# Safest Nearby Neighbor Queries in Road Networks

Punam Biswas, Tanzima Hashem, and Muhammad Aamir Cheema

***Abstract*—Safety on the roads has become a major concern in recent days. Travellers prefer to avoid road inconveniences that may occur from crime incidents, street harassment, protests or riots during unrest in a country. To facilitate safe travel, we introduce a novel query for road networks called the $k$ safest nearby neighbor (SNN) query. Given a query location $v_l$, a distance constraint $d_c$ and a point of interest $p_i$, we define the safest path from $v_l$ to $p_i$ as the path with the highest *path safety score* among all the paths from $v_l$ to $p_i$ with length less than $d_c$. The path safety score is computed considering the road safety of each road segment on the path. Given a query location $v_l$, a distance constraint $d_c$ and a set of POIs $P$, a $k$SNN query returns $k$ POIs with the $k$ highest path safety scores in $P$ along with their respective safest paths from the query location. We develop two novel indexing structures called $Ct$-tree and a safety score based Voronoi diagram (SNVD). We propose two efficient query processing algorithms each exploiting one of the proposed indexes to effectively refine the search space using the properties of the index. Our extensive experimental study on real datasets demonstrates that our solution is on average an order of magnitude faster than the baseline.**

***Index Terms*—$Ct$-tree, road networks, safest nearby neighbor, safest path, Voronoi diagram**

## I. Introduction

Crime incidents such as kidnapping and robbery on roads are not unusual especially in developing countries [1]–[3]. Street harassment (e.g., eve teasing, sexual assaults) is a common scenario that mostly women experience on roads [4], [5]. A traveler typically prefers to avoid a road segment with high crime or harassment rate. Similarly, during unrest in a country, people prefer to avoid roads with protests or riots. Also, elderly or sick people may prefer to avoid bumpy roads. However, traditional $k$ nearest neighbors ($k$NN) queries that find $k$ closest points of interest (POI) (e.g., a fuel station or a bus stop) fail to consider a traveler's safety or convenience on roads. In real-world scenarios, a user may prefer visiting a nearby POI instead of their nearest one if the slightly longer path to reach the nearby POI is safer than the shortest path to the nearest POI. In this paper, to allow travelers to avoid different types of inconveniences on roads (e.g., crime incidents, harassment, bumpy roads etc.), we introduce a novel query type, called a *$k$ safest nearby neighbors ($k$SNN) query* and propose novel solutions for efficient query processing.

Intuitively, a path is safer if it requires a smaller distance to be travelled on the least safe roads. Since in real-world scenarios, a user may not want to travel on paths that are longer than a user-defined distance constraint $d_c$, we incorporate $d_c$ in the formulation of the $k$SNN query. Given a set of POIs $P$ on a road network, a query location $v_l$ and a distance constraint $d_c$, a $k$SNN query returns $k$ POIs (along with paths from $v_l$ to them) with the highest Path Safety Score (PSS) considering only the paths with distances less than $d_c$.

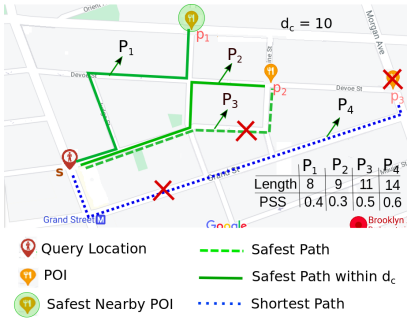



Fig. 1: An example $k$SNN query for $k = 1$ and $d_c = 10$.

Consider Fig. 1 that shows 3 restaurants ($p_1$ to $p_3$), a query located at $s$ and the Path Safety Scores (PSS) of some paths from $s$ to the three POIs (we formally define PSS in Section II). Assume a 1SNN query (i.e., $k = 1$) located at $s$ and $d_c =$ 10km. The restaurants that meet the distance constraint are the candidates for the query answer (i.e., $p_1$ and $p_2$). The POI $p_3$ is not a candidate because the length of the shortest path $P_4$ between $s$ and $p_3$ is 14, which exceeds $d_c$. The safest path between $s$ and $p_1$ is $P_1$. The PSS of $P_1$ is 0.4 and its length is 8km, which is smaller than $d_c = 10km$. Though the safest path $P_3$ between $s$ and $p_2$ has a higher PSS than that of $P_1$, we do not consider $P_3$ because it is longer than $d_c = 10km$. The PSS of the safest path $P_2$ between $s$ and $p_2$ within $d_c$ is 0.3. Thus, 1SNN query (also called SNN query) returns $p_1$ along with the path $P_1$.

Although there are efficient algorithms [6]–[9] for finding the safe paths between two locations, the straightforward application of those algorithms to evaluate $k$SNN queries would require multiple searches and incur prohibitively expensive computations. We consider this straightforward approach as one of the baselines in our experiments and compare the performance of our proposed efficient algorithms against it.

Finding $k$SNNs in a road network is a computational challenge because the number of POIs in the road network and the number of possible paths between a user's location and a POI can be huge. The processing overhead of a $k$SNN algorithm depends on the amount of required road network traversal and the number of POIs considered for finding the safest nearby POIs. In this paper, we develop two novel indexing structures called *Connected Component Tree ($Ct$-tree)* and *safety score based network Voronoi diagram (SNVD)* to refine the search space and propose two efficient algorithms for finding the

P.Biswas ( email: 0417052051@grad.cse.buet.ac.bd) and T.Hashem ( email: tanzimahashem@cse.buet.ac.bd) are with Department of Computer Science and Engineering, Bangladesh University of Engineering and Technology, Dhaka, Bangladesh

M.A. Cheema (e-mail: aamir.cheema@monash.edu) is with the Faculty of Information Technology, Monash University, Melbourne, Australia.

