Noname manuscript No.
(will be inserted by the editor)

Finding Lowest-Cost Paths in Settings with Safe and Preferred Zones

Saad Aljubayrin · Jianzhong Qi · Christian S. Jensen · Rui Zhang · Zhen He · Yuan Li

the date of receipt and acceptance should be inserted later

Abstract We define and study Euclidean and spatial network variants of a new path finding problem: given a set of safe or preferred zones with zero or low cost, find paths that minimize the cost of travel from an origin to a destination. In this problem, the entire space is passable, with preference given to safe or preferred zones. Existing algorithms for problems that involve unsafe regions to be avoided strictly are not effective for this new problem.

To solve the Euclidean variant, we devise a transformation of the continuous data space with safe zones into a discrete graph upon which shortest path algorithms apply. A naive transformation yields a large graph that is expensive to search. In contrast, our transformation exploits properties of hyperbolas in Euclidean space to safely eliminate graph edges, thus improving performance without affecting correctness. To solve the spatial network variant, we propose a different graph-to-graph transformation that identifies critical points that serve the same purpose as do the hyperbolas, thus also avoiding the extraneous edges. Having solved the problem for safe zones with zero costs, we extend the transformations to the weighted version of the problem, where travel in preferred zones has non-zero costs. Experiments on both real and synthetic data show that our approaches outperform baseline approaches by more than an order of magnitude in graph construction time, storage space, and query response time.

S. Aljubayrin, J. Qi, R. Zhang, and Y. Li
University of Melbourne, Australia
E-mail: aljubayrin@su.edu.sa, jianzhong.qi@unimelb.edu.au, rui.zhang@unimelb.edu.au, yuanl4@student.unimelb.edu.au

C. S. Jensen
Aalborg University, Denmark
E-mail: csj@cs.aau.dk

Z. He
Latrobe University, Australia
E-mail: z.he@latrobe.edu.au

Keywords Path Finding · Safest Path · Safe Zones · Preferred Zones · Hyperbola

1 Introduction

1.1 Motivation and Contribution Overview

Shortest path computation has been studied extensively. However, in some scenarios, the shortest path may not be the desired one. In hazardous environments, it can be life critical to minimize the distance traveled in unsafe regions. For example, a person who drives a long distance through the desert may try to travel via villages ("*safe zones*") because a breakdown in an unpopulated region can be life threatening. The traditional shortest path from an origin to a destination is likely to differ substantially from a "shortest" path that is based on a preference to travel as little as possible outside populated regions. In a more familiar scenario, a tourist who plans to walk to a given destination may prefer a path that visits interesting streets and blocks, e.g., with interesting houses, galleries, or other sights, as much as possible. Here, traveling in interesting regions ("safe zones") is merely *preferred*, and the tourist is unlikely to choose the "safest path" if it comes at the cost of a very long walk.

In the first of the above two scenarios, we may assign zero cost to travel in safe zones, while in the second, we may assign a reduced weight $\alpha \in [0, 1)$ to travel in safe zones in order to capture a user's degree of preference for "safety" versus distance. To distinguish these preferred regions from safe zones, we call them *preferred zones*. In a large city, the number of buildings and blocks is very large (e.g., there are over 1 million buildings in New York City[1]). Many of these buildings/blocks are regions of interests (preferred zones), and the number of preferred zones is in the

[1] https://www.mapbox.com/blog/nyc-buildings-openstreetmap/

hundreds of thousands. This problem can also help cyclists find routes with the least distance outside of bicycle lanes (preferred zones). According to OpenStreetMap[2], Amsterdam has over 645,000 road segments, many of which have bicycle lanes. As we will see, a naive algorithm for this problem has a quadratic time complexity with respect to the number of safe or preferred zones. With quadratic complexity, path computation in scenarios such as the above is very time-consuming, and more efficient algorithms are needed.

Motivated by the scenarios just covered, we formulate two new problems called the *safest path via safe zones* (SPSZ) and the *safest path via preferred zones* (SPPZ), which are to find a path that minimizes the distance traveled outside a set of discrete safe zones and a set of discrete preferred zones, respectively. We study the problems in both Euclidean and spatial network settings. Existing studies on finding a safe path aim to strictly avoid a set of unsafe regions [4, 19, 20, 23]. Our problem setting is different in that the entire space is unsafe except for a set of *safe or preferred zones*, and unsafe regions are still passable. Therefore, algorithms that solve these problems do not apply.

1.2 SPSZ Solution Overview

To solve the SPSZ problem in the Euclidean setting, we first transform the data space with safe zones into a graph where the safe zones are represented by vertices, and paths between them are represented by edges. As the locations and sizes of the safe zones are relatively static (e.g., it is unlikely to have new bicycle lanes constructed every day), the graph can be precomputed. When an SPSZ query is issued, the query origin and destination are added to the graph as vertices. Then, any shortest path algorithm may be applied to find the safest path.

Note that even though the graph is precomputed, it is still critical to have an efficient algorithm for the precomputation. A naive transformation of the data space into a graph introduces an edge between every pair of vertices with a weight that equals the unsafe distance between the two vertices. This yields $N(N-1)/2$ edges, where N is the number of safe zones. As will be shown in the experimental study, most such edges are not competitive and will never be used in any safest path. One way to eliminate superfluous edges is to apply the Floyd-Warshall algorithm [10] to find the shortest paths between all pairs of vertices. However, the cost of doing so is very high ($O(N^3)$). In our experiments, this naive graph precomputation algorithm takes more than a day to process $10,000$ safe zones. Even with an improved implementation, the algorithm takes more than a week to process $160,000$ safe zones. It is too slow to be used in practice.

[2] http://www.openstreetmap.org/

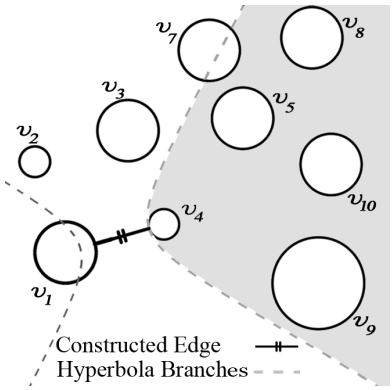


Fig. 1: Pruning with hyperbola

We observe that the regions containing the vertices with superfluous edges can be described elegantly by hyperbolas, and we propose an algorithm that utilizes the properties of hyperbolas to avoid such edges, thus obtaining a much more sparsely connected graph. Figure 1 illustrates the main idea. Assume that we are constructing edges between v_1 and the other vertices in the graph. The distances between the circles denote the unsafe distances between the vertices. First, we construct an edge between v_1 and its nearest vertex v_4. Then we compute a hyperbola based on the centers of the two vertices and the unsafe distance between them. We use the hyperbola branch closer to v_4 to divide the data space into two parts. As will be shown in Section 4, any vertex located on the v_4 side of the hyperbola branch (in the shaded region) has a shortest path to v_1 that goes through v_4. Therefore, we disregard vertices representing safe zones in the shaded region when creating edges between v_1 and other vertices.

When solving the SPSZ problem in the spatial network setting, straightforwardly introducing Euclidean space hyperbola is not possible because the hyperbola definition does not apply to network distance. Instead, we identify a set of critical points that bound the parts of the spatial network where traveling directly to a safe zone v has shorter unsafe distance than traveling to v through an intermediate safe zone u. We call these critical points *hyperbola points* in analogy with the solution for the Euclidean setting. To identify the hyperbola points of v, we traverse the spatial network from v in a breath-first fashion and label every network vertex d by the safe zones that the path reaching d has passed. The traversal terminates when v is surrounded by network vertices that have been labeled by other safe zones, and the surrounding vertices are the hyperbola points.

1.3 SPPZ Solution Overview

We generalize the safest path problem to the problem where travel in a safe zone has a possibly non-zero fraction of

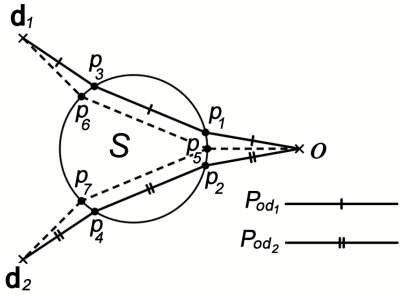


Fig. 2: Safest paths through a preferred zone

the cost of travel outside safe zones. With this generalization, the safe zones become preferred zones, and we refer to the problem as the safest path via preferred zones problem, i.e., SPPZ. In the spatial network setting of the SPPZ problem, our hyperbola points based method applies straightforwardly. We just need to add the costs of travel inside the preferred zones to the network path lengths when computing the hyperbola points.

In the Euclidean setting, however, the hyperbola based approach is not directly applicable. When the cost of travel inside a preferred zone is considered, the optimal path to travel from a point o to a preferred zone s no longer has to be the shortest path from o to s, because the shortest path may require a longer travel distance inside the preferred zone. The optimal path now depends on the weight of the cost of travel inside preferred zones, and the location of the next destination to reach from s.

Figure 2 illustrates the problem. Assume that travel inside the preferred zone has a non-zero weighted cost ($0 <$ weight < 1). A straightforward shortest path based algorithm will return $\overline{o, p_5, p_6, d_1}$ as the lowest-cost path from o to d_1. This is because $\overline{o, p_5}$ and $\overline{d_1, p_6}$ are the shortest paths from o and d_1 to the preferred zone. However, $\overline{p_5, p_6}$ is not the lowest-cost path through the preferred zone. Path $\overline{p_1, p_3}$ has lower cost and may be used to form a path P_{od_1} that has lower overall cost, depending on the weight of the cost of $\overline{p_1, p_3}$. Further, when given a different destination point d_2, another path P_{od_2} may be the lowest-cost path. For these reasons, precomputing a graph without knowing the origin and destination cannot produce the safest paths correctly. We observe that, even though the shortest path from o to s may no longer be optimal, it can provide a lower bound in distance computations. We redefine the hyperbolas based on this lower bound, which are then used for precomputing a graph. This graph may contain redundant edges, but will not miss any edge for safest path computation. When an SPPZ query is issued, its origin and destination are added to this graph to compute a shortest path.

Since the edges in the precomputed graph are simply shortest paths between the preferred zones, which may not be optimal, a shortest path between the query origin and destination in this graph may not contain the optimal edges to travel through the preferred zones on the path. We improve this shortest path by refining the edges on the path. This is done by computing the optimal entry and exit points between every pair of adjacent preferred zones (and the origin and destination) and introducing an edge to connect them. Since the boundary of a preferred zone has an infinite number of points, each pair of preferred zones can have an infinite number of pairs of entry and exit points. As there is no closed-form equation to compute the optimal edge to reach a preferred zone, we propose two heuristics to compute approximately optimal edges. Through experiments, we show that the resulting algorithms can compute the (approximately) safest paths both effectively and efficiently. Note that this edge refinement is unnecessary for the SPSZ problem: in SPSZ the optimal edge to connect two safe zones is simply the shortest path between them, because travel inside safe zones has no cost.

1.4 Contributions

In summary, we make the following contributions:

1. We propose a new safe zone based path finding problem in Euclidean and spatial network settings.
2. To solve the problem, we model the data space as a total graph from which unnecessary edges are eliminated during graph construction. We propose a novel edge pruning algorithm based on hyperbolas to solve the problem in Euclidean settings. We also report on a first investigation on extending the idea to spatial networks, specifically proposing an edge pruning algorithm based on hyperbola points in spatial network settings.
3. We generalize the problem to a preferred zone path finding problem. Here, a weighted distance is used for travel in the preferred zones to capture a user's preference for "safety" versus distance.
4. We extend the hyperbolas and hyperbola points based algorithm to solve the generalized problem. In particular, the generalized problem in the Euclidean setting has no closed-form equation to compute the optimal solution. We therefore propose two heuristics to compute approximate solutions for the generalized problem.
5. We perform extensive experiments to evaluate the efficiency of the proposed algorithms, and the results are summarized as follows:
   (a) In the Euclidean setting and for routing through safe zones, our graph construction algorithm is more than an order of magnitude faster than an improved naive algorithm and two orders of magnitude faster than

a naive algorithm. At query time, adding edges to connect the query origin and destination to the graph using our hyperbola based algorithm is up to an order of magnitude faster than that using the naive or improved naive algorithms.

(b) In the Euclidean space setting and for routing through preferred zones, our algorithms can compute efficiently paths that are constantly shorter (and hence safer) than those computed by a straightforward shortest path algorithm.

(c) In the spatial network setting, our edge pruning algorithm successfully reduces the number of edges in the graph constructed.

The paper extends an earlier paper [2]. There, we presented the safest path finding problem through safe zones and algorithms to solve the problem. Here, we generalize the problem to find the safest path through preferred zones, where a weighted distance is used for travel in preferred zones to capture a user's preference for "safety" versus distance. This generalized problem is non-trivial, especially in the Euclidean setting. The optimal path to reach a preferred zone is not simply the shortest path. Instead, the optimal path depends on the origin as well as the next destination to reach after passing a preferred zone. When the next destination is also a preferred zone, an infinite number of points can be used in the path computation. No closed-form equation exists for computing the optimal path under such circumstances. Even computing an approximately optimal path is challenging. We overcome this challenge by introducing two heuristics to efficiently compute near optimal edges for travel between the preferred zones (Section 5). We perform additional experiments (Section 7.2), which confirm the effectiveness and efficiency of the proposed algorithms.

1.5 Organization

The reminder of the paper is organized as follows. Related work is discussed in Section 2. Section 3 presents the preliminaries and a solution framework. Sections 4 and 5 detail the proposed algorithms for the SPSZ and the SPPZ problems in the Euclidean setting, respectively. Section 6 details the proposed algorithms for the two problems in the spatial network setting. Section 7 presents the experimental results, and Section 8 concludes the paper.

2 Related Work

Shortest path finding (e.g., [4,10,13,14]) has been a popular research topic in the past few decades. Most studies address route planing in graphs (e.g., [10,13,14]). Other studies consider route planing in open space, (e.g., [4]). When planing a route in a graph, the route must follow predefined edges, while in an open space there is no such restriction. Bast et al. [3] present a survey on route planning. However, we are unaware of any attempt to investigate the problem of finding the safest path via safe zones in an unsafe space. Only a few existing studies consider the safety aspect. Hallam et al. [13] propose to find *multicriterion shortest paths*, which aim to help submarines find shortest paths based on two factors: the travel time and the risk of being detected by enemy sensors. They assume the submarines to move between the vertices (safe points) of a two or three dimensional grid. The setting is different from ours in that it assumes safe points, where a submarine can stop at, rather than safe zones, in that the size of a safe point (i.e., the area it spans) is not considered, and in that navigating through safe points does not reduce the unsafe distance. In our problem setting, traveling through safe zones contributes a weighted cost to the overall traveling cost, and the aim is to travel as little as possible outside safe zones to reduce the overall traveling cost. Therefore, the solution of Hallam et al. is not applicable to our problem.

