heap contains entries for nodes d and f with the latter having a lower predicted unpreferred time to reach t. Therefore ALGO-U examines f's neighbors i, e. For the first, an entry $(i, \langle 8, 5\rangle, \langle 13, 7\rangle)$ is enheaped. Moreover, the upper bound $\overline{U^*}$ is decreased to 7, because there exists an extension of the current path from i to s that has acceptable total time $8 + TU(i).T = 13$ and unpreferred time $5 + TU(i).U = 7$. The entry for node e is $(e, \langle 9, 6\rangle, \langle 12, 8\rangle)$ but is pruned by Lemma 4.4 as its best possible unpreferred time 8 exceeds the upper bound.

Continuing its execution, ALGO-U deheaps the entry for node i, enheaps entry $(h, \langle 11, 5\rangle, \langle 13, 7\rangle)$, but prunes entries $(g, \langle 9, 6\rangle, \langle 15, 8\rangle)$, $(j, \langle 9, 5\rangle, \langle 15, 7\rangle)$ as they cannot be extended to near shortest paths. Subsequently, the entry for h is deheaped. Node h is connected to e whose entry $(e, \langle 12, 5\rangle, \langle 15, 7\rangle)$ is again pruned by Lemma 4.1, and to the target with entry $(t, \langle 13, 7\rangle, \langle 13, 7\rangle)$. The heap now contains entries for nodes d and t, where the latter has lower best possible unpreferred. Finally, upon deheaping t's entry, ALGO-U terminates having reached the target. □

5 EXPERIMENTAL ANALYSIS

This section reports our experimental evaluation. Section 5.1 details the setup of our analysis. Sections 5.2 and 5.3 compare for MPUP our compressed network approach denoted by CN against the HyperEdges algorithm denoted by HE. Finally, Section 5.4 compares for MPNSP the ALGO-U algorithm against the path skyline baseline approach denoted by P-SKY. All algorithms were implemented in C++ and the tests run on a Quad-Core Intel(R) Xeon(R) CPU E5-2667 v3 @ 3.20GHz with 96GBs of RAM running Ubuntu Linux.

5.1 Setup

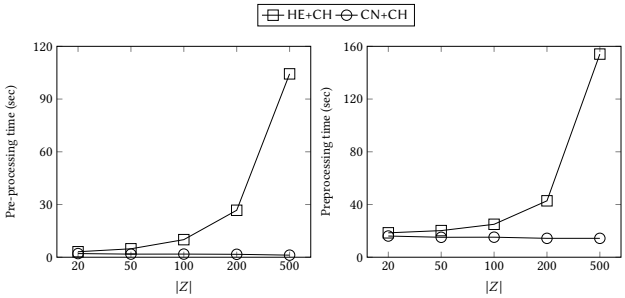
Our analysis involves the real-world road networks of two cities; Berlin with 37,126 nodes and 102,260 edges, and New York with 264,346 nodes and 730,100 edges. To conduct our experiments, we generated a number of preferred networks; the idea is the following. Normally, i.e., if we exclude professionals like taxi drivers, people drive around specific locations or parts of a city. For instance, they move around the location of their house, their work place, their children's school etc. In other words, a driver is familiar with the road segments on specific neighborhoods. To capture this behavior, we first partition the road networks to a predefined number of 1,024 neighborhoods and then, randomly select the center of $|Z|$ among them to populate our preferred zones. In particular, we add to each zone the network nodes who shortest path from the zone center is at most equal to a radius r. This generation procedure shares some commonalities with [3, 4]; however, the centers of our zones are based on the clustering of the road network instead of using a predefined set of nodes where police stations are located.