Manuscript received September, 2022

safest nearby POIs with a single search in road networks. Although a number of indexing structures [10]–[16] have been developed, they are developed to reduce the processing overhead for finding nearest neighbors and cannot be applied or trivially extended for efficient processing of $k$SNN queries. We extend index independent incremental network expansion (INE) technique [17] for nearest neighbor search to evaluate $k$SNN queries and consider it as one of the baselines.

A $Ct$-tree recursively partitions the road network graph into connected and safer subgraph(s) by recursively removing the roads with the smallest safety scores. Each node of the $Ct$-tree represents a subgraph, and for every subgraph, the $Ct$-tree stores some distance related information. By exploiting $Ct$-tree properties and the stored distance information, we develop pruning techniques that avoid exploring unnecessary paths that cannot be the safest ones to reach a POI and have distances less than the distance constraint.

In the past, the Voronoi diagram [18] has also been widely used for finding nearest neighbors with reduced processing overhead. The traditional Voronoi diagram divides the road network graph into subgrpahs such that each subgraph corresponds to a single POI which is guaranteed to be the nearest POI of every query location in this subgraph. We introduce SNVD that guarantees that, for every query location in a subgraph, its *unconstrained safest neighbor* (i.e., SNN where the distance constraint is ignored) is its corresponding POI. Note that the unconstrained safest neighbor of a query location is not necessarily its SNN depending on the distance constraint specified by the user. By exploiting the SNVD properties, we develop an algorithm to identify the SNNs, and similar to our $Ct$-tree based approach we improve the performance of our SNVD based algorithm with novel pruning techniques to refine the search space.

We are the first to address the problem of finding $k$SNNs. Our major contributions are as follows:

- We define a PSS measure by incorporating road safety scores and individual distances associated with the road safety scores. Our solution can be easily extended for any other PSS measure that satisfies certain properties.
- We introduce two indexing structures, named $Ct$-tree and SNVD, and develop two novel algorithms for processing $k$SNN queries efficiently using them, respectively.
- We propose novel pruning techniques to refine the search space and further improve the performance.
- We conduct an extensive experimental study using real datasets that demonstrate that the two proposed algorithms significantly outperform the baseline and the INE-based approach.

## II. Problem Formulation

We model a road network as a weighted graph $G=(V,E)$, where $V$ is a set of vertices and $E$ is a set of edges. A vertex $v_i \in V$ represents a road junction in the road network and an edge $e_{ij} \in E$ represents a road between $v_i$ to $v_j$. Each edge $e_{ij}$ is associated with two values $w^d_{ij}$ and $w^{ss}_{ij}$. Here, $w^d_{ij}$ is a positive value representing the weight of the edge $e_{ij}$, e.g., length, travel time, fuel cost etc. For simplicity, hereafter, we use length to refer to $w^d_{ij}$. $w^{ss}_{ij}$ represents the edge safety score (ESS) of $e_{ij}$ (higher the safer). Computing $w^{ss}_{ij}$ is beyond the scope of this paper and we assume that edge safety scores are given as input (e.g., computed using existing geospatial crime mapping approaches [19], traffic conditions, past incidents etc.). Table I summarizes the main notations used the paper.

TABLE I: A list of notations

| Notation | Explanation |
|---|---|
| $G(V,E)$ | The road network graph |
| $e_{ij}$ | An edge connecting vertices $v_i$ and $v_j$ |
| $v_l$ | A query location |
| $d_c$ | A user-defined distance constraint |
| $p_i$ | A POI located at $v_i$ |
| $w^d_{ij}$ | Weight (e.g., length, travel time) of $e_{ij}$ |
| $w^{ss}_{ij}$ | Edge safety score (ESS) of $e_{ij}$ |
| $s^{max}$ | Maximum ESS, i.e., $argmax_{e_{ij}\in E}\ w^{ss}_{ij}$ |
| $pt_{ij}$ | A path from a vertex $v_i$ to a vertex $v_j$ |
| $dist(pt)$ | Sum of weights of edges in $pt$ |
| $pt_{ij}.d_s$ | Total weight of edges in $pt_{ij}$ with ESS equal to $s$ |
| $pss(pt)$ | Path safety score of $pt$ |
| $pt^{sf}_{ij}$ | The safest path between $v_i$ and $v_j$ that has distance less than $d_c$ |
| $pt^{sh}_{ij}$ | The shortest path between $v_i$ and $v_j$ |

A path $pt_{ij}$ between two vertices $v_i$ and $v_j$ is a sequence of vertices such that the path starts at $v_i$, ends at $v_j$, and an edge exists between every two consecutive vertices in the path. Cost of a path $pt_{ij}$ is the sum of the weights of the edges in the path, e.g., total length, total travel time etc. For simplicity, hereafter, we use distance/length to refer to the cost of a path $pt_{ij}$ and denote it as $dist(pt_{ij})$. Let $d_c$ be a user-defined distance constraint. We say that a path $pt_{ij}$ is valid if $dist(pt_{ij}) < d_c$.

### A. Path Safety Score (PSS)

Intuitively, we want to define a Path Safety Score (PSS) measure that ensures that a path that requires smaller distance to be travelled on less safe roads has a higher PSS. Before formally representing this requirement using Property 1, we first define $s$-distance (denoted as $d_s$) of a path which is the total distance a path requires travelling on edges with safety score equal to $s$.

**Definition 1.** *$s$-distance: Given a path $pt_{ij}$ and a positive integer $s$, $s$-distance of the path (denoted as $pt_{ij}.d_s$) is the total length of the edges in $pt_{ij}$ which have safety score equal to $s$, i.e., $pt_{ij}.d_s = \sum_{e_{xy}\in pt_{ij} \wedge w^{ss}_{xy}=s} w^d_{xy}$.*

Consider the example of Fig. 2 that shows three paths $P_1$, $P_2$ and $P_3$ from source $s$ to a POI $p_1$, with lengths 9, 5 and 9, respectively. In Fig. 2, $P_1$ has two edges with ESS equal to 4. The lengths of these edges are 1 and 3, respectively. Therefore, 4-distance of $P_1$ is $P_1.d_4 = 1+3 = 4$. Table II shows $s$-distances for the three paths for each $s$ from 1 to 5, e.g., $P_1.d_1 = 0$, $P_2.d_1 = 1$, $P_3.d_1 = 1$, $P_2.d_2 = 1$, and $P_3.d_2 = 2+2 = 4$.