Other studies on safest path problems assume that the majority of the data space is safe and has a number of unsafe zones in it, which is opposite to our assumption. In robotic path planning [4, 16, 18, 19], it is an objective to find a path for a robot that is optimal in the sense that it avoids collisions with obstacles. Existing studies have considered path planning in both graphs (e.g., [16, 18, 19]) and open space (e.g., [4]). Some studies assume that the obstacles are moving (e.g., [4]) while others consider uncertainty in the movement of the robots (e.g., [18, 19]). The obstacles in these studies form the unsafe regions, while the rest of the space is safe. The A* algorithm is commonly used in these studies. In *military unit path finding* (MUPFP) [20, 24], the problem is to find an optimal path for a military unit to move from its current location to another location. These studies also try to avoid certain regions (e.g., obstacles and regions controlled by an enemy). For example, Leenen et al. [20] design an objective cost function to find the shortest path for a military unit based on a set of safety constraints. Studies of path finding for *unmanned aerial vehicles* (UAVs) [5, 23] aim to find a path for a UAV to move to a destination safely. Here, safety implies not flying in enemy radar detection zones that form the unsafe regions while the rest of the space is considered safe. The *safest paths for cruise missiles* problem [14] is addressed using a grid based approach. They assign a certain safety probability to each grid edge based on its distance from a threat region. Other works (e.g., [5]) use Voronoi diagrams to represent unsafe polygons and find the safest path when traveling along diagram edges. Most of the studies above assume the unsafe zones are strictly not passable, while unsafe regions are passable in our setting. Therefore, the existing solutions are not applicable even when the space definition of safe and unsafe areas is flipped around.

Table 1: Notation

Notation	Explanation
$\mathcal{S}$	The set of closed safe zones
o	The origin of an SPSZ query
d	The destination of an SPSZ query
P	A path from o to d
$\overline{p_i, p_{i+1}}$	A line segment in a path
$S(\overline{p_i, p_{i+1}})$	Sub-segments within the safe zones
$U(\overline{p_i, p_{i+1}})$	Sub-segments outside the safe zones
$\|U(P)\|$	The unsafe distance of P
s	A safe zone
α	Safe zone traveling cost weight
$dist_R()$	Spatial network distance

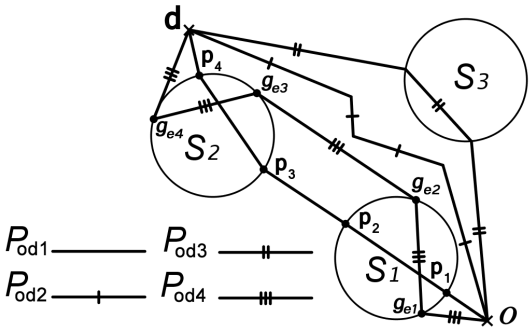


Fig. 3: Paths and safe zones

Lu and Shahabi [22] study the problem of finding the *most scenic path*. This problem and our SPPZ problem in the spatial network setting both consider a certain portion of the network to be more preferable. However, the problem settings differ. In the former problem, each network edge has a cost and an attractiveness value, where a more attractive edge does not necessarily have a low cost. In our problem setting, a preferred zone has a lower cost. Thus, the solution provided by Lu and Shahabi does not apply to our problem.

There are extensive studies on nearest neighbor or range queries [8, 9, 15, 17, 21, 25, 29], which find the objects in a database that have the smallest distances to a given query point or range. Two recent studies [27, 28] predict a traveller's destination based on her partial route. These studies do not consider the "safety" of a path. Most of these studies (e.g., [9, 15, 17, 25]) use Euclidean distance as the distance metric, while others (e.g., [8, 21]) use network distance. In the SPSZ problem, after transformation to a graph, we use graph distance, and we consider the safety of a path. The nearest neighbor or range query algorithms are not applicable to our problem.

3 Preliminaries

We proceed to formalize the SPSZ and the generalized SPPZ problems. We also describe the properties of hyperbolas in this section. Table 1 summarizes the symbols used.

3.1 The SPSZ Problem

We assume a set of safe zones denoted by $\mathcal{S}$, an origin point o, and a destination point d. In particular, we consider round and polygon safe zones. Let $P = \langle o, p_1, p_2, ..., p_n, d\rangle$ be a *path* between o and d, where p_i denotes a point on the boundary of a safe zone. Then $\overline{o, p_1}$, $\overline{p_1, p_2}$, ..., $\overline{p_n, d}$ each denotes a segment that is either entirely within a safe zone or not within any safe zone at all. For example, in path P_{od1} of Fig. 3, $\overline{p_1, p_2}$ and $\overline{p_3, p_4}$ are within safe zones s_1 and s_2, respectively, while remaining segments are not within any safe zone. We use $|P|$ to denote the length of P which is computed as the sum of the lengths of the segments, i.e., $|P| = |\overline{o, p_1}| + |\overline{p_1, p_2}| + ... + |\overline{p_n, d}|$. Let $S(P)$ and $U(P)$ be the two sets of segments of P that are inside and outside of safe zones, respectively. We call $|U(P)|$ the *unsafe distance* of P and $|S(P)|$ the *safe distance* of P. In Fig. 3, $|U(P_{od1})| = |\overline{o, p_1}| + |\overline{p_2, p_3}| + |\overline{p_4, d}|$, and $|S(P_{od1})| = |\overline{p_1, p_2}| + |\overline{p_3, p_4}|$

The problem of SPSZ is defined as follows:

Definition 1 (Safest Path via Safe Zones Query) *Given an unsafe data space, a set of safe zones $\mathcal{S}$ in it, an origin point o, and a destination point d, the safest path via safe zones (SPSZ) query finds a path P from o to d, such that for any other path P' from o to d, the unsafe distance of P is less than or equal to that of P', i.e., $\forall P', |U(P)| \leq |U(P)'|$.*

A straightforward solution is to compute and compare the unsafe distance of all the possible paths and then return the path with the shortest unsafe distance. In Fig. 3, the unsafe distance of the path P_{od1} is smaller than those of the other paths. However, in the Euclidean setting, there is potentially an infinite number of paths between any two points. Therefore, the straightforward solution may not be feasible. To overcome this problem, we redefine the problem to confine the number of candidate safest paths to a limited number, based on which we propose our problem solution.

Definition 2 (SPSZ Query (redefinition)) *Given an unsafe data space, a set of safe zones $\mathcal{S}$ in it, an origin point o, and a destination point d, the SPSZ query finds a path formed by a sequence of safe zones $s_1, s_2, ..., s_m$, such that $|o, s_1|^{\perp} + |s_1, s_2|^{\perp} + ... + |s_m, d|^{\perp}$ is minimized, where $|\cdot, \cdot|^{\perp}$ denotes the shortest unsafe distance between two objects (either safe zones or query points o and d). If o (or d) is in s_1 (s_m) then we let $|o, s_1|^{\perp}$ ($|s_m, d|^{\perp}$) be 0.*

We explain the intuition of the problem redefinition below after introducing some notation. Given a path P, let

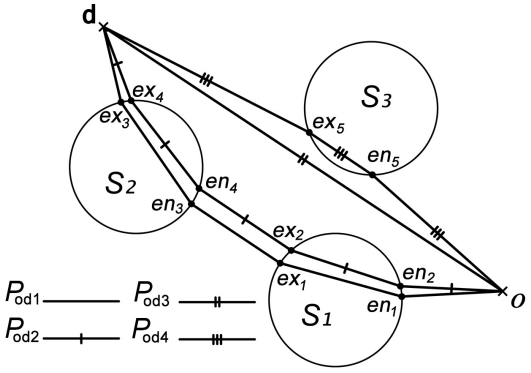


Fig. 4: Paths via preferred zones

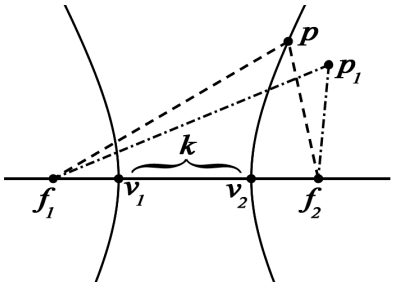


Fig. 5: A hyperbola

$\langle o, s_1, ..., s_m, d\rangle$ be the sequence of the origin point (o), safe zones (s_i), and the destination point (d) passed through by P. For example, in Fig. 3, both P_{od1} and P_{od4} pass through $\langle o, s_1, s_2, d\rangle$, while P_{od3} passes through $\langle o, s_3, d\rangle$. A segment $g \in U(P)$ connects a pair of adjacent objects $\langle ob_1, ob_2\rangle$ in the sequence of P, where ob_i is an origin (destination) point or a safe zone. For example, $\overline{g_{e2}, g_{e3}} \in U(P_{od4})$ connects s_1 and s_2. While there may be infinite segments that connect $\langle ob_1, ob_2\rangle$, we can identify the shortest *line* segment among them, e.g., $\overline{p_2, p_3}$ for $\langle s_1, s_2\rangle$. Given any sequence of origin/destination points and safe zones, we just need to consider the path formed by the shortest line segments that connect them. As a result, the problem of finding the safest path between o and d becomes one of finding a sequence of safe zones such that the line segments connecting them have the shortest total length. For example, in Fig. 3, sequence $\langle o, s_1, s_2, d\rangle$ has a path P_{od1} formed by line segments $\overline{o, p_1}$, $\overline{p_2, p_3}$, and $\overline{p_4, d}$, which have the shortest total unsafe distance. This is the safest path between o and d.

In the spatial network setting, the problem is defined analogously to the Euclidean definition. Here the number of possible paths between safe zones is not infinite, but it may be very large. Therefore, an analysis similar to the above applies where we just need to replace "the shortest line segment" with "the shortest path in the spatial network."

3.2 The SPPZ Problem

In the preferred zone scenario, the cost of travel inside preferred zones is added to the overall travel cost. This cost is weighted by a factor α ($0 \leq \alpha < 1$) representing the degree of preference of travel inside preferred zones. A smaller value of α means that travel inside preferred zones is more preferred. A safe zone is then a special case of a preferred zone where $\alpha = 0$. Now the overall cost of a path P becomes $|U(P)| + \alpha|S(P)|$. The problem of finding the safest path via preferred zones, SPPZ, is to find the path that minimizes this cost. We formalize it as the SPPZ query.

Definition 3 (Safest Path via Preferred Zones Query) *Given a data space with a set of preferred zones $\mathcal{S}$ in it, a weight α, an origin point o, and a destination point d, the safest path via preferred zones (SPPZ) query finds a path P from o to d, such that for any other path P' from o to d, the overall cost of P is less than or equal to that of P', i.e.,* $\forall P', |U(P)| + \alpha|S(P)| \leq |U(P')| + \alpha|S(P')|$.

Paths in the SPPZ problem do *not* have to be formed by shortest path between preferred zones. This is because taking the shortest paths between preferred zones may require taking longer paths inside preferred zones, thus yielding higher overall cost. Figure 4 is an example. There are four paths between o and d. When $\alpha = 0$, the problem falls back to SPSZ. The optimal path P_{od1} is formed by the shortest paths between o and d and the preferred zones. When $\alpha \neq 0$, the lowest-cost paths are generally different from the shortest paths, e.g., path P_{od2} may be a better path. The path has longer travel distances outside the preferred zones, but also shorter distances inside the two preferred zones. If α is close to 1, travel inside and outside preferred zones has little difference. The optimal path would become close to the direct path P_{od3} between o and d (e.g., P_{od4}).

3.3 Hyperbolas

We use *hyperbolas* to construct algorithms to solve the SPSZ and SPPZ problems. As illustrated in Fig. 5, a hyperbola is a smooth curve with two branches, such that every point p on the curve has the property that the difference between the distance from p to a point f_1 and the distance from p to a point f_2 is a positive constant k [12]:

$$|\overline{f_1, p}| - |\overline{f_2, p}| = k$$

Points f_1 and f_2 are the *focal points* of the hyperbola. A point p_1 to the *right* of the right hyperbola branch satisfies:

$$|\overline{f_1, p_1}| - |\overline{f_2, p_1}| > k$$

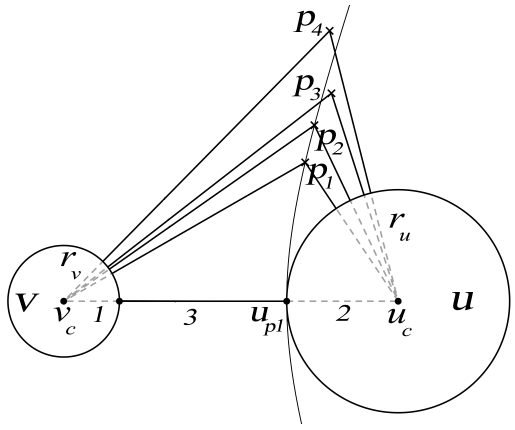


Fig. 6: Hyperbola for round safe zones

We exploit this property to prune paths between safe zones.

Let v_c and u_c be the centers of two round safe zones v and u, respectively. We use these two centers as the focal points to construct a hyperbola as shown in Fig. 6. The right hyperbola branch divides the space into two sub-spaces. A point (e.g., p_3) on the u_c side of the sub-space satisfies:

$$|\overline{v_c, p}| - |\overline{u_c, p}| > k \Rightarrow |\overline{v_c, p}| > |\overline{u_c, p}| + k$$

By letting k be $|\overline{v_c, u_{p1}}| - r_u$, where u_{p1} denotes the closest point to v on u and r_u denotes the radius of u, we obtain:

$$|\overline{v_c, p}| > |\overline{u_c, p}| + |\overline{v_c, u_{p1}}| - r_u$$

Since $|\overline{v_c, u_{p1}}| = |v, u|^{\perp} + r_v$, where $|v, u|^{\perp}$ denotes the shortest distance between v and u and r_v denotes the radius of v, we have:

$$|\overline{v_c, p}| > |\overline{u_c, p}| + |v, u|^{\perp} + r_v - r_u$$

$$\Rightarrow (|\overline{v_c, p}| - r_v) > (|\overline{u_c, p}| - r_u) + |v, u|^{\perp}$$

Here, $|\overline{v_c, p}| - r_v$ and $|\overline{u_c, p}| - r_u$ are the minimum unsafe distances from p to v and from p to u, respectively, while $|v, u|^{\perp}$ is the minimum unsafe distance between v and u. As a result, we know that the unsafe distance from p to v exceeds from p to u and then to v. It is safer to travel to u first, and there is no need to consider any direct path between v and any vertex in the right sub-space of the hyperbola branch of u. We describe the algorithms to solve the SPSZ and SPPZ problems based on this idea next.

4 SPSZ Query in the Euclidean Setting

First we describe the solution framework of the SPSZ problem. Next, we discuss two straightforward solutions. Then we present our algorithm named the HyperEdges algorithm. We give details on how the hyperbolas work for round and polygon safe zones. Finally we discuss how to deal with overlapping safe zones.