To assess the performance of the tested methods, we measure their response time on 1,000 MPUP and 1,000 MPNSP queries between randomly selected source and target nodes, varying the number of preferred zones $|Z|$ inside range $\{20, 30, 100, 200, 500\}$ and radius r inside $\{500, 1{,}000, 1{,}500, 2{,}000, 2{,}500\}$ in meters. For the MPNSP queries we also vary parameter ϵ inside range $\{0.1, 0.2, 0.3, 0.4, 0.5\}$; $\epsilon = 0.3$ means that we allow the total time of a path to be at most 30% higher than the shortest path's. On each experiment, we vary one of $|Z|$, r, ϵ while fixing the others to their default value; 100 for $|Z|$, 1,500 for r and 0.3 for ϵ. Finally, for the HE, CN algorithms we also measure the cost of their offline pre-processing phase.

As discussed in Section 3.3, the performance of both CN and HE can be enhanced by techniques like the contraction hierarchies proposed in [18], which accelerate the shortest path search. We denote by CN+CH and HE+CH the versions of our compressed network approach and the HyperEdges algorithm that use contraction hierarchies. We experimented with all for methods for MPUP, but in order to keep our figures clear we only plot the measurements for CN+CH and HE+CH. Nevertheless, we observed as expected, a significant drop of the query evaluation time of CN and HE, in the expense of a slightly longer pre-processing phase. Note that the algorithms for MPNSP cannot be accelerated using similar techniques, as they do not look for optimal paths.

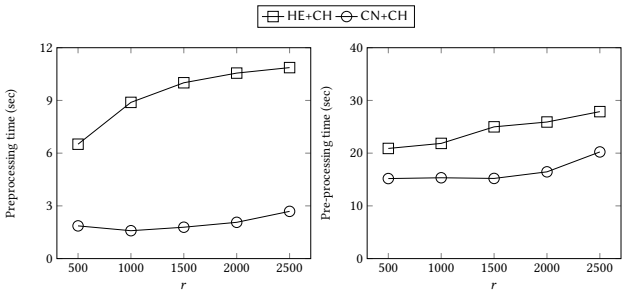
5.2 Pre-processing for MPUP

Figures 5 and 6 report the pre-processing time of HE+CH and CN+CH varying the number of safe zones and the radius of preferred zones. First in Figure 5, we observe that the pre-processing cost of HE+CH is clearly higher compared to CN+CH. The two approaches have comparable pre-processing time only for small number of preferred zones. However, when $|Z| > 50$, CN+CH's pre-processing is 1 to 2 orders of magnitude more expensive. In fact, the pre-processing time of our CN+CH is almost constant as it is dominated by the cost of building the contraction hierarchies on



(a) Berlin **(b) New York**

Figure 5: Pre-processing time varying # zones (r = 1,500 m).



(a) Berlin **(b) New York**

Figure 6: Pre-processing time varying radius ($|Z|$ = 100).

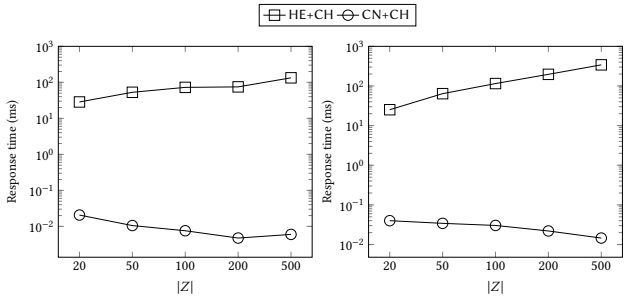
the original road network and also independent of $|Z|$. In contrast, we observe that the time of HE+CH is increasing with the number of preferred zones as a larger number of shortest path searches is required to connect the larger number of hypegraph nodes.

Second, in Figure 6 we observe that the pre-processing time of CN+CH is clearly lower than the time of HE+CH. Similar to Figure 5, the pre-processing cost of CN+CH is almost constant and unaffected by radius r. On the other hand, HE+CH is affected by the increase of r. Large preferred zones tend to contain more border nodes which increases the cost of the shortest path searches needed to define the hyperedges. Nevertheless, the increase of the pre-procesing time is not as abrupt as in Figure 5 as the number of hypergraph nodes remains fixed and equal to $|Z|$ = 100.