**Property 1.** *Let $pt_{ij}$ and $pt'_{ij}$ be two valid paths (i.e., have distances less than $d_c$). Let $s$ be the smallest positive integer for which $pt_{ij}.d_s \neq pt'_{ij}.d_s$. The path $pt_{ij}$ must have a higher PSS than $pt'_{ij}$ if and only if $pt_{ij}.d_s < pt'_{ij}.d_s$.*

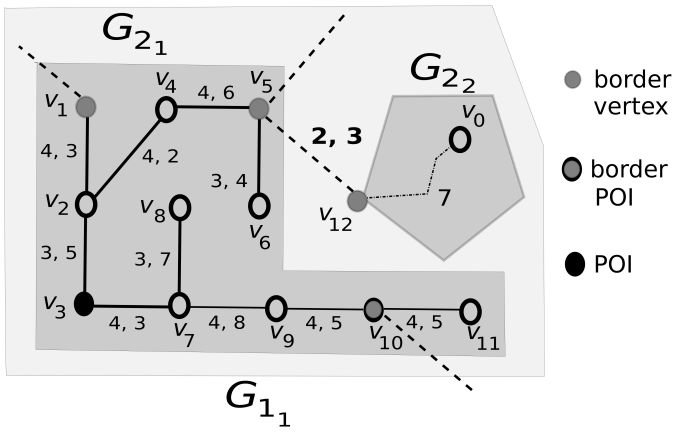



Fig. 2: For each road segment, its ESS and length (in km) are shown as ⟨ESS, length⟩. Assuming $d_c = 10$km, $p_1$ is the 1SNN for query $s$ and $P_1$ is the safest path from $s$ to $p_1$.

TABLE II: Three paths and their $s$-distances

| Path | $P_i.d_s$ ($s$-distance for each path $P_i$) | | | | |
|---|---|---|---|---|---|
| | $P_i.d_1$ | $P_i.d_2$ | $P_i.d_3$ | $P_i.d_4$ | $P_i.d_5$ |
| $P_1$ | 0 | 0 | 0 | 4 | 5 |
| $P_2$ | 1 | 1 | 0 | 1 | 2 |
| $P_3$ | 1 | 4 | 0 | 0 | 4 |

Consider Fig. 2 and Table II. According to Property 1, $P_1$ must have a higher PSS than $P_2$ and $P_3$ because $P_1.d_1 = 0$ is smaller than $P_2.d_1 = P_3.d_1 = 1$, i.e., $P_1$ does not require traveling on an edge with safety score 1 whereas $P_2$ and $P_3$ require traveling 1km on roads with safety score 1. $P_2$ must have a higher PSS than $P_3$ because, although $P_2.d_1 = P_3.d_1$, $P_2.d_2 = 1$ is smaller than $P_3.d_2 = 4$. Hereafter, we use $d_s$ to denote $pt_{ij}.d_s$ wherever the path $pt_{ij}$ is clear by context.

Next, we define a measure of PSS (Definition 2) which satisfies Property 1. For this definition, we assume that, for each edge, $w^d_{ij} \geq 1$ and $w^{ss}_{ij}$ is a positive integer. These assumptions do not limit the applications because $w^d_{ij} \geq 1$ can be achieved by appropriately scaling up if needed and, in most real-world scenarios, safety scores are integer values based on safety ratings (e.g., 1-10).

**Definition 2.** *Path Safety Score (PSS): Let $d_c > 1$ be the distance constraint and $pt_{ij}$ be a valid path (i.e., $dist(pt_{ij}) < d_c$). Let $s^{max}$ be the maximum ESS of any edge in the road network $G$. The PSS $pss(pt_{ij})$ of $pt_{ij}$ is computed as $\frac{1}{\sum_{s=1}^{s^{max}} w_s \times d_s}$, where $w_s = d_c^{(s^{max}-s)}$.*

Here $w_s$ represents the weight (importance) assigned to the edges with ESS equal to $s$. For example, if $d_c = 10$ and $s^{max} = 5$, we have $w_1 = 10^4$, $w_2 = 10^3$, $w_3 = 10^2$, $w_4 = 10^1$ and $w_5 = 10^0$. We calculate PSS for paths $P_1, P_2$ and $P_3$ of Figure 2 as follows. As shown in Table II, $d_s$ values for $P_1$ are $P_1.d_1 = P_1.d_2 = P_1.d_3 = 0$, $P_1.d_4 = 4$ and $P_1.d_5 = 5$. The PSS for path $P_1$ is $\frac{1}{(w_4 \times 4)+(w_5 \times 5)} = \frac{1}{(10 \times 4)+(1 \times 5)} = \frac{1}{45}$. For $P_2$, $d_s$ values are $P_2.d_1 = 1$, $P_2.d_2 = 1$, $P_2.d_3 = 0$, $P_2.d_4 = 1$, $P_2.d_5 = 2$. The PSS for path $P_2$ is $\frac{1}{(w_1 \times 1)+(w_2 \times 1)+(w_4 \times 1)+(w_5 \times 2)} = \frac{1}{(10^4 \times 1)+(10^3 \times 1)+(10 \times 1)+(1 \times 2)} = \frac{1}{11012}$. Finally, $d_s$ values for $P_3$ are $P_3.d_1 = 1$, $P_3.d_2 = 4$, $P_3.d_3 = 0$, $P_3.d_4 = 0$, $P_3.d_5 = 4$. The PSS for $P_3$ is $\frac{1}{(w_1 \times 1)+(w_2 \times 4)+(w_5 \times 4)} = \frac{1}{(10^4 \times 1)+(10^3 \times 4)+(1 \times 4)} = \frac{1}{14004}$. Thus, as required by Property 1, $P_1$ has the highest PSS followed by $P_2$ and then $P_3$.