4.1 SPSZ Solution Framework

In the Euclidean setting, we transform the SPSZ query to a shortest path problem as follows. We let the set of safe zones $\mathcal{S}$ plus the origin and destination points be the set of vertices $\mathcal{V}$, i.e., $\mathcal{V} = \mathcal{S} \cup \{o, d\}$. For every pair of vertices in $\mathcal{V}$, we add an edge to the set of edges $\mathcal{E}$, and we associate it with a weight denoting the shortest distance between the two vertices. After graph construction, finding the safest path between o and d is equivalent to finding the shortest path between o and d on a graph $\mathcal{G} = \langle \mathcal{V}, \mathcal{E} \rangle$, which can be done by a standard graph shortest path algorithm such as Dijkstra's algorithm. Different queries may have different origin and destination points, but they share the same sub-graph $G^{\circ} = \langle \mathcal{V}^{\circ}, \mathcal{E}^{\circ} \rangle$, where $\mathcal{V}^{\circ} = \mathcal{S}$ and $\mathcal{E}^{\circ}$ contains the edges connecting the vertices in $\mathcal{S}$. Therefore, the sub-graph G° can be precomputed. When a query is issued, we add edges to connect the origin and destination points to G°.

We achieve a two-stage solution framework for the SPSZ query as follows:

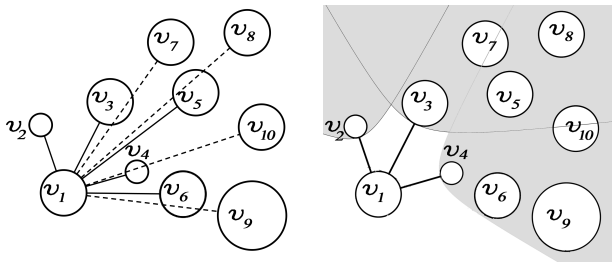
- **Stage 1:** Precompute the graph G° on $\mathcal{S}$.
- **Stage 2:** When an SPSZ query with an origin point o and a destination point d is issued:
  (a) Add o and d to G° to form a graph G.
  (b) Perform a shortest path search on G with o as the origin and d as the destination.

In this study, we aim to obtain a graph that contains as few edges as possible while not missing any edge that may appear in a shortest path between two vertices. This is because the number of edges plays a vital role in the efficiency of graph construction and shortest path finding. The shortest path algorithm used is orthogonal to the work in this paper. We have used Dijkstra's algorithm for simplicity, although any other graph shortest path algorithms such as Contraction Hierarchies [11] or Hub Labeling [1] may be used.

In the spatial network setting, we apply the same two-stage framework but replace the Euclidean distance by the network edge weights. In the following sections, we present our graph construction algorithms for both the Euclidean and the spatial network settings.

4.2 Baseline Algorithms

Naive algorithm: A naive algorithm for graph construction works as follows. First, we add an edge to every possible pair of vertices (safe zones). Second, we filter the edges by the Floyd-Warshall algorithm [10] to compute the shortest path between every pair of vertices. Although we used the Floyd-Warshall algorithm, any other all shortest path algorithm, e.g., PHAST [7], can be used. Only the edges appearing in at least one shortest path are kept. This algorithm produces the minimum graph in the precomputation stage.



(a) Improved Naive algorithm (b) HyperEdges algorithm

Fig. 7: Constructing edges for v_1

However, it is expensive in both time ($O(|\mathcal{V}|^3)$) and space ($O(|\mathcal{V}|^2)$).

Improved Naive algorithm: The second baseline algorithm improves on the naive algorithm. Instead of adding an edge between every pair of safe zones, we only add an edge if the line segment between two safe zones does not intersect with any other safe zone. Omission of such edges works because the route between the two safe zones that goes via an intersecting safe zone is safer than the direct route. As indicated in Fig. 7a, for vertex v_1, only the edges to v_2, v_3, v_4, and v_6 are added to the graph since edges to any other vertex intersects with a safe zone. We also filter the superfluous edges using Dijkstra's algorithm.

Although the number of edges created by this algorithm is significantly smaller than that of the naive algorithm, it is still expensive to check the overlap between line segments and safe zones to avoid unnecessary edges.

4.3 The HyperEdges Algorithm

We propose an efficient algorithm to filter out the superfluous edges not used in any safest path. Our solution is based on the observation that the regions containing the vertices with superfluous edges can be elegantly described by hyperbolas, and we propose an algorithm that utilizes the properties of hyperbolas to avoid such edges, thus obtaining a much more sparsely connected graph. We call our algorithm the **HyperEdges** algorithm.

The main idea of the HyperEdges algorithm is that, for each vertex v, we add edges connecting v with other vertices progressively, during which we use the connected vertices to prune part of the data space from being considered for edge creation based on the properties of hyperbolas. Next, we first present our graph construction and query processing algorithms and then explain in detail how we compute the parameters of the hyperbolas used and prove its correctness.

Graph precomputation: We start with a graph containing just vertices (the safe zones) and add edges progressively. Consider again the example in Fig. 1. The circles represent safe zones that are the vertices in the graph. To add the edges for a vertex v_1, we first create an edge between v_1 and its nearest vertex, which is v_4.

Algorithm 1: HyperEdges - SPSZ in Euclidean setting

```
Input: A set of safe zones S indexed in a quad-tree Q
Output: A precomputed graph G°
V° ← S;
// create edges
for v ∈ V° do
    u ← next_nearest_neighbor(v, Q);
    while u ≠ null do
        v.add_edge(⟨v, u⟩);
        h ← compute_hyperbola(v,u);
        // use the created hyperbola h to
           prune nodes in quad-tree Q
        prune_nearest_neighbor(h, Q);
        // perform best-first search that
           only considers zones outside the
           computed hyperbolas
        u ← next_nearest_neighbor(v, Q);
// remove superfluous edges
for v ∈ V° do
    for e ∈ v.edges do
        P ← Dijkstra(e.v1, e.v2);
        if e ∉ P then
            v.remove_edge(e);
```

Then we compute a hyperbola using the two vertices as the focal points. We use the branch of the hyperbola closer to v_4 and call it the *hyperbola branch* of v_4 (the shaded curve in Fig. 1). This hyperbola branch divides the data space into two parts. Any safe zone located on the v_4 side of the hyperbola branch has a shortest path to v_1 that goes through v_4. A path that goes directly to v_1 is longer. Therefore, no edges between v_1 and any vertex representing a safe zone on the v_4 side of the hyperbola branch is needed. This way, we have pruned a large number of possible edges.

Next, we find the nearest vertex of v_1 located on the unpruned side of the hyperbola branch, create an edge between it and v_1, and compute another hyperbola branch to prune edges. This process repeats until no vertex is left to be connected to v_1 (cf. Fig. 7b). Having done the above for every vertex, we obtain a graph that contains all the necessary edges for safest path computation. In Section 4.4, we prove that the hyperbola-based pruning algorithm is safe in that no edge that can belong to a shortest path is pruned.

As will be discussed soon, although this graph construction algorithm is very effective, it may still contain a small number of edges that will not be in any shortest path. We call these edges *superfluous edges*. To filter superfluous edges, we run Dijkstra's algorithm for every pair of vertices that have an edge between them, and we only keep an edge if it is the shortest path between the two vertices.

Algorithm 1 summarizes the process described above, with a spatial index $\mathcal{Q}$ to index the safe zones for fast nearest

vertex computation. We use the quad-tree [26] in our implementation, although any hierarchical index can be used.

Query processing: When an SPSZ query is issued, there are three cases of the origin and destination points: (i) both are in safe zones; (ii) one of them is in a safe zone; (iii) both are outside safe zones. If either is in a safe zone, we simply use the safe zone to replace the point. Otherwise, we add edges to connect the points to the graph G°, which is done by applying the edge creation strategy described above. We then run Dijkstra's algorithm to find the safest path. We do not need to filter superfluous edges because Dijkstra's algorithm is run anyway.

Next we detail how the hyperbolas are used for edge pruning for round and polygon safe zones, respectively.

4.4 Hyperbolas for Round Safe Zones

Based on the hyperbola properties described in Section 3.3, we formalize the pruning strategy as the following theorem.

Theorem 1 *Let two round safe zones v and u be given along with a hyperbola defined by:*

$$(|\overline{v_c, p}| - r_v) - (|\overline{u_c, p}| - r_u) = |v, u|^\perp$$

For any point p located beyond the hyperbola branch closer to u, it is safer to travel through u rather than to travel directly from v, i.e.,

$$(|\overline{v_c, p}| - r_v) > (|\overline{u_c, p}| - r_u) + |v, u|^\perp$$

Proof The correctness of the theorem is guaranteed by the definition of hyperbola as discussed in Section 3.3.

As an example, in Fig. 6, $|\overline{p_1, v_c}| - r_v = 4$, $|\overline{(p_1, u_c)}| - r_u = 1$ and $|v, u|^\perp = 3$. At p_1, it is the same in terms of safety to travel to u then to v as to travel directly to v. The same applies for point p_2 because $|\overline{(p_2, v_c)}| - r_v = 1.5$ and $|\overline{(p_2, u_c)}| - r_u = 4.5$. However, point p_3 is located beyond the hyperbola branch closer to u, and hence it is safer to travel through u to v rather than to v directly. In contrast, p_4 is not located beyond the hyperbola branch. Traveling from p_4 to v directly is safer than traveling through u:

$$|\overline{p_4, v_c}| - r_v = 6.37 < |\overline{p_4, u_c}| - r_u + |v, u|^\perp = 6.53.$$

Theorem 1 guarantees no false negatives in edge creation, i.e., only superfluous edges are pruned. However, it cannot guarantee no false positives, meaning that superfluous edges can be created. While such edges do not affect correctness, they may affect performance. Since the number of superfluous edges is usually small due to the pruning capability of the HyperEdges algorithm (within 1% of the total number of edges created in experiments), filtering them incurs small overhead.

There are two cases of superfluous edges:

- If the safe zone of a vertex spans the two sub-spaces created by a hyperbola branch, we cannot prune the vertex since it contains points that should not be pruned. However, it is still possible that it is safer to travel from the vertex through u to v. Figure 1 shows an example, where edges are being created for v_1, and a hyperbola is drawn for v_1 and v_4. Vertex v_7 spans the two sub-spaces created by the hyperbola branch of v_4, and hence it cannot be pruned by v_4. This type of vertex usually does not yield many superfluous edges, as they may still be fully enclosed in a sub-space pruned by some other vertices. For example, in Fig. 7b where hyperbolas have been drawn for v_1 and a few more vertices, v_7 is pruned by v_3.
- When adding an edge between v and u, our pruning strategy essentially discards the vertices whose respective shortest paths to v contain only one intermediate vertex u. However, we cannot prune the vertices whose respective shortest paths to v contain additional intermediate vertices, even though this case is infrequent especially in the Euclidean setting.

Correctness: The correctness of using hyperbolas to prune the edges is guaranteed as follows. Using the proposed hyperbola based pruning algorithm, only when there is a path $v \rightarrow u \rightarrow ... \rightarrow w$ shorter than edge $v \rightarrow w$, the edge $v \rightarrow w$ will be pruned. Such a path will not exist if edge $v \rightarrow w$ appears in some shortest path. This means that the graph constructed in the pre-computing stage keeps the edges that appear in at least one of the shortest paths between any two safe zones. Therefore, in the path finding stage the correct shortest path can be found.

Complexity: The key advantage of the HyperEdges algorithm is that, as Fig. 6 shows, a hyperbola branch can prune edges to vertices representing safe zones in a significant portion of the space. If the vertices surrounding a vertex v are distributed evenly, only a few hyperbola branches are needed to cover the whole space. Thus, only a few edges will be created for each vertex. Roughly speaking, unless the vertices are highly skewed, the number of edges created per vertex is on the order of $O(1)$, and the number of edges created for $|\mathcal{V}|$ vertices is on the order of $O(|\mathcal{V}|)$. In the worst case, none of the safe zones can serve as an intermediate node in any other safe zone's safest paths. Then no edge can be pruned and the time complexity to generate these edges is the same as the naive algorithm. However, this is usually not the case for real data and our pruning algorithm is very effective as shown by experiments. In contrast, the naive algorithm always creates an edge for every pair of vertices, i.e., the number of edges created for $|\mathcal{V}|$ vertices is always on the order of $O(|\mathcal{V}|^2)$.

4.5 Hyperbolas for Polygonal Safe Zones

For polygons, we need more than one hyperbola per pair of vertices to prune edges. This is because a polygon does not have a center that is equidistant to every point on the boundary, and therefore we cannot define $(|\overline{v_c, p}| - r_v) - (|\overline{u_c, p}| - r_u) > |v, u|^{\perp}$ on the centers v_c and u_c of two polygons v and u. To overcome this difficulty we divide the space for each vertex into multiple partitions and use multiple hyperbolas for pruning the different partitions.

We observe that the above inequality for round safe zones essentially defines a region where the unsafe distance from a point p to v ($|\overline{v_c, p}| - r_v$) exceeds the sum of the unsafe distance from p to u ($|\overline{u_c, p}| - r_u$) and the minimum unsafe distance between v and u ($|v, u|^{\perp}$). We rewrite this inequality as follows:

$$|v, p|^{\perp} - |u, p|^{\perp} > |v, u|^{\perp}$$

Here, $|v, p|^{\perp}$ and $|u, p|^{\perp}$ denote the minimum unsafe distance from p to v and u, respectively.

We relax this inequality as follows to obtain the hyperbola for two polygon safe zones v and u:

$$|\overline{v_{p1}, p}| - |\overline{u_{p1}, p}| > |v, u|^{\perp}$$

Here, v_{p1} and u_{p1} are two points from v and u that serve as the focal points (cf. Fig. 8, where the dashed line connecting v and u is $|v, u|^{\perp}$). We need to find the two points that satisfy the following for every point p:

$$|\overline{v_{p1}, p}| \leq |v, p|^{\perp} \text{ and } |\overline{u_{p1}, p}| \geq |u, p|^{\perp},$$

so that the original inequality is also satisfied, and we guarantee the correctness of pruning using the hyperbola. By definition, $|\overline{u_{p1}, p}| \geq |u, p|^{\perp}$ is satisfied for any point u_{p1} on u. Meanwhile, $|\overline{v_{p1}, p}| \geq |v, p|^{\perp}$ holds for any point v_{p1} on v. Thus, to satisfy $|\overline{v_{p1}, p}| \leq |v, p|^{\perp}$, we need a point v_{p1} such that $|\overline{v_{p1}, p}| = |v, p|^{\perp}$. This means that v_{p1} must be the closest point on v to any point p. Since different points have different closest points on v, no single point v_{p1} can satisfy $|\overline{v_{p1}, p}| = |v, p|^{\perp}$ for every point p. To overcome this limitation, we divide the space into multiple partitions, where points in each partition share the same closet point on v.

We extend the edges of the *minimum bounding rectangle* (MBR) of v across the space to divide the space into 8 partitions $s_1, s_2, ..., s_8$ as shown in Fig. 8. We use the MBR of v rather than v directly to simplify the finding of the closest point on v for the points in each partition. This does not introduce false negatives because any point outside the MBR must be at least as close to the MBR as it is to v. It may result in superfluous edges, but the number of these is expected to be small as the MBR is usually a good approximation.