5.3 Computing MPUP

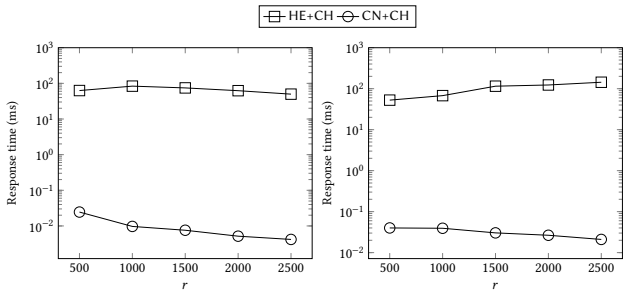
Figure 7 shows the response time of HE+CH and CN+CH varying the number of preferred zones. We observe that CN+CH outperforms HE+CH for both road networks and all setups by 3 to 4 orders of magnitude. Another observation is that while the response time of HE+CH increases with $|Z|$, the response time of CN+CH decreases. For HE+CH, as the number of preferred zones increases, the cost of connecting the source and the target query nodes to the hypergraph rises. On the other hand, the size of the compressed network becomes increasingly smaller benefiting CN+CH.

Figure 8 shows the response time of HE+CH and CN+CH for 100 preferred zones varying the radius of preferred zones from 500 to 2500 meters. Similar to Figure 7, CN+CH outperforms HE+CH for both road networks and all setups by three to four orders of magnitude. Furthermore, we observe a similar behavior regarding the performance of CN+CH. In this case, large preferred zones



(a) Berlin **(b) New York**

Figure 7: Query response time varying # zones (r = 1,500 m).



(a) Berlin **(b) New York**

Figure 8: Query response time varying radius ($|Z|$ = 100).

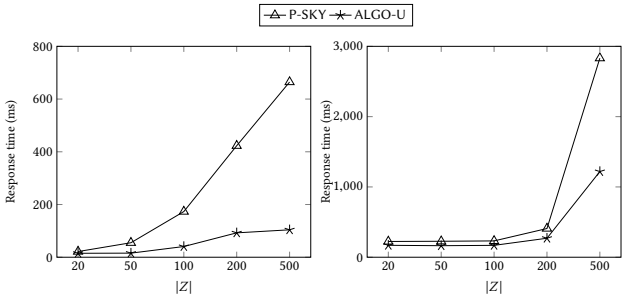
lead to a smaller compressed network; hence, the response time of CN+CH improves with an increasing radius, i.e., larger preferred zones. Lastly, we observe that the performance of HE+CH is not affected significantly by the size of the radius as it is by the number of preferred zones.

A general observation is that CN+CH clearly outperforms HE+CH. In practice, CN+CH reduces the MPUP query to a single shortest path query that is processed with CH, a state-of-the-art method. In contrast, HE+CH requires a large number of shortest path queries to connect the source and the target, and traverse the hypergraph.

5.4 Computing MPNSP

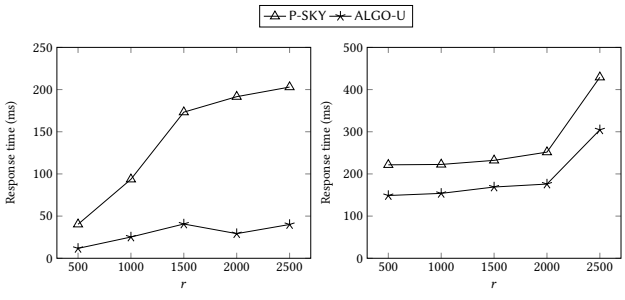
Figure 9 shows the response time of P-SKY and ALGO-U varying the number of preferred zones from 20 to 500 with a fixed radius of 1500 meters and $\epsilon = 0.3$. First, for the road network of Berlin we observe that ALGO-U outperforms P-SKY. The performance of P-SKY degrades quite abruptly with an increasing number of preferred zones. The response time of ALGO-U is also increasing, however, at a significantly lower rate; e.g., when the number of safe zones is 500, ALGO-U is approximately six times faster than P-SKY. Next, for the much larger New York network, we observe that the performance of both algorithms degrades much more abruptly than in Berlin. For 50 to 200 safe zones the response time of both algorithms is quite low with ALGO-U being slightly faster. For 500 safe zones their response time increases considerably, but ALGO-U is three times faster having about one second response time.