Note that the importance $w_s$ is set such that it fulfils the following condition: a unit length edge with ESS equal to $s$ contributes more in the sum than all other edges in a valid path with ESS greater than $s$, i.e., $w_s > dist(pt_{ij}) \times w_{s+x}$ for $x \geq 1$ (this is because $dist(pt_{ij}) < d_c$ and $w_{s+x} = w_s / d_c^x$). This condition ensures that our PSS measure satisfies Property 1. For example, when $d_c = 10$, we have $w_1 = 10^4$ and $w_2 = 10^3$. Since each edge has length at least 1, this implies that $w_1 \times d_1 > w_2 \times dist(pt_{ij})$ because length of $pt_{ij}$ is less than $d_c = 10$.

Although the PSS of a path can vary depending on $d_c$, the relative ranking of two paths based on PSSs remains the same irrespective of the value of $d_c$ as long as both paths are valid (i.e., have distance less than $d_c$).

### B. kSNN Query

We denote the safest valid path between two vertices $v_i$ and $v_j$ as $pt^{sf}_{ij}$ which is a path with the highest PSS among all valid paths from $v_i$ to $v_j$. Now, we define $k$SNN query.

**Definition 3.** A $k$ Safest Nearby neighbors ($k$SNN) Query: Given a weighted road network $G(V,E)$, a set of POIs $P$, a query location $v_l$ and a distance constraint $d_c$, a $k$SNN query returns a set $P_k$ containing $k$ POIs along with the safest valid paths from $v_l$ to each of these POIs such that, for every $p_i \in P_k$ and every $p_i' \in P \setminus P_k$, $pss(pt^{sf}_{li}) \geq pss(pt^{sf}_{li'})$ where $pss(pt^{sf}_{li})$ and $pss(pt^{sf}_{li'})$ represent the safest valid paths from $v_l$ to $p_i$ and $p_i'$, respectively.

In Fig. 2, $P_1$ has the highest PSS among all valid paths from $s$ to any of the three POIs, Thus a 1SNN query returns $p_1$ along with the path $P_1$ as the answer. In some cases, there may not be $k$ POIs whose distances from the query location are less than $d_c$. In such scenario, a $k$SNN query returns all POIs with distances less than $d_c$, ranked according to their PSSs. Hereafter, a 1SNN query (i.e., $k = 1$) is also simply referred as a SNN query. Following most of the existing works on POI search, we assume that the POIs and query location lie on the vertices in the graph.

### C. Generalizing the Problem

We remark that our definition of PSS is more general than the definitions in previous works [6]–[8] that do not have any distance constraint and treat each road segment either as safe or unsafe instead of assigning different safety scores to each road as in our work. Consequently, our definition is more general and we can easily use our definition to compute the path that minimizes the distance travelled through unsafe zones (as in [6]–[8]) by assigning each edge in the safe zone a safety score $x$ and each edge in the unsafe zone a safety score $y$ such that $x > y$ and setting $d_c$ to be larger than the length of the longest path in the network.

Although our definition of PSS is already more general than the previous work, in this section, we show that our algorithms

can be immediately applied to a variety of other definitions of PSS. This further generalizes the problem studied in this paper and the proposed solutions. Specifically, we propose two algorithms to solve $k$SNN queries and both algorithms are generic in the sense that they do not only work when PSS is defined using Definition 2 but are also immediately applicable to a variety of other definitions of PSS. Specifically, our *Ct*-tree based algorithm works for any other definition of PSS as long as it satisfies both of the Properties 2 and 3, whereas, our SNVD based algorithm is immediately applicable to any definition of PSS as long as it satisfies Property 2.

**Property 2.** *Let $pt_{xy}$ be a subpath of a valid path $pt_{xz}$. The PSS computed using the defined measure must satisfy $pss(pt_{xz}) \leq pss(pt_{xy})$.*

Property 2 requires that a subpath must have equal or higher PSS than any path that contains this subpath. This requirement is realistic as the risk/inconvenience associated with a path inherits the risk/inconvenience associated with travelling on any subpath of this path.

**Property 3.** *Let $pt_{xy}.minSS$ be the minimum safety score among all edges in the path $pt_{xy}$, i.e., $pt_{xy}.minSS = argmin_{e_{ij} \in pt_{xy}} w_{ij}^{ss}$. Given two valid paths $pt_{xy}$ and $pt'_{xy}$ such that $pt_{xy}.minSS < pt'_{xy}.minSS$, the PSS computed using the defined measure must satisfy $pss(pt_{xy}) < pss(pt'_{xy})$.*

Property 3 is also realistic as it dictates that if the least safe edge on path $P$ is safer than the least safe edge on path $P'$, the PSS of $P$ must be equal to or greater than that of $P'$.

Many intuitive definitions of PSS satisfy the above-mentioned properties. E.g., assume that ESS represents the probability that no crime will occur on this edge (as in [20]). If PSS is defined as multiplication of ESS on all edges on a path [20] (i.e., probability that no crime will occur on the whole path), this definition of PSS satisfies Property 2 but not Property 3. Thus, SNVD based algorithm can be used to handle queries involving such PSS. On the other hand, if PSS corresponds to the minimum ESS on any edge on the path, then this definition of PSS satisfies both properties, therefore, both Ct-tree and SNVD based approaches can be used.