After the space division, every shaded partition has only one point in the MBR of v as its closest point, which is the corresponding corner point. For example, in Fig. 8, v_{p1} is the closest point among the points in partitions s_1 to the MBR. Similarly, v_{p2}, v_{p3}, and v_{p4} are the closest points for the points in partitions s_3, s_5, and s_7, respectively. We use the corner point and a point from u to compute a hyperbola for pruning in a shaded partition. The non-shaded partitions still do not have a unique point that is the closest to all the points in the partition. This causes a problem which we call the *blind region* problem. A blind region is a small region with uncertain safety that therefore cannot be used straightforwardly for edge pruning. Blind regions are covered shortly.

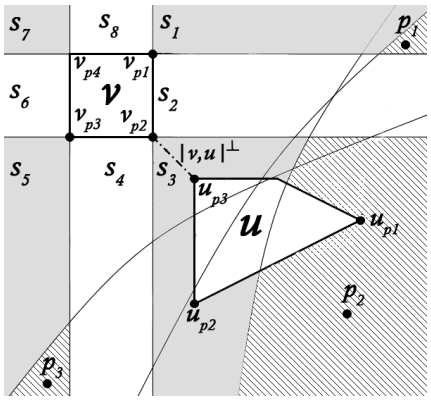
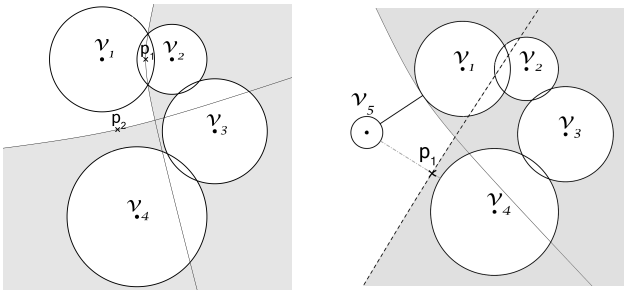


Fig. 8: Hyperbolas for three partitions

Since there are multiple partitions to be used for pruning, we compute multiple hyperbolas, each with a different pair of focal points. In Fig. 8, three hyperbola branches are computed for partitions s_1, s_3, and s_5. When u is in different partitions, we use different sets of hyperbolas, which is straightforward and hence omitted.

Blind region: As mentioned above, for an unshaded partition, there is no single point that can serve as a focal point. This is because different points in an unshaded partition have different closest points on the MBR of v. For example, in Fig. 8, points in s_2 may view v_{p1}, v_{p2} or some other point in-between as their closest point on the MBR, depending on the positions of the points. As a result, there is no single hyperbola for pruning in s_2. However, we can still achieve some pruning in this type of partitions.

Assume that we can find all the points on $\overline{v_{p1}, v_{p2}}$ and compute hyperbolas for them with a focal point on u, such as u_{p1}. Then the intersection of the pruning regions of these hyperbolas is a pruning region. Since this is impossible, we relax the pruning by only using the hyperbola of v_{p1} and u_{p1}. Because v_{p1} is farthest from u_{p1} among all the points on $\overline{v_{p1}, v_{p2}}$, we know that $Hyp(v_{p1}, u_{p1})$ must have the rightmost intersection point on the extended edge of $\overline{v_{p4}, v_{p1}}$. Any point in s_2 to the right of the intersection is enclosed by the pruning region defined by the hyperbola of any other point on $\overline{v_{p1}, v_{p2}}$. In contrast, we cannot infer that it is safe



(a) Same overlapping set (b) Different overlapping sets

Fig. 9: Hyperbolas for overlapping round safe zones

to use the region to the left of the intersection for pruning; thus, we call it a *blind region* and do not use it for pruning.

4.6 Overlapping Safe Zones

Until now we have implicitly assumed that safe zones do not overlap. This assumption may not hold in the Euclidean setting. Thus, the minimum unsafe distance between two safe zones may not be the direct distance between them, but a distance through some other safe zones they overlap with. We thus adjust the HyperEdges algorithm to handle this case.

Overlapping round safe zones: We first group the vertices (safe zones) to form subsets of vertices $\mathcal{V}_1$, $\mathcal{V}_2$, ..., $\mathcal{V}_m$ as follows. For each vertex v, if it has not been assigned to any subset, we find the subset that contains at least one vertex that overlaps v assign v to the subset. If no such subset is found, we create a new subset and assign v to it. Then for any subset $\mathcal{V}_i$, traveling between any two points in $\mathcal{V}_i$ can always occur within safe zones, and the minimum unsafe distance between any two vertices in $\mathcal{V}_i$ is 0. For example, in Fig. 9a, all the vertices belong to the same subset, while in Fig. 9b, there are two subsets $\{v_1, v_2, v_3, v_4\}$ and $\{v_5\}$.

In edge creation, if two vertices v and u are in the same subset, the minimum unsafe distance between them is 0, which is then used as the constant k in hyperbola computation, i.e., we compute the hyperbola as:

$$(|\overline{v_c, p}| - r_v) - (|\overline{u_c, p}| - r_u) = 0$$

If v and u are in two subsets $\mathcal{V}_i$ and $\mathcal{V}_j$, we use the minimum unsafe distance between the two subsets as the constant k rather than the minimum unsafe distance between the two vertices. This is because the former may be shorter as there is no cost of travel in the same subset. Formally,

$$(|\overline{v_c, p}| - r_v) - (|\overline{u_c, p}| - r_u) = \min\{|v_i, v_j|^{\perp}\},$$

where $v_i \in \mathcal{V}_i$ and $v_j \in \mathcal{V}_j$. For example, in Fig. 9b, to compute the hyperbola for v_5 and v_4, we use the distance between v_5 and v_1 (instead of v_4) as k.

Overlapping polygonal safe zones: Handling overlapping polygonal safe zones is simpler. When two polygonal safe zones v and u overlap, we simply merge them and generate a new polygon $m_{v,u}$. This newly generated polygon $m_{v,u}$ then replaces v and u in all subsequent computations.

5 SPPZ Query in the Euclidean Setting

The SPPZ problem generalizes the SPSZ problem in that it assigns a cost to travel in preferred zones that is a non-zero fraction of the cost of travel outside preferred zones. As exemplified in Fig. 4, different paths that enter and exit the same preferred zones at different points can all be the optimal path, depending on the cost of travel inside the preferred zones. To solve the SPPZ problem, we need to not only identify the optimal sequence of preferred zones to go through, but also the optimal entry and exit points to go through these preferred zones. In Section 5.3 we show that even for a single preferred zone, there is no closed-form equation for computing the optimal entry and exit points. This makes it impossible to compute precisely the optimal entry and exit points for a whole sequence of preferred zones. Further, this means that it is impossible to compute precisely the optimal sequence of preferred zones to go through.

In the following we present a heuristic based approximate solution for the SPPZ problem.

5.1 SPPZ Solution Framework

We adapt the two-stage solution framework presented in Section 4.1 to compute a sequence of preferred zones as an approximate solution. We add an extra step to this framework to optimize the entry and exit points of the preferred zones on the (approximately) optimal path found.

The modified framework becomes as follows:

- **Stage 1:** Precompute an approximate graph G° on the set $\mathcal{S}$ of preferred zones based on parameter α, which introduces a weighted cost of travel inside preferred zones when computing the hyperbolas and creating the edges in G°. We detail this stage in Section 5.2.
- **Stage 2:** When an SPPZ query with an origin point o and a destination point d is issued:
  (a) Add o and d to G° to form a graph G.
  (b) Perform a lowest-cost path search between o and d on G while considering the weighted cost of travel inside preferred zones.
  (c) Optimize the shortest path found by computing the (approximately) optimal entry and exit points of the preferred zones on the path, which is detailed in Section 5.3.

Here, the reason for adding Stage 2 (c) is that the optimal entry and exit points of preferred zones are query points dependent. Precomputing a graph G° without the query points

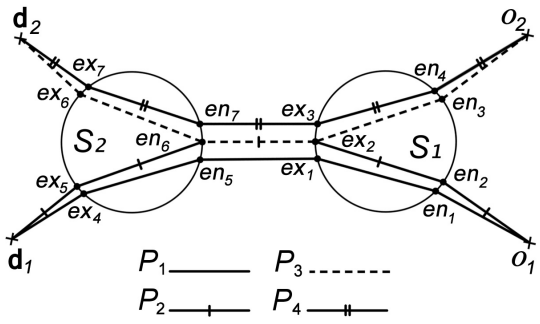


Fig. 10: Preferred zones with multiple edges

may not produce the safest path in query processing. Figure 10 illustrates the problem. There are two query origins o_1 and o_2 and two destinations d_1 and d_2. If s_1 and s_2 were safe zones, the safest path between them is always $\overline{ex_2, en_6}$, regardless of the origin and destination points. However, when s_1 and s_2 are preferred zones with weighted costs, the safest path between them might be $\overline{ex_1, en_5}$, $\overline{ex_2, en_6}$, or $\overline{ex_3, en_7}$, depending on the origin and destination used.

Using this adapted framework, an SPPZ query based on Fig. 10 ($\alpha \neq 0$) is processed as follows. At Stage 1, we precompute an approximate graph G° consisting of the two preferred zones and the edge $\overline{ex_2, en_6}$. At Stage 2, when an SPPZ query is issued, e.g., from o_1 to d_1, the two query points are added to G° to form the graph G through the edges $\overline{o_1, en_2}$ and $\overline{ex_5, d_1}$, which are their corresponding shortest paths to reach any preferred zone (Stage 2 (a)). Next, a shortest path algorithm is run, which returns the path P_2 consisting of the computed edges (Stage 2 (b)). The entry and exit points of the preferred zones are optimized (Stage 2(c)), with the entry and exit points en_2, ex_2, en_6, and ex_5 being replaced by en_1, ex_1, en_5, and ex_4, respectively. This yields an optimized path P_1 returned as the query result.

We detail only Stages 1 and 3 (c) in the following subsections, as the other stages are straightforward.

5.2 Precomputing an Approximate Graph

We first consider the case where a single preference level α is used and then consider the case of a varying preference level.

5.2.1 Single Preference Level

We show how to adapt HyperEdges to produce an approximate graph for the SPPZ problem. We need to integrate the cost of travel inside preferred zones into the definition equation of hyperbolas.

Let u be a circular preferred zone, u_c, r_u be the center and radius of u, and en_1, ex_1 be points on the boundary of u. The weighted distance for travel inside u from en_1 to ex_1 is denoted by $S(\overline{en_1, ex_1})$, which equals $\alpha|\overline{en_1, ex_1}|$. Since $\overline{en_1, ex_1}$ is a line segment inside u, $|\overline{en_1, ex_1}|$ cannot exceed the diameter of u, i.e., $|\overline{en_1, ex_1}| \leq 2r_u$. Thus:

$$S(\overline{en_1, ex_1}) = \alpha|\overline{en_1, ex_1}| \leq 2\alpha r_u \quad (1)$$

Let v be another circular preferred zones, with center v_c and radius r_v. Let $|v, u|^\perp$ be the shortest distance between u and v. We integrate the weighted cost of travel inside the preferred zones based on the following theorem.

Theorem 2 *Given two circular preferred zones u and v and the hyperbola defined by:*

$$(|\overline{v_c, p}| - r_v) - (|\overline{u_c, p}| - r_u + 2\alpha r_u) = |v, u|^\perp$$

For any point p located beyond the hyperbola branch closer to u, it is preferable to travel through u rather than to travel directly to v.

Proof Given a hyperbola as defined in the theorem, any point p beyond the hyperbola branch closer to u satisfies

$$(|\overline{v_c, p}| - r_v) > (|\overline{u_c, p}| - r_u + 2\alpha r_u) + |v, u|^\perp$$

Here, $|\overline{v_c, p}| - r_v$ is the cost of traveling directly to v from p; $|\overline{u_c, p}| + |v, u|^\perp$ is the cost of traveling directly to u from p and to v from u; $2\alpha r_u$ is the upper bound of the cost of travel inside u as shown in Equation 1. The inequality implies that there is at least one path travel to v through u that is shorter than travel directly to v. Therefore, it is preferable to travel through u than to travel directly to v.

Note that there may be even shorter paths through u, depending on the weight α and positions of p, u, and v.

Figure 11 gives an example where $\alpha = 0.5$. Point p is on the right branch of the hyperbola defined according to Theorem 2:

$$|\overline{p, v_c}| - r_v = 7 = |\overline{p, u_c}| - r_u + |v, u|^\perp + 2\alpha r_u = 2 + 3 + 2$$

This equation assumes that when traveling inside u, one travels from en_1 to u_c and then to ex_1. In reality, one can travel directly from en_1 to ex_1 to obtain a shorter path. Travel through u is preferred over travel directly to v. For point p' that is beyond the right branch of the hyperbola, the following inequality always holds:

$$|\overline{p', v_c}| - r_v > |\overline{p', u_c}| - r_u + |v, u|^\perp + 2\alpha r_u$$

Thus, there is no need to create any edge to connect v and the preferred zones beyond the right branch of the hyperbola.

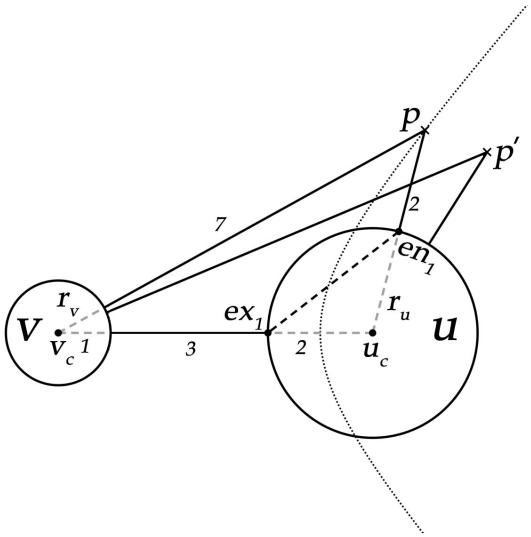


Fig. 11: Hyperbolas for SPPZ ($\alpha = 0.5$)

5.2.2 Varying Preference Level

In a setting where the preference for travel inside safe zones, i.e., parameter α, varies, precomputation faces the challenge that it is impossible to precompute graphs for all possible α values. Instead, we can precompute graphs for a few α values (e.g., $\alpha = 0, 0.2, 0.4, 0.6, 0.8$). When a user issues an SPPZ query with preference value α', we use the precomputed graph with the smallest α no smaller than α' to answer the query. This works because hyperbolas defined with a larger α value have narrower curves and hence smaller pruning power in creating the edges. A graph constructed with a larger α value includes all the edges for any graph constructed with a smaller α value. This also enables incremental computation of the safest path. If the user later uses another preference value α'' for the query and α'' falls in the same range as α', we can reuse the preferred zones in the shortest path computed for α', and we need only to recompute the entry and exit points for them. If α'' falls in a different range then we need to recompute the shortest path with a different precomputed graph.

Our problem definition uses a universal preference weight α on all preferred zones. However, when different zones have different user preference levels, our algorithms still apply, but with possibly lower efficiency. There are two cases to consider: (i) If all different user preference levels are pre-known, we can use the largest preference level of all the adjacent preferred zones of a preferred zone v to compute hyperbolas for v. Extra edges may be created since larger α values create narrower hyperbolas, which have lower pruning power. This may impact the path finding efficiency, but the correctness of the hyperbola based pruning still holds. (ii) If the preference levels are only known at query time, we need to at least know an upper bound of any preference level allowed. Then we may use this upper bound to precompute a graph. Similar to Case (i), extra edges may be created, but the correctness of the hyperbola based pruning still holds.