Figure 10 shows the response time of P-SKY and ALGO-U for 100 preferred zones and $\epsilon = 0.3$ varying the radius of preferred zones from 500 to 2500 meters. We observe that ALGO-U outperforms P-SKY for both road networks and all radii, the margin being greater



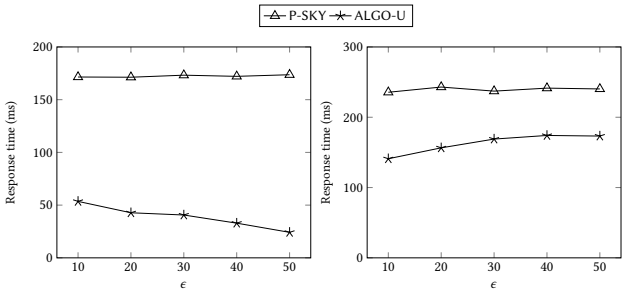
(a) Berlin **(b) New York**

Figure 9: Query response time varying # zones (r = 1,500 m).



(a) Berlin **(b) New York**

Figure 10: Query response time varying radius ($|Z|$ = 100).



(a) Berlin **(b) New York**

Figure 11: Query response time varying ϵ. (r = 1,500 m, $|Z|$ = 100)

for Berlin. Finally, Figure 11 shows the response time of P-SKY and ALGO-U for 100 preferred zones with a fixed radius of 1500 meters varying parameter ϵ. For both road networks and all values of ϵ, ALGO-U clearly outperforms P-SKY. Additionally, we observe that the performance of P-SKY is unaffected by the the parameter. With regard to the performance of ALGO-U, we observe that ϵ has a small effect on its performance.

Overall, in all settings tested ALGO-U outperforms P-SKY, while in particularly hard settings (many zones, large radius, large ϵ) the margin widens.

6 RELATED WORK

Computing the *shortest path* (SP) between two graph nodes is a fundamental problem that has attracted a lot of attention both by the research community and the industry. Traditionally, SP was addressed by Bellman-Ford and Dijkstra algorithms which traverse the graph building upon the notion of relaxation and subpath optimality. The ALT algorithms [19, 20, 33] perform a bidirectional A* search and exploit a lower bound of the distance between two nodes to direct the search. There exist also a number of materialization techniques [2, 24, 25] or encoding/labeling schemes [14, 16] that can be used to enhance the computation. In particular for routing on road networks, a plethora of pre-processing based methods have been proposed to answer SP queries in almost constant time, even for continental sized networks [1, 15, 18, 37, 43]. A recent survey and a experimental evaluation for road networks can be found in [5] and [41], respectively. The focus of our work is not to enhance the SP computation however, our methods employ aspects of the above literature such as the A* traversal using distance lower bounds and the contraction hierarchies of [18].

Our work is also related to variants of SP namely the *near-shortest path* and the *multi-criteria* SP. In the former, the goal is to identify the paths whose length is within a factor of $(1 + \epsilon)$ of the shortest-path length, for a user-specified $\epsilon \geq 0$. [9] was among the first works that addressed near-SP, adopting ideas from dynamic programming. Later, [10] proposed an number of extensions to enhance this solution, including a backward search from the target. In multi-criteria SP problems, the quality of a path is measured by multiple metrics, and the goal is to find all paths for which no better exists. Algorithms are categorized into three classes. The methods of the first class (e.g., [11]) apply a user preference function to reduce the original multi-criteria problem to a conventional SP problem. The second class contains the interactive methods (e.g., [21]) that interact with a decision maker to come up with the answer path. Finally, the third class includes label-setting and label-correcting methods (e.g., [22, 27, 35, 38, 39]) which compute a set of pareto-optimal paths, a.k.a. the path skyline. Our MPNSP problem is defined on the basis of near-SP while aspects of the above literature are considered or extended in our baseline P-SKY method of Section 3.1 and ALGO-U.