We remark that our PSS measure (Definition 2) satisfies both properties and thus, both of our algorithms can be used for it. For example, in Fig. 2, the PSS for path $P_1$ is $\frac{1}{(w_4 \times 4)+(w_5 \times 5)} = \frac{1}{(10\times4)+(1\times5)} = \frac{1}{45}$. Now if we consider the subpath of $P_1$ by removing the last edge with ESS 5 and length 2, the PSS increases to $\frac{1}{(w_4 \times 4)+(w_5 \times 3)} = \frac{1}{(10\times4)+(1\times3)} = \frac{1}{43}$ (which satisfies Property 2). Similarly, $P_1.minSS = 4$ and $P_2.minSS = 1$, and the PSSs of $P_1$ and $P_2$ are $\frac{1}{45}$ and $\frac{1}{11012}$, respectively (which satisfies Property 3).

For the ease of presentation, in our problem setting, we assume that each edge has a single safety score $w_{ij}^{ss}$. However, if an edge has multiple safety scores representing the road safety conditions at different times of the day (e.g., day vs night) or different types of crimes, our solutions can be immediately applied by only considering the relative safety score for each edge (e.g., safety scores for night time if the query is issued at night).

## III. Related Work

A variety of route planning problems have been studied in the past such as: *shortest route computation* [13], [21] which requires to find the path with the smallest overall cost; *multi-criteria route planning* [22], [23] which aims to return routes considering multiple criteria (e.g., length, safety, scenery etc.); *obstacle avoiding path planning* [24]–[28] which returns the shortest path avoiding a set of obstacles in the space; *safe path planning* [6]–[8] that aims to return paths that are safe and short; *alternative route planning* [22], [29]–[31] where the goal is to return a set of routes significantly different from each other so that the user has more options to choose from; and *multi-stop route planning* (also called trip planning) [23], [32]–[34] which returns a route that passes through multiple stops/POIs satisfying certain constraints. Below, we briefly discuss some of these route planning problems most closely related to our work.

**Multi-criteria route planning.** Existing techniques [22], [23], [35], [36] typically use a scoring function (e.g., weighted sum) to compute a single score of each edge considering multiple criteria. Once the score of each edge has been computed, the existing shortest path algorithms can be used to compute the path with the best score. In contrast, our work cannot trivially use the existing shortest path algorithms due to the nature of the PSS. Also, as noted in [20], it is non-trivial for a user to define an appropriate scoring function combining the multiple criteria. Thus, some existing works [20], [37] approach the multi-criteria route planning differently and compute a set of *skyline routes* which guarantees that the routes returned are not *dominated* by any other routes. This is significantly different from our work as it returns possibly a large number of routes instead of the safest route.

**Safe path planning.** There are several existing works that consider safety in path planning. In [6]–[8], the authors divide the space into safe and unsafe zones and minimize the distance travelled through the unsafe zones. However, these works are unable to handle different safety levels of roads and do not consider any distance constraint on the paths. In Section II, we show that our work is more general and is applicable to a wider range of definitions of safe paths. Some other existing works [20], [35], [36] also consider safety, however, they model the problem as multi-criteria route planning and are significantly different from the problem studied in this paper (as discussed above). A couple of recent approaches [38], [39] consider minimizing the total risk scores or maximizing the total safety score of a path. Optimizing the total scores does not allow a path to exclude a high risk road and thus, cannot ensure a traveler's safety. The most closely related work to our work is [9] which develops an algorithm to find the safest paths between a source and a destination by considering individual distances associated with different safety scores of a path, which is similar to the definition of our path safety score.

The problem studied in this paper is significantly different from the above-mentioned route planning problems. Unlike route planning problems, our problem does not have a fixed destination, (i.e., each POI is a possible destination) and the goal is to find $k$ POIs with the safest paths. In other words, the

problem studied in this work is a POI search problem similar to a $k$ nearest neighbor ($k$NN) query. We use [9] in one of our baselines to find the safest paths to candidate $k$SNNs.

**$k$NN Queries.** Answering $k$NN queries has been extensively studied, e.g., see [40] which describes and evaluates some of the most notable $k$NN query processing algorithms on road networks. A straightforward application of a existing $k$NN algorithms to $k$SNNs is prohibitively expensive as it would require two independent traversals of the road network for first finding $k' > k$ nearest neighbors as candidate $k$SNNs, and then computing the safest paths from the query location to each of the candidate $k$SNN, respectively. We extend the index independent INE [17] technique for $k$NN search to identify $k$SNNs with a single search in the road network and consider it as a baseline (please see Section IV for details).

## IV. Incremental Network Expansion

Incremental network expansion (INE) [17] is among the most efficient nearest neighbour search techniques that do not rely on any distance-based indexing structure. Since the PSS of a path is no larger than that of its subpath (Property 2), we can adapt the INE search to find $k$SNNs.

Starting from the query location $v_l$, the INE based search explores the adjacent edges of $v_l$. For each edge $e_{li}$, a path consisting of $v_l$ and $v_i$ is enqueued into a priority queue $Q_p$. The entries in $Q_p$ are ordered in the descending order based on their PSSs. Then the search continues by dequeueing a path from $Q_p$ and repeating the process by exploring adjacent edges of the last vertex of the dequeued path. Before enqueueing a new valid path $p_{lj}$ into $Q_p$, its PSS is incrementally computed from the PSS of the dequeued path $p_{li}$ and the PSS of the path $p_{ij}$ that consists of a single edge $e_{ij}$ using the following lemma (the proof of the lemma is included in the full version[1] [41] of this paper).

**Lemma 1.** *Let $pt_{lj}$ be a valid path such that $pt_{lj} = pt_{li} \oplus pt_{ij}$ where $\oplus$ is a concatenation operation. Then, $pss(pt_{lj}) = \frac{1}{\frac{1}{pss(pt_{li})} + \frac{1}{pss(pt_{ij})}}$.*

The first SNN is identified once the last vertex of a dequeued path from $Q_p$ is a POI and the distance of the path is less than $d_c$. The search for $k$SNNs terminates when paths to $k$ distinct POIs have been dequeued from $Q_p$ and the distances of the paths are smaller than $d_c$.

To refine the search space, we check if a path can be pruned before enqueueing it into $Q_p$ using the following pruning rules. Pruning Rule 1 is straightforward as it simply uses the distance constraint for the pruning condition.