5.2.3 Algorithm Correctness

The correctness of pruning with hyperbolas in the SPPZ problem is guaranteed by Theorem 2, and there are no false negatives in edge creation. However, superfluous edges can still be created due to the use of an upper bound on the cost of travel inside preferred zones in Theorem 2. We can filter the superfluous edges using Dijkstra's algorithm. Meanwhile, in Theorem 2 we assume that the edge between two preferred zones or between a point and a preferred zone is simply the shortest line segment between them. This may not be true as discussed in Section 5.2. Next, we describe two optimization techniques that aim to alleviate the possible negative impact of this assumption.

5.3 Optimizing the Preferred Zone Entry and Exit Points

We optimize the entry and exit points for each preferred zone on the approximate safest path using a greedy approach. Starting from the first preferred zone reached from the origin, for each preferred zone, we optimize its entry and exit points for the local path from the exit point of the preceding preferred zone (or the origin if no preceding preferred zone) to the entry point of the following preferred zone (or the destination if no subsequent preferred zone). This process is repeated until we reach the destination.

We use the path from o_1 to d_1 in Fig. 10 to illustrate the process. The shortest path computation in the previous stages returns P_2 as the path from o_1 to d_1, which consists of preferred zone entry and exit points en_2, en_6, ex_2, and ex_5. We start with optimizing the local path from the origin o_1 to the entry point en_6 of s_2. This results in new entry and exit points for s_1, namely en_1 and ex_1. Next, we perform the local path optimization from ex_1 to d, which results in new entry and exit points for s_2, namely en_5 and ex_4. Now we have reached the destination, and the optimization process terminates. The new path P_1 formed by the new entry and exit points is then returned as the query answer.

This approach reduces the optimization of a series of entry and exit points to a simpler problem: given two points A and D outside a circle S, find two points B and C on S so that $|P| = |\overline{AB}| + \alpha|\overline{BC}| + |\overline{CD}|$ is minimized.

We use the polar coordinate system to formulate this reduced optimization problem. As exemplified in Fig. 12, the pole O is located at the center of the circle S. The coordinates of A, B, C, and D are (ρ_A, θ_A), (R, θ_B), (R, θ_C), and (ρ_D, θ_D), respectively. Here, R is the radius of S; ρ_A and ρ_D are the Euclidean distance between O and A and

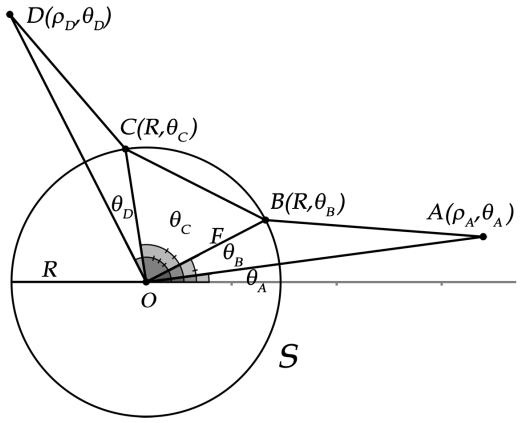


Fig. 12: Preferred zone entry and exit points optimization

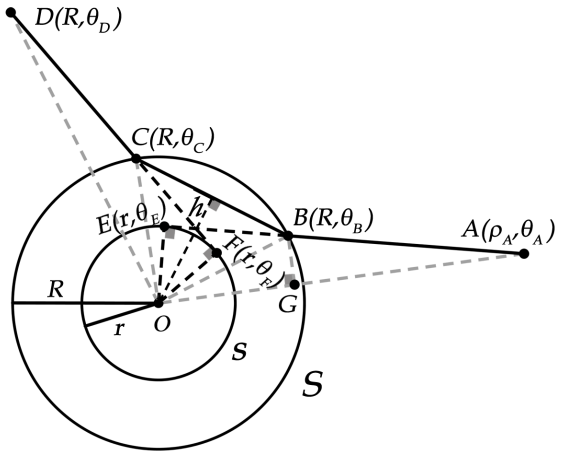


Fig. 13: The optimization condition

D, respectively; θ_A and θ_D are the angles of A and D from the polar axis, respectively. These parameters can be considered as known. Without loss of generality, we assume that $0 \leq \theta_A < \theta_D \leq \pi$. The unknown parameters are θ_B and θ_C, which are the angles of B and C from the polar axis. They satisfy $\theta_B, \theta_C \in [\theta_A, \theta_D]$ and $\theta_B \leq \theta_c$. The goal is find values for them that minimize $|P|$:

$$\min_{\theta_B,\theta_C} \quad |P| = |\overline{AB}| + \alpha|\overline{BC}| + |\overline{CD}|,$$
$$0 \leq \theta_A \leq \theta_B \leq \theta_C \leq \theta_D \leq \pi$$

According to the law of sines and the law of cosines:

$$|\overline{AB}| = \sqrt{\rho_A^2 + R^2 - 2\rho_A R\cos(\theta_B - \theta_A)}$$
$$|\overline{BC}| = 2R\sin\tfrac{\theta_C - \theta_B}{2}$$
$$|\overline{CD}| = \sqrt{\rho_D^2 + R^2 - 2\rho_D R\cos(\theta_D - \theta_C)}$$

To derive the optimal values of θ_B and θ_C, we first compute the partial derivatives:

$$\begin{aligned}
&\frac{\partial|P|}{\partial\theta_B} \\
&= \frac{\rho_A R\sin(\theta_B - \theta_A)}{\sqrt{\rho_A^2 + r^2 - 2\rho_A R\cos(\theta_B - \theta_A)}} - \alpha R\cos\frac{\theta_C - \theta_B}{2} \\
&= \frac{\rho_A R\sin(\theta_B - \theta_A)}{|\overline{AB}|} - \alpha R\cos\frac{\theta_C - \theta_B}{2} \\
&\frac{\partial|P|}{\partial\theta_C} \\
&= \alpha R\cos\frac{\theta_C - \theta_B}{2} - \frac{\rho_D R\sin(\theta_D - \theta_C)}{\sqrt{\rho_D^2 + r^2 - 2\rho_D R\cos(\theta_D - \theta_C)}} \\
&= \alpha R\cos\frac{\theta_C - \theta_B}{2} - \frac{\rho_D R\sin(\theta_D - \theta_C)}{|\overline{CD}|}
\end{aligned} \tag{2}$$

By letting $\frac{\partial|P|}{\partial\theta_B} = \frac{\partial|P|}{\partial\theta_C} = 0$, we have:

$$\begin{aligned}
&\frac{\rho_A R\sin(\theta_B - \theta_A)}{|\overline{AB}|} = \alpha R\cos\frac{\theta_C - \theta_B}{2} \\
&= \frac{\rho_D R\sin(\theta_D - \theta_C)}{|\overline{CD}|}
\end{aligned} \tag{3}$$

The first and last expressions in Equation 3 are the heights of two triangles $\triangle AOB$ and $\triangle COD$. This can be seen from Fig. 13, where $|\overline{OB}| = R$ and $\angle AOB = \theta_B - \theta_A$. Then $R\sin(\theta_B - \theta_A)$ is the height on edge $\overline{OA}$, denoted by $\overline{BG}$. Further, the length of $\overline{OA}$ is ρ_A. Thus, $\rho_A R\sin(\theta_B - \theta_A)$ is twice the area of $\triangle AOB$. Divided by $|\overline{AB}|$, this gives the height of $\triangle AOB$ on edge $\overline{AB}$. We denote this height by $\overline{OE}$ in the figure. A similar analysis applies to $\triangle COD$, and the last expression in the equation is the height on edge $\overline{CD}$, denoted by $\overline{OF}$. Therefore,

$$|\overline{OE}| = \frac{\rho_A R\sin(\theta_B - \theta_A)}{|\overline{AB}|} = \frac{\rho_D R\sin(\theta_D - \theta_C)}{|\overline{CD}|} = |\overline{OF}|$$

For the second expression $\alpha R\cos\frac{\theta_C - \theta_B}{2}$, since $\triangle BOC$ is an isosceles triangle ($|\overline{OB}| = |\overline{OC}| = R$), we can derive that $R\cos\frac{\theta_C - \theta_B}{2}$ is the height on edge $|\overline{BC}|$.

Let $r = |\overline{OE}| = |\overline{OF}|$ and $h = R\cos\frac{\theta_C - \theta_B}{2}$. Then we have:

$$r = \alpha h \tag{4}$$

We draw a circle centered at O with radius r in Fig. 13. Then $\overline{AE}$ and $\overline{DF}$ are two tangent lines of this circle. We

can rewrite θ_B and θ_C as functions of r:

$$\begin{aligned}\theta_B &= \theta_A + \angle AOE - \angle BOE \\ &= \theta_A + \arccos\frac{r}{\rho_A} - \arccos\frac{r}{R} \\ \theta_C &= \theta_D - \angle FOD + \angle FOC \\ &= \theta_D - \arccos\frac{r}{\rho_D} + \arccos\frac{r}{R}\end{aligned}$$

We replace θ_B and θ_C in Equation 4 and obtain:

$$\frac{r}{\alpha} = h = R\cos\frac{\theta_C - \theta_B}{2} = R\cos\left(\frac{\theta_D-\theta_A}{2} + \arccos\frac{r}{R} - \frac{1}{2}\arccos\frac{r}{\rho_A} - \frac{1}{2}\arccos\frac{r}{\rho_D}\right) \tag{5}$$

Equation 5 can be transformed into an equation without the cos and arccos functions. However, the transformed equation contains many square root computations and is a polynomial equation with degree greater than five. According to the Abel-Ruffini Theorem [6], there is no general algebraic solution to polynomial equations of degree five or higher with arbitrary coefficients. Thus, this equation cannot be solved with a precise solution.

We thus proceed to propose two approximate solutions to the equation.

5.3.1 Iterative Optimization

Let $f(r) = r - \alpha H(r)$, where $H(r)$ represents the h term in Equation 5 that is a function of r. Then solving Equation 5 becomes finding the root for $f(r) = 0$. Since $r = \alpha h$ and $h \le R$, αR is an upper bound on r. Further, we have:

$$f(0) = 0 - \alpha H(0) = -cos(\frac{\theta_D - \theta_A}{2}) < 0$$

$$f(\alpha R) = \alpha(R - h) \ge 0$$

Thus, there must exist one root in $[0, \alpha R]$.

Several common root-finding methods may be used, e.g., *Bisection*, *False Position*, *Newton-Raphson*, and *Secant*. All are iterative methods. Newton-Raphson and Secant offer quadratic and superlinear convergence rates. However, they both require the initial guess of the root to be close to the root. Otherwise, they may fail to converge. In contrast, Bisection and False Position both converge linearly but always keep the root bracketed and shrink the bracket at every iteration. In practice, False Position often converges faster than Bisection, but may converge slower in some cases. To avoid such cases, one option is, when such cases are recognized, to fall back to Bisection for several iterations and then resume with False Position.

Taking both speed and reliability into account, we use False Position with the fall back option to find the root. In our experiments, we find that False Position converges quickly and offers high accuracy, which is due to the steep shape of the function curve.

5.3.2 Polynomial Degree Reduction

In this second method, we transform Equation 5 by *Taylor expansion* to obtain a polynomial of degree three, which is solvable. Thus, we call this method the *polynomial degree reduction* optimization method. We first expand h based on the priority that given two angles x and y, we have $\cos(x + y) = \cos x \cos y + \sin x \sin y$.

$$\begin{aligned}h &= R\cos\left(\frac{\theta_D - \theta_A}{2} + \arccos\frac{r}{R} - \frac{1}{2}\arccos\frac{r}{\rho_A} - \frac{1}{2}\arccos\frac{r}{\rho_D}\right) \\ &= R\cdot X\cos\left(\arccos\frac{r}{R}\right) - R\cdot Y\sin\left(\arccos\frac{r}{R}\right) \\ &= R\cdot X\frac{r}{R} - R\cdot Y\frac{\sqrt{R^2-r^2}}{R} = rX - Y\sqrt{R^2-r^2}\end{aligned}$$

where

$$X = \cos\left(\frac{\theta_D - \theta_A}{2} - \frac{1}{2}\arccos\frac{r}{\rho_A} - \frac{1}{2}\arccos\frac{r}{\rho_D}\right)$$

$$Y = \sin\left(\frac{\theta_D - \theta_A}{2} - \frac{1}{2}\arccos\frac{r}{\rho_A} - \frac{1}{2}\arccos\frac{r}{\rho_D}\right)$$

Squaring both sides of $rX - h = Y\sqrt{R^2 - r^2}$ and substituting h by $\frac{r}{\alpha}$ and Y^2 by $1 - X^2$, we have

$$(1 - X^2)R^2 = \left(1 - \frac{2}{\alpha}X + \frac{1}{\alpha^2}\right)r^2 \tag{6}$$

Then we approximate X by its first order Taylor expansion.

$$X \approx X|_{r=0} + \left.\frac{\partial X}{\partial r}\right|_{r=0} r,$$

where

$$\frac{\partial X}{\partial r} = \sin\left(\frac{\theta_D - \theta_A}{2} - \frac{1}{2}\arccos\frac{r}{\rho_A} - \frac{1}{2}\arccos\frac{r}{\rho_D}\right) \cdot \left(-\frac{1}{2}\frac{1}{\sqrt{1 - \left(\frac{r}{\rho_A}\right)^2}}\frac{1}{\rho_A} - \frac{1}{2}\frac{1}{\sqrt{1 - \left(\frac{r}{\rho_D}\right)^2}}\frac{1}{\rho_D}\right)$$

Further reduction requires the assumptions that $r \ll \rho_A$ and $r \ll \rho_D$. As shown in Fig. 13, these assumptions can hold when A and D are not too close to S. Under the assumptions, $\frac{r}{\rho_A} \approx 0$ and $\frac{r}{\rho_D} \approx 0$. Therefore, we can further simplify the equation as follows.

$$X \approx \sin\left(\frac{\theta_D - \theta_A}{2}\right) + \frac{1}{2}\left(\frac{r}{\rho_A} + \frac{r}{\rho_D}\right)\cos\left(\frac{\theta_D - \theta_A}{2}\right)$$

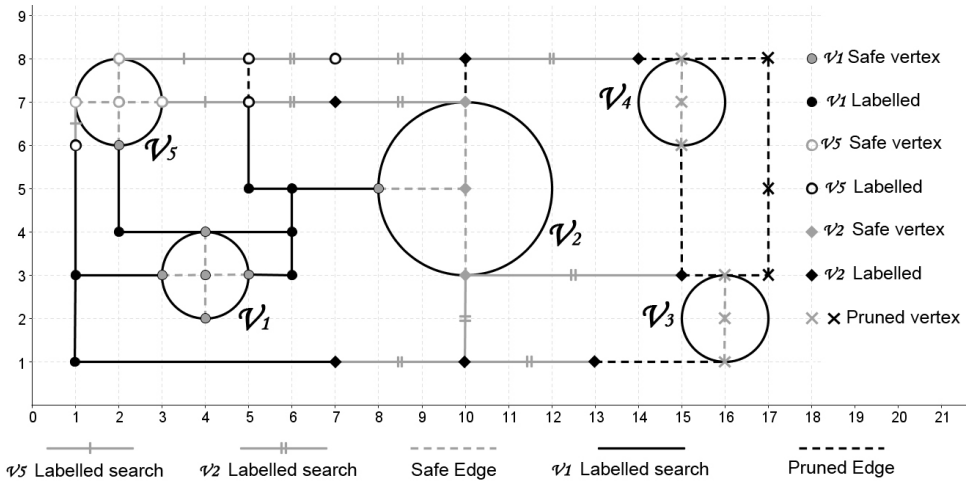


Fig. 14: HyperEdges in spatial networks

Now X is a linear function of r. The degree of the left hand side of Equation 6 is 2, and the degree of the right hand side is 3. Using the root formula for cubic equations, we can solve Equation 6 to obtain an approximate value of r. In Section 7.2, we investigate the effectiveness of this approximation empirically.