Conventional routing operates under the assumption that traveling time or distance is the most important optimization objective. In practice however, there exist a number of hard-to-formalize factors that affect people routing decisions. For this purpose, research efforts have focused on delivering personalized or context-aware routing, e.g., [12, 13, 17, 30, 32, 42] by extracting moving habits and patterns or popular paths from historical trajectory data. Recently, [31, 34] investigated attractiveness of the scenery in routing. However, the setup of the routing problems addressed by these works differs from the setting of the most preferred path we study. The notion of preferred zones that people are more acquainted to drive through is not defined or the objective is to maximize a collective attractiveness score instead of minimizing the time spend outside the preferred network.

Another line of related work involves safest path computation where the goal is to avoid obstacles or unsafe areas. This setup finds application among others in military applications, e.g., [6, 23, 29], or robotic path planning, e.g., [26, 28, 40]. However, the settings differ from ours as (i) the driver (or the moving object) is not allowed to move inside these unsafe zones, or (ii) the problems are defined in the Euclidean space, not on road networks. Hence, the proposed solutions are not applicable to our Problems 1 and 2.

To the best of our knowledge, computing the *Safest Path via Safe Zones* (SPSZ) and the *Safest Path via Preferred Zones* (SPPZ) proposed in [3, 4] are the most relevant problems to our work. As discussed in Sections 2 and 3, MPUP restates of SPSZ in our setting; Section 5 showed that our compressed network based solution always outperforms the HyperEdges algorithm proposed in [4]. The MPUP/SPSZ problem ignores the travel time inside the preferred/safe zones; on the other hand, SPPZ tries to minimize the total travel time captured by a linear combination of the time spent inside and outside these zones. This setting however is different from our MPNSP; our focus is to minimize only the time outside the preferred/safe zones but under a total time user-defined constraint. We believe that such a problem setting is more practical where a driver can only afford a specific increase of the total travel time.

7 CONCLUSIONS

In this work we introduced the problem of finding the most preferred path over road networks, modeling the scenario where a person prefers to travel as much as possible along a specific subset of the network, and study two instances. The first simply looks for the path that minimizes the time spent in the unpreferred network. The second looks for a path that minimizes the unpreferred time, but restricts the total traveling time to not exceed much that of the shortest path. We propose algorithms that are significantly more efficient than existing state-of-the-art methods, up to four orders of magnitude for the first instance, and up to six times for the second.

ACKNOWLEDGEMENTS

This work was partially supported by Innovation Fund Denmark as part of the Future Cropping project (J. nr. 5107-00002B), and by CRC-2016 as part of the EMMA project.