**Pruning Rule 1.** *A path $pt_{lj}$ can be pruned if $dist(pt_{li}) \geq d_c$, where $d_c$ represents the distance constraint.*

A path $pt_{lj}$ between $v_l$ and $v_j$ can be pruned if there is already a dequeued path $pt'_{lj}$ which is at least as short as $pt_{lj}$ *and* at least as safe as $pt_{lj}$. The following lemma justifies the above intuition (the proof of the lemma is included in the full version [41] of this paper).

[1]The full version includes detailed descriptions (e.g., proof of the lemmas, complexity analysis) and more results of our experiments.

**Lemma 2.** *If path $pt'_{lj}$ is at least as short as path $pt_{lj}$ and at least as safe as $pt_{lj}$, then path $pt'_{lk} = pt'_{lj} \oplus pt_{jk}$ is at least as short as and at least as safe as path $pt_{lk} = pt_{lj} \oplus pt_{jk}$.*

Since the search dequeues paths in descending order of PSSs from $Q_p$ and Pruning Rule 2 is applied to a path $pt_{lj}$ before enqueueing it, the PSS of any dequeued path is higher than $pss(pt_{lj})$. Thus, Pruning Rule 2 only checks whether there is a dequeued path to $v_j$ that is at least as short as $pt_{lj}$.

**Pruning Rule 2.** *A path $pt_{lj}$ can be pruned if $dist(pt_{lj}) \geq D_{sh}[j]$, where $D_{sh}[j]$ represents the distance of the shortest path $pt'_{lj}$ from $v_l$ to $v_j$ dequeued so far.*

We remark that, since $D_{sh}[j]$ is less than $d_c$ when a valid dequeued path to $v_j$ exists, Pruning Rule 2 facilitates the pruning of the valid paths (i.e., the path length is smaller than $d_c$). On the other hand, Pruning Rule 1 prunes the invalid paths, when there is no existing dequeued path that ends at $v_j$, i.e., $D_{sh}[j] = \infty$. We provide complexity analysis of the algorithm in the full version [41].

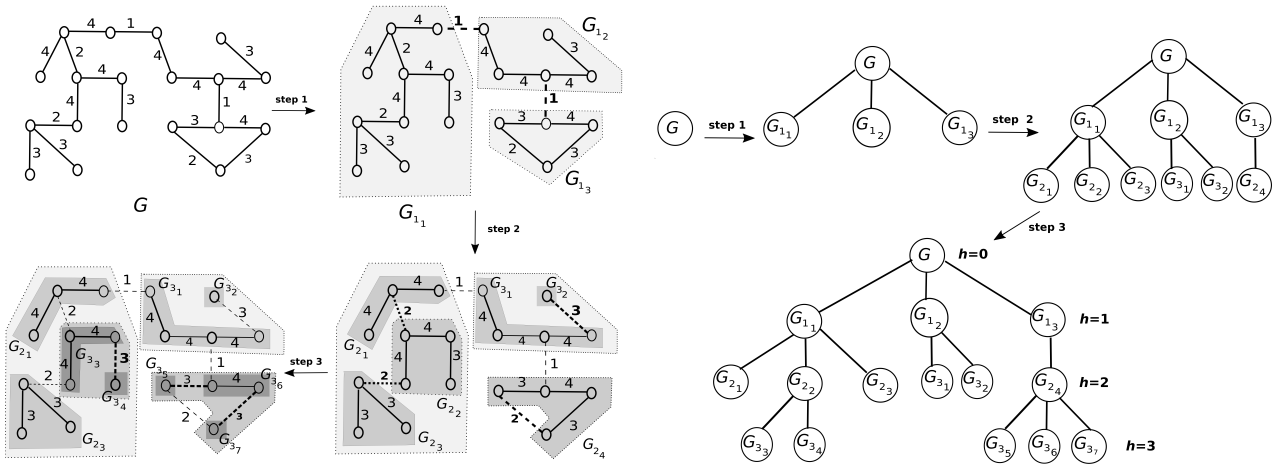
## V. Ct-tree

In this section, we introduce a novel indexing structure, **C**onnected component tree (*Ct*-tree) and develop an efficient solution based on *Ct*-tree to process $k$SNN queries. Index structures like *R*-tree [10], Contraction Hierarchy (CH) [42], Quad tree [43], ROAD [12], *G*-tree [13], $G^*$-tree [14], *LG*-tree [15] and *HN*-tree [16] have been proposed for efficient search of the query answer based on the distance metric and are not applicable for $k$SNN queries. Some of these indexing techniques [10], [43] divide the space into smaller regions based on the position of the POIs, whereas some other indexing techniques [12], [42] divide the space based on the properties of the road network graph. These existing indexing techniques cannot be applied or trivially extended to compute $k$SNNs because they do not incorporate Edge Safety Score (ESS)s of the edges in the road network graph.

### A. Ct-Tree Construction and Properties

The key idea to construct a *Ct*-tree is to recursively partition the graph by removing the edges with the smallest ESSs in each step. Removing the edges with the smallest ESS $s$ may partition the graph into one or more connected components. A component is denoted as $G_{s_t}$, where $t$ is a unique identifier for the partitions created by removing edges with ESS $s$. Each connected component $G_{s_t}$ is recursively partitioned by removing the edges with the smallest ESS within $G_{s_t}$. The recursive partitioning stops when $G_{s_t}$ either contains a single vertex or all edges within $G_{s_t}$ have the same ESS.