6 SPSZ and SPPZ Queries in the Spatial Networks

We consider a spatial network R represented by a graph $G_R = \langle V_R, E_R \rangle$, where vertices V_R represents the set of network vertices (e.g., intersections) and edges E_R represents the set of segments connecting the vertices. A safe (preferred) zone s in the spatial network is represented by the set of network vertices s_V and the set of edges s_E covered by the safe (preferred) zone, i.e., $s = \langle s_V, s_E \rangle$. If an edge is partly covered by a safe (preferred) zone, a new vertex is introduced at the safe (preferred) zone intersection point, and the edge is replaced by a safe and an unsafe edge. We call these network vertices and edges *safe vertices* and *safe edges*, respectively. In Fig. 14, every circle represents a safe (preferred) zone that covers the safe vertices and edges denoted by the gray points and dashed line segments.

6.1 SPSZ Query

To solve the SPSZ problem in a spatial network R, we construct a graph G° where the safe zones serve as the vertices. To connect the vertices we add the paths in G_R to G°. A *naive approach* to connect the vertices is to compute the shortest path between every pair of safe zones in G_R and add the (safe and unsafe) edges and vertices used by each such path to G_R. However, this would result in searching the whole graph G_R for each safe zone, which is expensive.

6.1.1 The HyperEdges Algorithm

We again consider using hyperbolas to reduce the edge creation cost. However, straightforwardly applying the Euclidean hyperbolas does not guarantee correct edge pruning. This is because the Euclidean distance between two points in a spatial network is usually different from the network distance. To overcome this problem, we use *network hyperbola*. A network hyperbola is a set H of points in a spatial network, where every point p in the set satisfies:

$$|dist_R(f_1, p) - dist_R(f_2, p)| = k$$

Here, $dist_R()$ returns the network distance between two points, i.e., the length of the shortest path in the spatial network. The two points f_1 and f_2 are the focal points of the network hyperbola, and k is a given positive constant. A network hyperbola contains a set of points that partition the network into two. In one partition, the shortest path to f_1 of *every* point passes through f_2. In the other partition, the shortest path to f_1 of *no* point passes through f_2. Given two safe zones s_v and s_u, their network hyperbola is computed by testing whether the points in the network vertices satisfy the hyperbola inequality[3]. When a point p is to be tested, we use two safe vertices of s_v and s_u that are the closest to p as f_1 and f_2, respectively. The network distance between s_v and s_u is used as the constant k. For example in Fig. 14, for the two safe zones v_1 and v_5, the minimum distance between them (4 distance units) is k. The network vertex at

[3] We do a best-first traversal on the spatial network and hence only a limited number of points are tested.

$(5, 7)$, denoted by $n_{5,7}$, is a network hyperbola point:

$$dist_R(n_{5,7}, n_{4,4}) - dist_R(n_{5,7}, n_{3,7}) = 6 - 2 = 4 = k$$

We use Fig. 14 to illustrate the HyperEdges algorithm using network hyperbolas:

1. For every safe zone vertex v (e.g., v_1), perform a single source shortest path search on G_R starting from all of v's border safe vertices (vertices inside v with an edge ending outside v, e.g., $n_{3,3}, n_{4,4}$). This is to find a path to connect v with every neighboring safe zone u.
2. For every safe zone u (e.g., v_5), use the network distance $dist_R(v, u)$ as k (e.g., 4) along with two safe vertices f_1 and f_2 from v and u to form a hyperbola equation. We do this for all the neighboring safe zones at the same time using a graph vertex labeling technique, which is detailed below.
3. The search of paths to neighboring safe zones continues until v is fully surrounded by network hyperbola points. The remainder of the spatial network is pruned (e.g., the 'x' vertices).

Finding the network vertices satisfying the network hyperbola condition: In the graph pre-computation stage, we need to identify the network vertices in v's partition of the spatial network as defined by a network hyperbola. A query starting at any of these network vertices towards v should go directly to v rather than going through any intermediate safe zones. We use a labeling technique to identify these network nodes. We attach a variable l to each network vertex n to store the id of the safe zone that is passed through by the search before reaching n. The labeling technique works as follows:

1. For every safe zone v (e.g., v_1), use Dijkstra's algorithm to search the neighboring network vertices in all directions. Every found network vertex (e.g., the black dots in Fig. 14) is labeled with the id of v and added to a priority queue Q. When a new neighboring safe zone u is reached by the search (e.g., v_5), we continue the search beyond u, but omit the distance to travel through u.
2. Label every network vertex reached through u with the id of u (e.g., the hollow dots for v_5) and add it to Q.
3. Continue the search as long as there are vertices labeled with v's id in Q. The search stops when Q becomes empty or contains no vertex labeled with v's id.

Here, the termination condition means that either all network vertices have been accessed or any path from the safe zone v has reached a network vertex that is labeled by some other safe zone. In the latter scenario, a network vertex n labeled by another safe zone u has a safer path through u to v than going directly to v. Since any network vertex reached from n should also go through n first, it should also go through u first, and hence a direct path to v is unnecessary.

At query time, we treat the origin and destination points as safe zones and label the network vertices for them similarly to build edges from them to the neighboring safe zones. Then the origin and destination points are connected to the precomputed graph G°, and we can apply Dijkstra's algorithm to find the safest path. Note that when the query origin and destination are close, there may not be a safe zone between them. In this case, the problem falls back to a classic graph shortest path problem. Any graph shortest path algorithm may be used to solve the problem.

6.2 SPPZ Query

For the SPPZ query, we introduce a weight $\alpha \in [0, 1)$ and add the cost of traveling in preferred zones to the formalization of network hyperbola as follows:

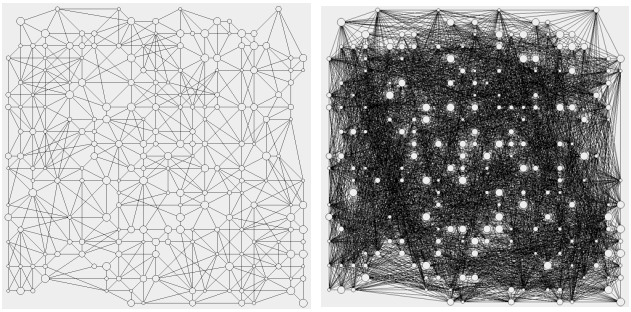
$$|dist_R(v, p) - dist_R(u, p) + (\alpha \cdot dist_R(u))| = k$$

Here, $dist_R(u)$ denotes the length of the shortest path to travel through a preferred zone u. To accommodate this cost in the HyperEdges algorithm, we just need to change the computation of the network hyperbolas in the algorithm to use the equation above.

Discussion: We have shown how the HyperEdges algorithm works in spatial network settings. We emphasize that this is an initial attempt at demonstrating the feasibility of applying Hyperbola based edge pruning in spatial network settings, thus making this paper complete and self-contained. A more extensive study that fully maps out the advantages and limitations of the algorithm us beyond the scope of this paper.

7 Experimental Study

In this section, we study the empirical performance of the proposed algorithms. In the experiments for the Euclidean setting, we compare HyperEdges with both the naive and the improved naive (here denoted as *Im-Naive*) algorithms described in Section 4.2. We observe that the naive algorithm is significantly less efficient in terms of running time and the number of created edges (and hence memory consumption) than both Im-Naive and HyperEdges: the naive algorithm takes more than a day and 500 MB memory to construct a graph for a dataset of 10,000 safe zones. It cannot handle larger datasets (e.g., with 160,000 safe zones) with feasible time and memory consumption. This is expected, as the initial graph contains an edge for every pair of safe zones. Therefore, we omit the results of the naive algorithm. In the experiments for the spatial network setting, we compare HyperEdges with the naive algorithm as described at the beginning of Section 6.



(a) HyperEdges (b) Im-Naive

Fig. 15: Edges created in graph construction

We conducted the experiments on a desktop PC with 8GB RAM and a 3.4GHz Intel$^{(R)}$ Core$^{(TM)}$ i7 CPU. The disk page size is 4KB. For the Euclidean setting experiments, we use a real dataset containing the locations of 1042 villages in Saudi Arabia[4]. The coordinates of the villages form the **base safe zone dataset**, and we generate additional safe zones following a Gaussian distribution ($\sigma = 0.166$) centered at the real villages. By default, we use 10,000 safe zones covering 0.8% of the total area of the data space. We call this the **default dataset**. For the spatial network setting experiments, we use the London road network extracted from OpenStreetMap[5], which contains 515,120 vertices and 838,702 edges. We also extracted the location of 192 police stations in London and use them as safe zone centers. We call this the **Police Station dataset**.

We vary parameters such as dataset size, safe zone density, and data distribution to gain insight into the algorithm performance in different settings. The detailed settings are given in the individual experiments.

7.1 SPSZ Experiments - Euclidean Setting

First, we evaluate the effectiveness of the HyperEdges algorithm in constraining the number of edges created in both graph construction and query processing. Then we evaluate the algorithm efficiency, also in both stages.

7.1.1 Edge Pruning Effectiveness

We measure the number of edges created by the different algorithms. We make the following observations. (i) HyperEdges creates a much smaller number of edges before filtering. The graph constructed by HyperEdges has almost no superfluous edges (less than 1%), while more than 90% of the edges created by Im-Naive are superfluous edges and need to be filtered. (ii) HyperEdges creates the same set of edges as the improved naive algorithm does after filtering.

To give a more intuitive view of the algorithm performance on edge pruning, we generate 200 round safe zones with random distribution as shown in Fig. 15. We ran HyperEdges and Im-Naive to create the edges, and the figure shows the edges produced before filtering. We can see that HyperEdges creates a much smaller number of edges, which demonstrates the effectiveness of our hyperbola based edge pruning technique.

[4] http://www.sl3sl.com/vb/showthread.php?t=7032

[5] http://metro.teczno.com/# london

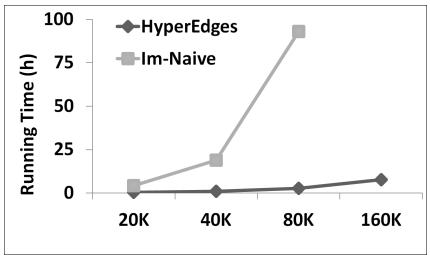


Fig. 16: Effect of safe zone cardinality (graph construction)

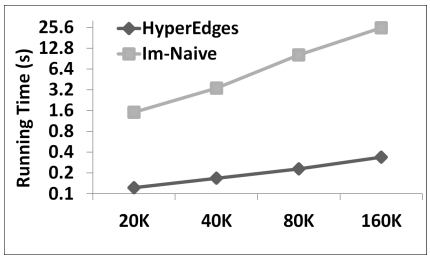


Fig. 17: Effect of safe zone cardinality (query processing)

7.1.2 Algorithm Efficiency

We measure the running time in both graph construction and query processing over different dataset size, safe zone distribution and safe zone density level. In addition, we measure the memory consumption of the two algorithms for storing the graphs created.

Effect of dataset cardinality: We used the base safe zone dataset to create datasets of different sizes. Figure 16 shows the running time on graph construction for the generated datasets. HyperEdges is more than an order of magnitude faster than Im-Naive when the dataset size is within 80,000. At the 160,000 dataset, the Im-Naive algorithm cannot finish within 7 days, and no result is reported. This again confirms the advantage of our hyperbola based edge pruning techniques in reducing the graph construction time. Also the running time of HyperEdges increases much slower with the increase in dataset cardinality in the graph construction stage, while that of Im-Naive increases dramatically. This demonstrates the scalability of HyperEdges.

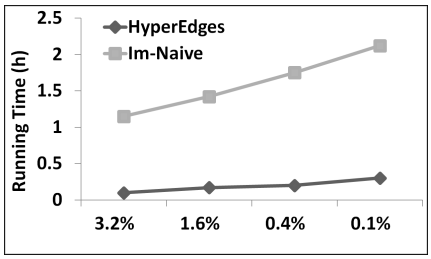


Fig. 18: Effect of density level

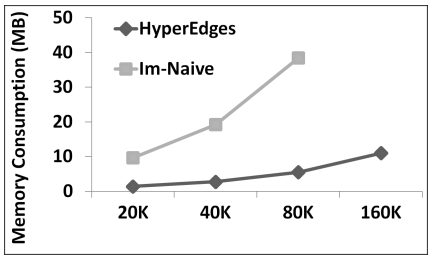


Fig. 19: Effect of memory consumption

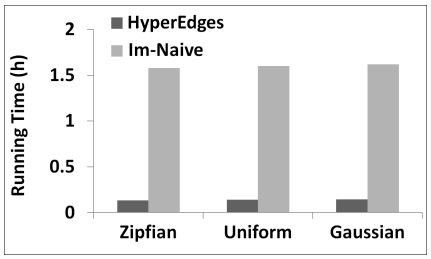


Fig. 20: Effect of distribution

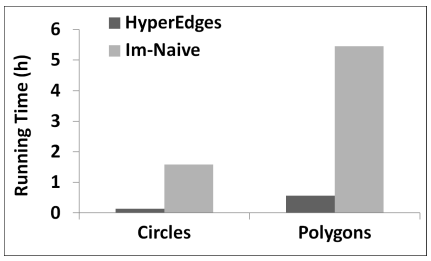


Fig. 21: Effect of zone shape

We further compare the average running time of HyperEdges with that of Im-Naive for processing 100 queries where each query has randomly generated origin and destination points. Note that in these experiments the precomputed graphs for both algorithms are the same as both algorithms produce the same graph after filtering.

Figure 17 illustrates the query processing time, which is the time taken to add the origin and destination points to the graph and to find the safest path. We observe that, HyperEdges can be more than 50 times faster than Im-Naive in query processing. This is because HyperEdges inserts the query origin and destination points into the graph using the hyperbola based technique, which is more efficient. The total query processing time of HyperEdges is at least 90% smaller than that of Im-Naive for the various datasets tested.