REFERENCES

[1] Ittai Abraham, Daniel Delling, Andrew V. Goldberg, and Renato Fonseca F. Werneck. 2011. A Hub-Based Labeling Algorithm for Shortest Paths in Road Networks. In *Experimental Algorithms - 10th International Symposium (SEA)*. 230–241.
[2] Rakesh Agrawal and H. V. Jagadish. 1989. Materialization and Incremental Update of Path Information. In *ICDE*. 374–383.
[3] Saad Aljubayrin, Jianzhong Qi, Christian S. Jensen, Rui Zhang, Zhen He, and Yuan Li. 2017. Finding lowest-cost paths in settings with safe and preferred zones. *VLDB J.* 26, 3 (2017), 373–397.
[4] Saad Aljubayrin, Jianzhong Qi, Christian S. Jensen, Rui Zhang, Zhen He, and Zeyi Wen. 2015. The safest path via safe zones. In *ICDE*. 531–542.
[5] Hannah Bast, Daniel Delling, Andrew V. Goldberg, Matthias Müller-Hannemann, Thomas Pajor, Peter Sanders, Dorothea Wagner, and Renato F. Werneck. 2016. Route Planning in Transportation Networks. In *Algorithm Engineering - Selected Results and Surveys*. Vol. 9220. 19–80.
[6] Scott A. Bortoff. 2000. Path planning for UAVs. In *American Control Conference*. 364–368.
[7] Stephan Börzsönyi, Donald Kossmann, and Konrad Stocker. 2001. The Skyline Operator. In *ICDE*. 421–430.
[8] Panagiotis Bouros. 2011. *Evaluating Queries over Route Collections*. Ph.D. Dissertation. NTUA, Greece.
[9] Thomas H. Byers and Michael S. Waterman. 1984. Determining all optimal and near-optimal solutions when solving shortest path problems by dynamic programming. *Operations Research* 32 (1984), 1381–1384.
[10] W. Matthew Carlyle and R. Kevin Wood. 2005. Near-shortest and K-shortest simple paths. *Networks* 46, 2 (2005), 98–109.
[11] Robert L. Carraway, Thomas L. Morin, and Herbert Moskowitz. 1990. Generalized dynamic programming for multicriteria optimization. *European Journal of Operational Research* 44, 1 (January 1990), 95–104.
[12] Vaida Ceikute and Christian S. Jensen. 2015. Vehicle Routing with User-Generated Trajectory Data. In *MDM*. 14–23.
[13] Zaiben Chen, Heng Tao Shen, and Xiaofang Zhou. 2011. Discovering popular routes from trajectories. In *ICDE*. 900–911.
[14] Jiefeng Cheng and Jeffrey Xu Yu. 2009. On-line exact shortest distance query processing. In *EDBT*. 481–492.
[15] Theodoros Chondrogiannis and Johann Gamper. 2016. ParDiSP: A Partition-Based Framework for Distance and Shortest Path Queries on Road Networks. In *MDM*. 242–251.
[16] Edith Cohen, Eran Halperin, Haim Kaplan, and Uri Zwick. 2002. Reachability and distance queries via 2-hop labels. In *SODA*. 937–946.
[17] Daniel Delling, Andrew V. Goldberg, Moisés Goldszmidt, John Krumm, Kunal Talwar, and Renato F. Werneck. 2015. Navigation made personal: inferring driving preferences from GPS traces. In *SIGSPATIAL GIS*. 31:1–31:9.
[18] Robert Geisberger, Peter Sanders, Dominik Schultes, and Daniel Delling. 2008. Contraction Hierarchies: Faster and Simpler Hierarchical Routing in Road Networks. In *WEA*. 319–333.
[19] Andrew V. Goldberg and Chris Harrelson. 2005. Computing the shortest path: A search meets graph theory. In *SODA*. 156–165.
[20] Andrew V. Goldberg, Haim Kaplan, and Renato F. Werneck. 2006. Reach for A*: Efficient point-to-point shortest path algorithms. In *Proceedings of the 8th WS on Algorithm Engineering and Experiments (ALENEX). SIAM, Philadelphia*. 129–143.
[21] Janusz Granat and Francesca Guerriero. 2003. The interactive analysis of the multicriteria shortest path problem by the reference point method. *European Journal of Operational Research* 151, 1 (November 2003), 103–118.
[22] F. Guerriero and R. Musmanno. 2001. Label correcting methods to solve multicriteria shortest path problems. *Journal of Optimization Theory and Applications* 111, 3 (2001), 589–613.