Without loss of generality, we explain the *Ct*-tree construction process using an example shown in Figure 3. The original graph has edge ESSs in the range of 1 to 4, and for the sake of simplicity, we do not show the edge distances in the figure. The root of the *Ct*-tree represents the original graph $G$. After removing the smallest edge ESS 1, $G$ is divided into three connected components $G_{1_1}$, $G_{1_2}$ and $G_{1_3}$. These connected components are represented by three child nodes of the root




(a) Ct-tree subgraphs (b) Ct-tree

Fig. 3: Ct-tree Construction Steps

node at tree height $h = 1$. Each of these components is then recursively partitioned by removing the edges with the smallest ESS. For example, the edges with ESS 2 are removed from $G_{1_1}$, and $G_{1_1}$ is divided into three connected components $G_{2_1}$, $G_{2_2}$ and $G_{2_3}$. The recursive partitioning of $G_{2_1}$ and $G_{2_3}$ stop as they have edges with same ESS. On the other hand, after removing the edges with ESS 3, $G_{2_2}$ is divided into $G_{3_3}$ and $G_{3_4}$. We formally define a *Ct*-tree as follows:

**Definition 4.** ***Ct-tree:*** *A Ct-tree C is a connected component based search tree, a hierarchical structure that has the following properties:*

- *The Ct-tree root node represents the original graph G.*
- *Each internal or leaf Ct-tree node represents a connected component $G_{s_t}$, where $G_{s_t}$ does not include any edge with ESS smaller than or equal to s and $G_{s_t}$ is included in the graph represented by its parent node.*
- *The maximum height of the tree, $h^{max} = s^{max} - 1$*
- *Each internal or leaf Ct-tree node maintains the following information: the number of POIs $n(G_{s_t})$ in $G_{s_t}$, a border vertex set $B_{s_t}$, the minimum border distance $d_B^{min}(v_x, G_{s_t})$ and the minimum POI distance $d_p^{min}(v_x, G_{s_t})$ for each border vertex $v_x \in B_{s_t}.s$*

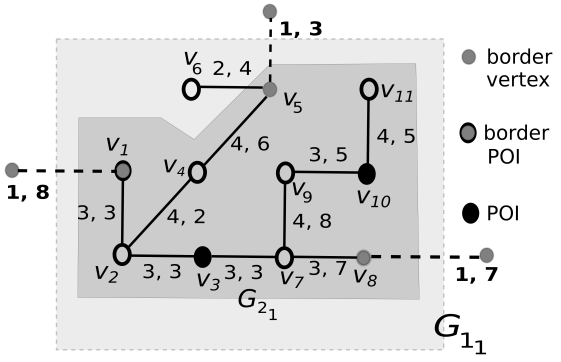



Fig. 4: Border vertices, the minimum border distance and the minimum POI distance

***Border vertices, the minimum border distance and the minimum POI distance.*** A vertex $v_x$ is called a border vertex of a subgraph $G_{s_t}$, if there is an outgoing edge from $v_x$ whose ESS is smaller than or equal to $s$ and the edge is not included in $G_{s_t}$. We denote the set of border vertices of $G_{s_t}$ with $B_{s_t}$. For example, in Figure 4, $B_{1_2}$ includes $\{v_1, v_5, v_8\}$. A border vertex of $G_{s_t}$ represented by a *Ct*-tree node is also a border vertex of the subgraphs represented by its descendent nodes. For example, $v_5$ is a border vertex of both $G_{1_2}$ and $G_{2_1}$.

For each border vertex, the corresponding *Ct*-tree node stores the minimum border distance and the minimum POI distance. The minimum border distance $d_B^{min}(v_x, G_{s_t})$ of a border vertex $v_x$ of $G_{s_t}$ is defined as the minimum of the distances of the shortest paths from $v_x$ to $v_y$ for $v_y \in B_{s_t} \setminus v_x$. In Figure 4, the distances of border vertex $v_8$ from other border vertices $v_1$ and $v_5$ are 16 and 21 respectively. Thus, $d_B^{min}(v_8, G_{1_1}) = 16$.

The minimum POI distance $d_p^{min}(v_x, G_{s_t})$ of a border vertex $v_x$ is defined as is the distance from $v_x$ to its closest POI in $G_{s_t}$. In Figure 4, the distances of border vertex $v_8$ from POIs $v_1$, $v_3$ and $v_{10}$ are 16, 10 and 20 respectively. Thus, $d_p^{min}(v_8, G_{1_1}) = 10$.

After the *Ct*-tree construction, for each vertex $v$ in $G$, we store a pointer to each *Ct*-tree node whose subgraph contains $v$. Since the height is $s^{max} - 1$, this requires adding at most $O(s^{max})$ pointers for each $v$. We remark that the height $h$ of the Ct-tree can be controlled if needed. Specifically, to ensure a height $h$, the domain of possible ESS values is divided in $h$ contiguous intervals and, in each iteration, the edges with ESS in the next smallest interval are removed. E.g., if ESS domain is 1 to 10, a Ct-tree of height 5 can be constructed by first removing edges with ESS in range $(0,2]$ and then $(2,4], (4,6]$ and $(8,10]$ (in this order).

### *B. Query Processing*

*1) kSNN search:* The efficiency of any approach for evaluating a *k*SNN query depends on the area of the graph search space for finding the safest paths having distances less than $d_c$ from $v_l$ to the POIs and the number of POIs considered for identifying *k*SNNs. The *Ct*-tree structure and Property 3 of our PSS measure allow us to start the search from the smallest and safest road network subgraph that has the possibility to include

$k$SNNs for $v_l$. By construction of $Ct$-tree, it is guaranteed that the subgraph of a child node is smaller and safer than that of its parent node. Thus, starting from the root node, our approach recursively traverses the child nodes that include $v_l$. The traversal ends once a child node that includes less than $k$ POIs and $v_l$ is reached. The parent of the last traversed child node is selected as the starting node of our $k$SNN search. Note that the subgraph $G_{s_t}$ of the starting node includes greater than or equal to $k$ POIs.