Effect of safe zone density: Figure 18 shows the graph construction time where we vary the percentage of the data space covered by the safe zones from 3.2% to 0.1% by varying the radii of the safe zones. Again, HyperEdges outperforms Im-Naive constantly and it is at least seven times faster for all density levels tested. Note that the size and density of the safe zones do affect the efficiency of both algorithms. For HyperEdges, the decrease in the radii of the safe zones leads to narrower hyperbolas. Thus, fewer regions can be pruned and more edges are created. When the density is 0.1%, the running time of HyperEdges is about three times that of when the density is 3.2%. This difference is not too observable in the figure due to the large range of the Y-axis.

Memory consumption: Figure 19 shows the maximum memory consumption (in MB) of the two algorithms in graph construction on the datasets of different sizes. As the figure shows, HyperEdges constantly consumes less memory comparing with Im-Naive. This is because HyperEdges creates much fewer edges. Meanwhile, the advantage of HyperEdges grows as the dataset cardinality increases. This is in accordance to our discussion at Section 4.4 that HyperEdges creates edges whose number increases approximately linearly to the number of safe zones, while the number of Im-Naive increases approximately quadratically. For the 160,000 dataset, the Im-Naive algorithm cannot finish within 7 days and no result is obtained.

Effect of safe zone distribution: Figure 20 shows the graph construction time when the safe zone distribution is varied. We use $\mu = 0.5$ and $\sigma = 0.166$ for the Gaussian distribution and $\rho = 0.1$ for the Zipfian distribution to generate 10,000 safe zones covering 0.8% of the data space, where the safe zone centers follow the given distributions around the safe zones in the base dataset. As the figure shows, HyperEdges outperforms Im-Naive in all distributions tested.

Effect of safe zone shape: Both HyperEdges and Im-Naive can process round and polygon shaped safe zones. We test their performance with the default dataset (round safe zones) and a dataset of the same parameters but with polygon safe zones. Figure 21 shows that the running time of HyperEdges in the different safe zone shapes is significantly less than that of Im-Naive. We notice that HyperEges is about three time faster in the round safe zone experiment than in the polygon safe zone experiment. This is expected as for polygon safe zones we need more hyperbolas for each pair of safe zones to constrain the edges created.

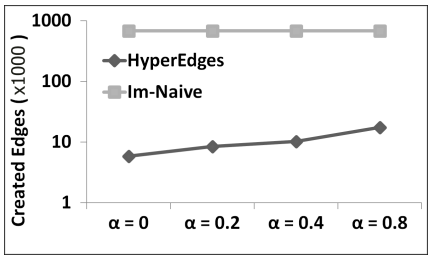


Fig. 22: Edges created in graph construction

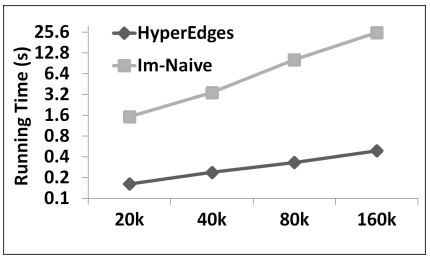


Fig. 24: Effect of preferred zone cardinality (query proc.)

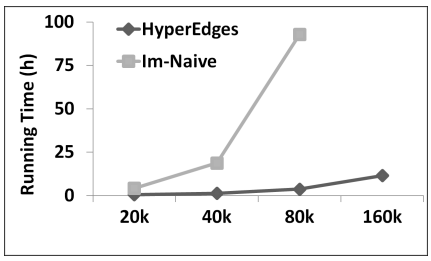


Fig. 23: Effect of preferred zone cardinality (graph const.)

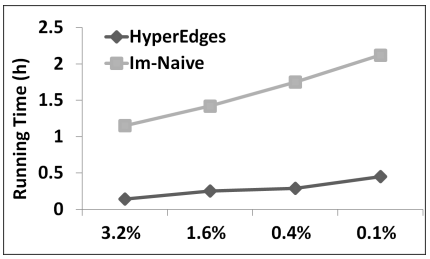


Fig. 25: Effect of preferred zone density

Experiments where these settings are varied for the query processing stage show similar results. To summarise, HyperEdges is more than an order of magnitude faster than Im-Naive for the different density levels and distributions tested. When polygonal safe zones are used, both algorithms become slower (by three times) because more edges are created. However, HyperEdges is still significantly faster. We omit the figures for conciseness.

7.2 SPPZ Experiments - Euclidean Setting

In the SPPZ experiments, we also first evaluate the effectiveness of the adapted HyperEdges algorithm in reducing the number of edges created in graph construction. Then we evaluate the efficiency of the algorithm in both graph construction and query processing. Finally, we add the optimization of the preferred zone entry and exit points to the algorithm and evaluate both the effectiveness and efficiency of the two optimization techniques. In the experiments, the improved naive algorithm is used as the baseline algorithm.

We use the same default dataset of 10,000 safe zones (now treated as preferred zones) that is used in Section 7.1. By default, the cost weighting parameter $\alpha = 0.5$.

7.2.1 Edge Pruning Effectiveness

We vary the value of α to obtain different graphs for the default dataset. We measure the number of edges created by the different algorithms before filtering. Figure 22 shows the result. We see that as α increases, the number of edges created by HyperEdges increases. This is because a larger α leads to narrower hyperbola curves and hence smaller pruned regions in edge construction. Im-Naive is not affected by α, as it does not consider α at all. The number of edges created by HyperEdges is always smaller than that of Im-Naive by more than an order of magnitude. This demonstrates the effectiveness of the adapted HyperEdges algorithm at reducing the graph size for the SPPZ problem.

7.2.2 Algorithm Efficiency

Similar to the SPSZ experiments, to evaluate the efficiency of the adapted HyperEdges algorithm, we measure the running time of both graph construction and query processing on datasets with different cardinality, preferred zone distribution, and preferred zone density. In addition, we measure the impact of the value of α on the algorithm running time, as well as the memory consumption of the two algorithms for storing the graphs created.

Effect of dataset cardinality: Figure 23 shows the result on varying dataset cardinality. HyperEdges is more than an order of magnitude faster than Im-Naive when the dataset cardinality is within 80,000. No result is obtained for Im-Naive on 160,000 preferred zones as the running time exceeds 100 hours. This confirms the effectiveness of the hyperbola based edge pruning techniques in reducing the graph construction time. The figure also shows that even with a weighted cost of travel inside preferred zones, the running time of HyperEdges increases only moderately with the increase in dataset cardinality. This demonstrates the scalability of HyperEdges when applied to SPPZ.

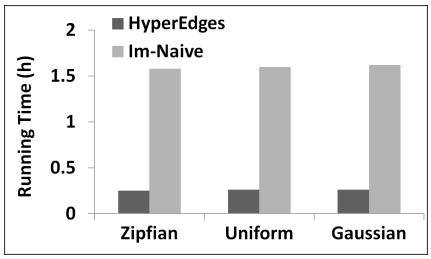


Fig. 26: Effect of preferred zone distribution

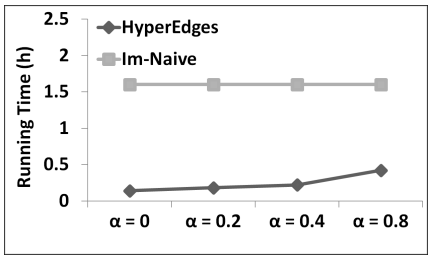


Fig. 27: Effect of α

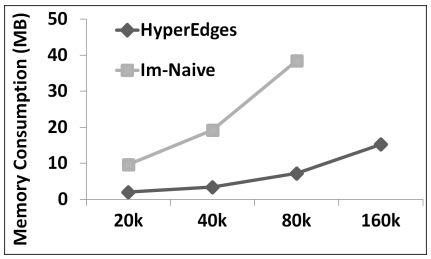


Fig. 28: Memory consumption

Table 2: Effect of the number of iterations

Iteration	Path cost
0	477.256624603370
1	460.825490733940
2	460.824809874138
3	460.824809874117
4	460.824809874116
5	460.824809874115
...	...
100	460.824809874080

Table 3: Effect of the optimization techniques

Density	20k	40k	80k	160k
No optimization	492.24	477.2	441.07	373.52
Poly. degree reduction	485.29	462.79	409.69	306.06
Iterative (2 iterations)	484.39	460.82	406.41	298.95

In the query processing stage, we compare the average running time of HyperEdges (without the path optimization) with that of Im-Naive for processing the same 100 queries used in the SPSZ cardinality experiments. Figure 24 shows the result. We observe that HyperEdges can be 50 times faster than Im-Naive when processing the SPPZ queries. This is because HyperEdges inserts the query origin and destination points into the graph using the hyperbola based technique, which is more efficient.

When comparing the performance of HyperEdges for SPSZ (Fig. 17 and Fig. 16) and HyperEdges for SPPZ (Fig. 23 and Fig. fig:Safest2A2), we see that HyperEdges for SPSZ is faster than HyperEdges for SPPZ in both graph construction and query processing. This is because when there is a cost of travel inside preferred zones, the hyperbola curves become narrower. This leads to smaller pruned regions and hence higher costs in graph construction and query processing.

Effect of preferred zone density: Figure 25 shows the graph construction time where we vary the percentage of the data space covered by the preferred zones from 3.2% to 0.1% by varying the radii of the preferred zones. Again, HyperEdges outperforms Im-Naive consistently, and it is at least five times faster. Similar to the experiment on safe zones, the performance of HyperEdges decreases as the density level of the preferred zones decreases. This is because smaller preferred zones also lead to narrower hyperbolas and hence fewer regions being pruned. Further, taking the cost of travel inside preferred zones into consideration affects the pruning capability of HyperEdges. Comparing Fig. 18 with Fig. 25 at density level 0.1%, we see a 20% performance decrease of HyperEdges when the cost of travel inside preferred zones is considered.

Effect of preferred zone distribution: Figure 26 shows the graph construction time when the preferred zone distribution is varied. We use $\mu = 0.5$ and $\sigma = 0.166$ for the Gaussian distribution and $\rho = 0.1$ for the Zipfian distribution when generating 10,000 preferred zones covering 0.8% of the data space, where the preferred zone centers follow the given distributions around the preferred zones in the base dataset. The figure shows that, HyperEdges outperforms Im-Naive for all distributions tested.

Effect of α: Figure 27 shows the graph construction time, where we vary α from 0 to 0.8. Again, HyperEdges outperforms Im-Naive consistently and is at least three times faster for all α values tested. We also observe that the running time of HyperEdges increases with α. This is because a larger α yields narrower hyperbola curves and smaller pruned regions. The running time of Im-Naive is not affected because it does not consider that cost of preferred zones.

Experiments where these settings are varied for the query processing stage show similar results. For the different density levels and distributions tested, HyperEdges only takes less than 100 milliseconds to process a query on average while Im-Naive takes more than half a second. For all

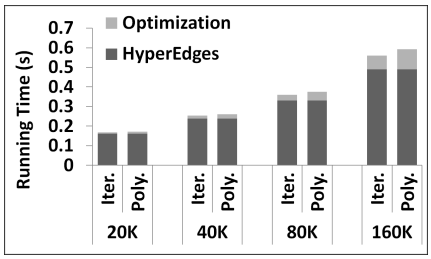


Fig. 29: Path optimization efficiency

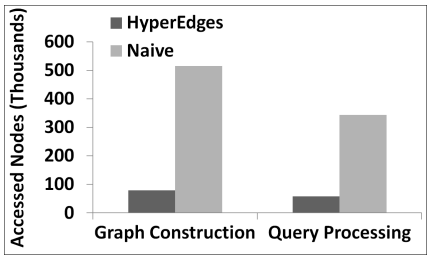


Fig. 30: Number of vertices accessed

the values of α tested, Hyperedges is at least five times faster than Im-Naive. We omit the figures for conciseness.

Memory consumption: Figure 28 shows the maximum memory consumption (in MB) of the two algorithms in SPPZ graph construction on the datasets of different cardinality. As the figure shows, HyperEdges constantly consumes less memory. This is because of the pruning power of the hyperbolas when creating edges. When the dataset cardinality increases, the memory consumption of both algorithms increases, as more edges are created. The increase is only moderate for the HyperEdges algorithm. For the 160,000 dataset, the Im-Naive algorithm cannot finish within 7 days, and no result is reported.

7.2.3 Path Optimization

Next, we evaluate the performance of the two proposed techniques to optimize the entry and exit points of the preferred zones. First we study the effect on path optimality of the number of iterations in the iterative method. Then we compare the costs of the paths obtained with and without optimization. We also evaluate the impact on the query processing time when the optimization techniques are applied. We measure the average path cost (in kilometer) of 100 randomly generated queries.

Effect of the number of iterations: In Section 5.3.1 we use an iterative method (i.e., False Position) for the optimization. Table 2 shows the average path cost obtained by different numbers of iterations on a dataset of 40,000 preferred zones. We see that after two iterations, the path cost is reduced by 16.43 kilometer. This shows that the optimization technique can effectively reduce the overall path cost. However, performing additional iterations yields little improvement. This is because the function to optimize has a steep curve. Therefore, we can early terminate the iterative optimization for better query processing efficiency. In the following experiments, we use 2 iterations.

Effect of optimization techniques: Here we test the effectiveness of the optimization techniques by varying the dataset cardinality. Table 3 shows the average cost of the safest paths obtained by three different versions of the HyperEdges algorithm: no optimization, optimization with the iterative method, and optimization with the polynomial degree reduction method.

We see that both optimization techniques can reduce the path cost in the datasets tested. As the dataset cardinality increases, the superiority of the optimization techniques becomes more significant. For the $160,000$ dataset, the cost reduction is up to 20%. This indicates that when there are more preferred zones, paths will go through more preferred zones, which brings more opportunity for the optimization. The iterative method obtains the lowest path cost, meaning that it obtains the most accurate solution for the optimization problem. The polynomial degree reduction also is able to reduce the path cost but not as much. This is because the assumption it relies on does not always hold. We also observe that path costs are lower when there are more preferred zones. This is because a larger number of preferred zones yields more opportunities for travel in the preferred zones.

Figure 29 shows the average query processing time when applying the two optimization techniques. Here the gray segment denotes the time for running the optimization techniques. We see that the optimization techniques only increase the overall query processing time by a small fraction (up to 10% for the iterative method on the $160,000$ dataset). The iterative method with two iterations runs faster than the polynomial degree reduction method. Overall, the iterative method has the best performance in both cost reduction and running time and so is the recommended method.

7.3 SPSZ and SPPZ Experiments - Spatial Network Setting

First we validate the effectiveness of HyperEdges in reducing the number of network vertices accessed in both graph construction and query processing. Then, we evaluate the algorithm efficiency. The default setting is 192 safe zones (all police stations) where each safe zone has a radius of 2 kilometers and $\alpha = 0$, and the spatial network used is the London road network.