[23] R. V. Helgason, J. L. Kennington, and K. H. Lewis. 1997. *Shortest path algorithms on grid graphs with applications to strike planning*. Technical Report. DTIC Document.
[24] Ning Jing, Yun-Wu Huang, and Elke A. Rundensteiner. 1998. Hierarchical Encoded Path Views for Path Query Processing: An Optimal Model and Its Performance Evaluation. *TKDE* 10, 3 (1998), 409–432.
[25] Sungwon Jung and Sakti Pramanik. 2002. An Efficient Path Computation Model for Hierarchically Structured Topographical Road Maps. *TKDE* 14, 5 (2002), 1029–1046.
[26] Rahul Kala, Anupam Shukla, and Ritu Tiwari. 2010. Fusion of probabilistic A* algorithm and fuzzy inference system for robotic path planning. *Artif. Intell. Rev.* 33, 4 (2010), 307–327.
[27] Hans-Peter Kriegel, Matthias Renz, and Matthias Schubert. 2010. Route skyline queries: A multi-preference path planning approach. In *ICDE*. 261–272.
[28] Alain Lambert and Dominique Gruyer. 2003. Safe path planning in an uncertain-configuration space. In *ICRA*. 4185–4190.
[29] Louise Leenen, Alexander Terlunen, and Herman Le Roux. 2012. A constraint programming solution for the military unit path finding problem. *Mobile Intelligent Autonomous Systems* 9, 1 (2012), 225–240.
[30] Julia Letchner, John Krumm, and Eric Horvitz. 2006. Trip Router with Individualized Preferences (TRIP): Incorporating Personalization into Route Planning. In *Proceedings, The Twenty-First National Conference on Artificial Intelligence and the Eighteenth Innovative Applications of Artificial Intelligence*. 1795–1800.
[31] Ying Lu and Cyrus Shahabi. 2015. An arc orienteering algorithm to find the most scenic path on a large-scale road network. In *SIGSPATIAL GIS*. 46:1–46:10.
[32] Kayur Patel, Mike Y. Chen, Ian E. Smith, and James A. Landay. 2006. Personalizing routes. In *Proceedings of the 19th Annual ACM Symposium on User Interface Software and Technology*. 187–190.
[33] I Pohl. 1971. Bi-directional Search. *Machine Intelligence* 6 (1971), 127–140.
[34] Daniele Quercia, Rossano Schifanella, and Luca Maria Aiello. 2014. The shortest path to happiness: recommending beautiful, quiet, and happy routes in the city. In *HT*. 116–125.
[35] Michael Shekelyan, Gregor Jossé, and Matthias Schubert. 2015. Linear path skylines in multicriteria networks. In *ICDE*. 459–470.
[36] Michael Shekelyan, Gregor Jossé, and Matthias Schubert. 2015. ParetoPrep: Efficient Lower Bounds for Path Skylines and Fast Path Computation. In *SSTD*. 40–58.
[37] Christian Sommer. 2014. Shortest-path queries in static networks. *ACM Comput. Surv.* 46, 4 (2014), 45:1–45:31.
[38] Yuan Tian, Ken C. K. Lee, and Wang-Chien Lee. 2009. Finding skyline paths in road networks. In *SIGSPATIAL GIS*. 444–447.
[39] Chi Tung Tung and Kim Lin Chew. 1992. A multicriteria Pareto-optimal path algorithm. *European Journal of Operational Research* 62, 2 (1992), 203 – 209.
[40] Jur P. van den Berg and Mark H. Overmars. 2006. Planning the Shortest Safe Path Amidst Unpredictably Moving Obstacles. In *WAFR*. 103–118.
[41] Lingkun Wu, Xiaokui Xiao, Dingxiong Deng, Gao Cong, Andy Diwen Zhu, and Shuigeng Zhou. 2012. Shortest Path and Distance Queries on Road Networks: An Experimental Evaluation. *PVLDB* 5, 5 (2012), 406–417.
[42] Bin Yang, Chenjuan Guo, Yu Ma, and Christian S. Jensen. 2015. Toward personalized, context-aware routing. *VLDB J.* 24, 2 (2015), 297–318.
[43] Andy Diwen Zhu, Hui Ma, Xiaokui Xiao, Siqiang Luo, Youze Tang, and Shuigeng Zhou. 2013. Shortest path and distance queries on road networks: towards bridging theory and practice. In *SIGMOD*. 857–868.