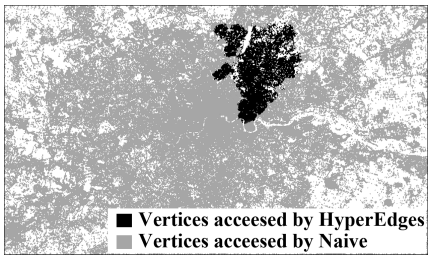


Fig. 31: Query point search range

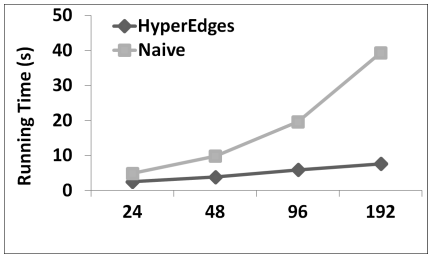


Fig. 32: Effect of zone cardinality (graph construction)

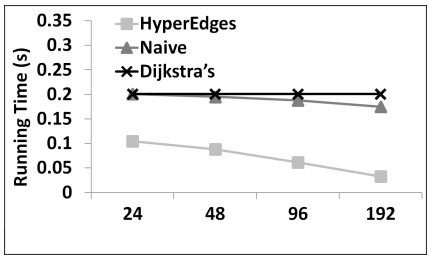


Fig. 33: Effect of zone cardinality (query processing)

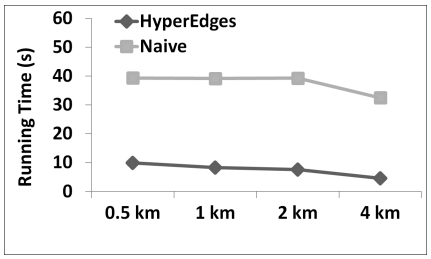


Fig. 34: Effect of zone size (graph construction)

7.3.1 Network Pruning Effectiveness

We measure the number of network vertices accessed by both HyperEdges and the naive algorithm and show that HyperEdges accesses a much smaller number of network vertices. This leads to the better performance of HyperEdges in graph construction and query processing.

Figure 30 shows the average number of vertices accessed for each safe zone (query) on the London road network. We see that the average number of vertices accessed by HyperEdges in both graph construction and query processing is about 85% less than that of the naive algorithm. Figure 31 illustrates the search range needed by the two algorithms to add a random query point to the constructed graph in the London network. As the figure shows, the naive algorithm accesses about nine times more vertices.

7.3.2 Algorithm Efficiency

We measure the running time in both graph construction and query processing using different number of safe zones, safe zone sizes and α values. In the query processing stage in the following experiments, we further compare the running time of HyperEdges with that of running Dijkstra's algorithm on the original road network where the cost of travel on the edges in safe (preferred) zones is weighted by α.

Effect of safe zone cardinality: We randomly sample the Police Station dataset to obtain safe zone datasets of different sizes. Figure 32 shows that the running time of HyperEdges is up to 85% less than that of the naive algorithm in graph construction. This is due to the advantage of using the hyperbola labeling technique to prune the edges considered. Figure 33 shows that, the average running time of HyperEdges is again up to seven times faster than those of the naive and Dijkstra's algorithms. An important observation is that the performance of HyperEdges improves as there are more safe zones, while the naive and Dijkstra's algorithms are almost unaffected. This is because HyperEdges terminates the search earlier when it reaches enough neighboring safe zones, while the naive and Dijkstra's algorithms may search the entire network.

Effect of safe zone size: We vary the radius of each safe zone from 0.5 to 4 kilometers. As Fig. 34 and Fig. 35 show, HyperEdges again outperforms the naive and Dijkstra's algorithms consistently. We notice that when the safe zone size is 4, both HyperEdges and naive algorithms achieve the best performance. This is due to overlapping of the safe zones, which yields fewer safe zones, as discussed in Section 4.6. This in turn, reduces the graph construction time. On the other hand, running Dijkstra's algorithm on the original road network is not affected by the size of the safe zones as the edges in the safe zones are not treated differently by the algorithm. We also observe from Fig. 35 that the naive algorithm outperforms Dijkstra's algorithm as the size of the safe zones increases. This is because, when the safe zones are larger, the query points have a higher probability to be in safe zones, making it sufficient for the naive algorithm to use the precomputed (smaller) graph for query processing.

Effect of α: We further evaluate the effect of the value of α for the SPPZ query, which controls the weight of the cost of traveling inside the preferred zones. We vary the value of α from 0 to 0.75. As Figs. 36 and 37 show, the running time

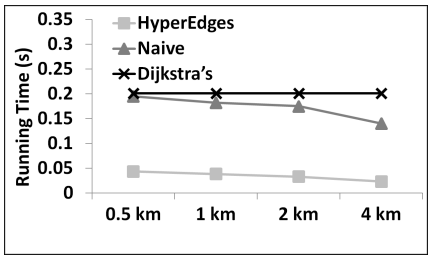


Fig. 35: Effect of zone size (query processing)

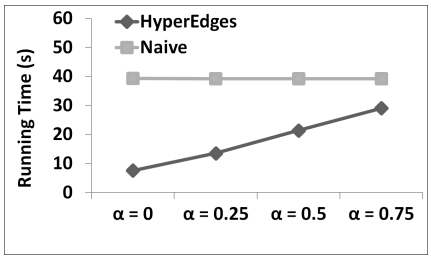


Fig. 36: Effect of α (graph construction)

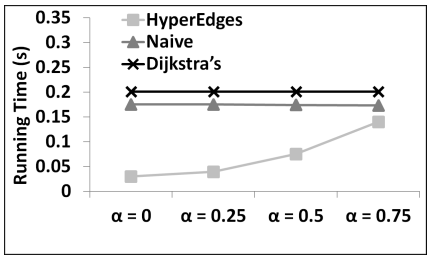


Fig. 37: Effect of α (query processing)

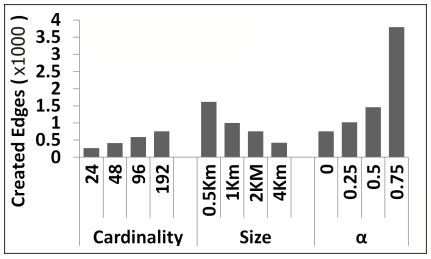


Fig. 38: Number of created edges

of HyperEdges increases as the value of α increases, which is expected because when the cost of traveling within the preferred zones has a higher weight, the labeling technique needs to access more vertices to ensure finding the safest path. However, HyperEdges still outperforms the naive and Dijkstra's algorithms in all cases considered.

Number of created edges: We further measure the number of edges created in the graph precomputed by HyperEdges in the spatial network setting. First, we increase the dataset cardinality from 24 to 192. As Fig. 38 shows, the number of edges increases moderately. Also, as the number of safe zones increases, the average number of edges per safe zone decreases. This is because having more safe zones in the network increases their pruning effectiveness, and hence reduces the number of competing edges among the safe zones. Second, we increase the size of the safe zones. As the same figure shows, the number of created edges decreases. This is natural as larger safe zones need fewer connecting edges. Third, we evaluate the effect of increasing the value of α. Figure 38 shows that the number of created edges increases. This occurs because intermediate preferred zones become less effective in creating better paths as α increases. However, even with the increasing numbers of edges, HyperEdges achieves better performance than the naive and Dijkstra's algorithms, as shown in Figs. 33, 35, and 37.

8 Conclusions and Future Work

We proposed a new path finding problem, safest path via safe zones (SPSZ), which finds the path between two points with the shortest unsafe distance. We modeled the problem in both Euclidean and spatial network settings as a graph shortest path problem and proposed a solution framework, which contains a precomputed graph construction stage and a path finding stage at query time. This framework uses the properties of hyperbolas to prune the search space and reduce the number of edges created. We further solved a generalized version of the problem (SPPZ), where there is a non-zero fractionally weighted cost for travel inside the safe zones (now preferred zones). We propose two techniques to optimize the entry and exit points for preferred zones. As shown by the experimental study, our algorithm outperforms both the naive and the improved naive baseline algorithms in the Euclidean setting in three aspects. First, the number of edges created by our algorithm before filtering is an order of magnitude smaller than that of the improved naive algorithm. This successfully reduces the memory consumption. Second, the time taken for edge creation by our algorithm is an order of magnitude smaller than that of the improved naive algorithm and two orders of magnitude smaller than that of the naive algorithm. Third, our algorithm in query processing is up to an order of magnitude faster than the naive and improved naive algorithms. In addition, the two proposed entry and exit point optimization techniques can reduce the overall path cost in the SPPZ problem by 20%. Similarly, in the spatial network setting, our algorithm consistently outperforms the baseline algorithm in term of the running time and the number of vertices accessed in both graph construction and query processing stages.

This study has put forward a new formulation of regional preference in path finding problems. We have con-

tributed pertinent solutions in both Euclidean and spatial network settings. However, it remains an open challenge to obtain an approximate algorithm for the SPPZ problem with a bounded approximation ratio. It is also of interest to investigate how polygonal preferred zones can be handled. Further, support for a setting where different preferred zones have different α values would further enhance the applicability of the SPPZ problem.

9 Acknowledgment

This work is supported by Australian Research Council (ARC) Discovery Project DP130104587, Australian Research Council (ARC) Future Fellowships Project FT120100832, and partially supported by the National Natural Foundation of China (Nos. 61402155). Saad Aljubayrin is sponsored by Shaqra University, KSA. Jianzhong Qi is supported by University of Melbourne Eearly Career Researcher Grant (project number 603049).

References

1. Ittai Abraham, Daniel Delling, Andrew V Goldberg, and Renato F Werneck. A hub-based labeling algorithm for shortest paths in road networks. In *SEA*, pages 230–241, 2011.
2. Saad Aljubayrin, Jianzhong Qi, Christian S Jensen, Rui Zhang, Zhen He, and Zeyi Wen. The safest path via safe zones. In *ICDE*, pages 531–542, 2015.
3. Hannah Bast, Daniel Delling, Andrew Goldberg, Matthias Müller-Hannemann, Thomas Pajor, Peter Sanders, Dorothea Wagner, and Renato F Werneck. Route planning in transportation networks. *Algorithm Engineering*, 9220, 2016.
4. Jur Berg and Mark Overmars. Planning the shortest safe path amidst unpredictably moving obstacles. In *Algorithmic Foundation of Robotics VII*, pages 103–118, 2008.
5. Scott A Bortoff. Path planning for UAVs. In *American Control Conference*, pages 364–368, 2000.
6. Edgar Dehn. *Algebraic equations: An introduction to the theories of Lagrange and Galois*. Courier Corporation, 2012.
7. Daniel Delling, Andrew V Goldberg, Andreas Nowatzyk, and Renato F Werneck. PHAST: Hardware-accelerated shortest path trees. *Journal of Parallel and Distributed Computing*, 73(7):940–952, 2013.
8. Alexandros Efentakis and Dieter Pfoser. ReHub: Extending hub labels for reverse k-nearest neighbor queries on large-scale networks. *J. Exp. Algorithmics*, 21:1.13:1–1.13:35, 2016.
9. Mohammed Eunus Ali, Rui Zhang, Egemen Tanin, and Lars Kulik. A motion-aware approach to continuous retrieval of 3d objects. In *ICDE*, pages 843–852, 2008.
10. Robert W Floyd. Algorithm 97: Shortest path. *Commun. ACM*, 5(6):345, 1962.
11. Robert Geisberger, Peter Sanders, Dominik Schultes, and Daniel Delling. Contraction hierarchies: Faster and simpler hierarchical routing in road networks. In *SEA*, pages 319–333, 2008.
12. Alfred Gray, Elsa Abbena, and Simon Salamon. *Modern Differential Geometry of Curves and Surfaces with Mathematica*. Chapman and Hall/CRC, 2006.
13. Christina Hallam, KJ Harrison, and JA Ward. A multiobjective optimal path algorithm. *Digital Signal Processing*, 11(2):133–143, 2001.
14. RV Helgason, JL Kennington, and KH Lewis. Shortest path algorithms on grid graphs with applications to strike planning. Technical report, DTIC Document, 1997.
15. HV Jagadish, Beng Chin Ooi, Kian-Lee Tan, Cui Yu, and Rui Zhang. idistance: An adaptive B+-tree based indexing method for nearest neighbor search. *TODS*, 30(2):364–397, 2005.
16. Rahul Kala, Anupam Shukla, and Ritu Tiwari. Fusion of probabilistic A* algorithm and fuzzy inference system for robotic path planning. *Artificial Intelligence Review*, 33(4):307–327, 2010.
17. Nick Koudas, Beng Chin Ooi, Kian-Lee Tan, and Rui Zhang. Approximate NN queries on streams with guaranteed error/performance bounds. In *VLDB*, pages 804–815, 2004.
18. Alain Lambert, S Bouaziz, and R Reynaud. Shortest safe path planning for vehicles. In *Intelligent Vehicles Symposium*, pages 282–286, 2003.
19. Alain Lambert and Dominique Gruyer. Safe path planning in an uncertain-configuration space. *Robotics and Automation*, 3:4185–4190, 2003.
20. Louise Leenen, Alexander Terlunen, and Herman Le Roux. A constraint programming solution for the military unit path finding problem. *Mobile Intelligent Autonomous Systems*, 9(1):225–240, 2012.
21. Chuanwen Li, Yu Gu, Jianzhong Qi, Ge Yu, Rui Zhang, and Qingxu Deng. INSQ: an influential neighbor set based moving knn query processing system. In *ICDE*, pages 1338–1341, 2016.
22. Ying Lu and Cyrus Shahabi. An arc orienteering algorithm to find the most scenic path on a large-scale road network. In *SIGSPATIAL*, pages 46:1–46:10, 2015.
23. Shashi Mittal and Kalyanmoy Deb. Three-dimensional offline path planning for UAVs using multiobjective evolutionary algorithms. *Congress on Evolutionary Computation*, 7(1):3195–3202, 2007.
24. Antonio Miguel Mora, Juan Julian Merelo, Cristian Millan, Juan Torrecillas, Juan Luís Jiménez Laredo, and Pedro A Castillo. Enhancing a MOACO for solving the bi-criteria pathfinding problem for a military unit in a realistic battlefield. In *Applications of Evolutionary Computing*, pages 712–721. 2007.
25. Sarana Nutanong, Rui Zhang, Egemen Tanin, and Lars Kulik. The V*-diagram: a query-dependent approach to moving KNN queries. *PVLDB*, 1(1):1095–1106, 2008.
26. Hanan Samet. The quadtree and related hierarchical data structures. *ACM Comput. Surv.*, 16(2):187–260, 1984.
27. Andy Yuan Xue, Jianzhong Qi, Xing Xie, Rui Zhang, Jin Huang, and Yuan Li. Solving the data sparsity problem in destination prediction. *VLDB J.*, 24(2):219–243, 2015.
28. Andy Yuan Xue, Rui Zhang, Yu Zheng, Xing Xie, Jin Huang, and Zhenghua Xu. Destination prediction by sub-trajectory synthesis and privacy protection against such prediction. In *ICDE*, pages 254–265, 2013.
29. Rui Zhang, Beng Chin Ooi, and Kian-Lee Tan. Making the pyramid technique robust to query types and workloads. In *ICDE*, pages 313–324, 2004.