If the lengths of the paths from $v_l$ to at least $k$ POIs in $G_{s_t}$ is smaller than $d_c$, then our approach does not need to expand $G_{s_t}$. This is because, by definition, the edges that connect the border vertices in $G_{s_t}$ to other vertices that are not in $G_{s_t}$ have lower ESSs than those of the edges in $G_{s_t}$. If there are less than $k$ POIs in $G_{s_t}$ whose safest paths from $v_l$ have distances less than $d_c$, our $Ct$-tree based approach recursively updates $G_{s_t}$ with the subgragh of its parent node until $k$SNNs are identified.

To find the safest path having distance less than $d_c$ from $v_l$ to the POIs in $G_{s_t}$, we improve the INE based safest path search discussed in Section IV by incorporating novel pruning techniques using $Ct$-tree properties. Specifically, the minimum border distance and the minimum POI distance stored in the $Ct$-tree node allow us to develop Pruning Rules 3 and 4 to further refine the search space in $G_{s_t}$.

**Pruning Rule 3.** *A path $pt_{lj}$ can be pruned if $dist(pt_{lj}) + d_B^{min}(v_j, G_{s_t}) \geq d_c$ and $dist(pt_{lj}) + d_p^{min}(v_j, G_{s_t}) \geq d_c$, where $v_j$ is a border vertex of $G_{s_t}$ and $d_B^{min}(v_j, G_{s_t})$ and $d_p^{min}(v_j, G_{s_t})$ are the minimum border distance and the minimum POI distance of $v_j$, respectively.*

**Pruning Rule 4.** *A path $pt_{lj}$ can be pruned if $dist(pt_{lj}) + d_B^{min}(v_j, G_{s_t}) \geq d_c$ and $dist(pt_{lj}) + d_p^{min}(v_j, G_{s_t}) \geq maxD$, where $v_j$ is a border vertex of $G_{s_t}$, $P_{s_t}$ represents the set of POIs in $G_{s_t}$, and maxD represents the maximum of the current shortest distances of the POIs in $P_{s_t}$ from $v_l$, i.e., $maxD = \max_{p_i \in P_{s_t}} D_{sh}[i]$.*

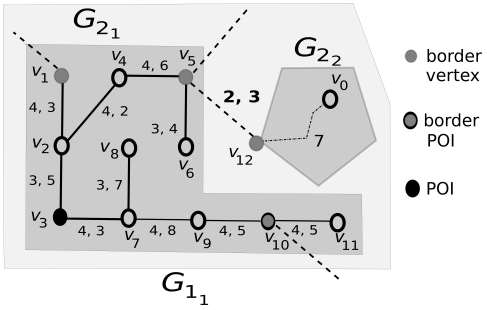



Fig. 5: $pt_{05}$ can be pruned using Pruning Rule 3 for a query location $v_0$ and $d_c = 20$

If the first condition that uses the minimum border distance in Pruning Rule 3 or 4 becomes true then it is guaranteed that the expanded path through $pt_{lj}$ cannot cross $G_{s_t}$ to reach a POI outside of $G_{s_t}$ due to the violation of distance constraint. On the other hand, satisfying the second condition in Pruning Rule 3 means that the expanded path through $pt_{lj}$ cannot reach a POI inside $G_{s_t}$ due to the violation of distance constraint. For the second condition, Pruning Rule 4 exploits that if the POIs inside $G_{s_t}$ are already reached using other paths then $maxD$, the maximum of the current shortest distances of the POIs in $G_{s_t}$ can be used to prune a path. If the second condition in Pruning Rule 4 becomes true then it is guaranteed that the expanded path through $pt_{lj}$ cannot provide paths that are safer than those already identified for the POIs in $G_{s_t}$ (please see Pruning Rule 2 for details).

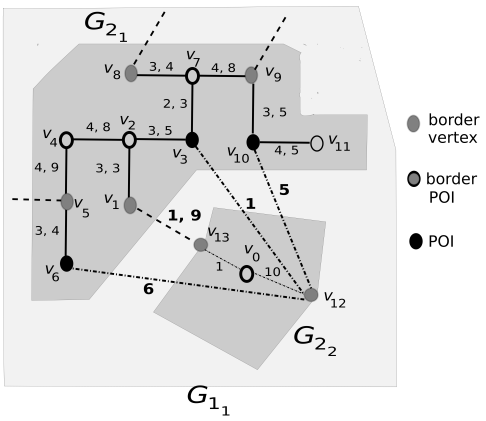



Fig. 6: $pt_{01}$ can be pruned using Pruning Rule 4 for a query location $v_0$ and $d_c = 20$

Pruning Rule 4 can prune more paths than Pruning Rule 3 when $maxD$ is less than $d_c$, i.e., at least one path from $v_l$ to every POI in $G_{s_t}$ have been identified. On the other hand, Pruning Rule 3 is better than Pruning Rule 4 when $maxD$ is $\infty$, i.e., no path has yet been identified for a POI in $G_{s_t}$. Hence we consider all pruning rules (Pruning Rule 1–Pruning Rule 4) to check whether a path can be pruned.

Figure 5 shows an example where a path $pt_{05}$ is pruned using Pruning Rule 3 for a query location $v_0$ and $d_c = 20$. In the example, $v_5$ is a border vertex of $G_{2_1}$, $dist(pt_{05}) = 10$, $d_B^{min}(v_5, G_{2_1}) = 11$ and $d_p^{min}(v_5, G_{2_1}) = 13$. Here, both $dist(pt_{05}) + d_B^{min}(v_5, G_{2_1})$ and $dist(pt_{05}) + d_p^{min}(v_5, G_{2_1})$ are greater than $d_c$. Hence according to Pruning Rule 3, path $pt_{05}$ can be pruned. From the figure we also observe that if we expand $pt_{05}$, it cannot reach a POI outside $G_{2_1}$ through a border vertex ($v_1$ or $v_{10}$) or a POI ($v_3$ or $v_{10}$) in $G_{2_1}$ due to the violation of the distance constraint.

In Figure 6, the POIs ($v_3$, $v_6$ and $v_{10}$) in $G_{2_1}$ are already reached using other paths and $D_{sh}[3] = 11$, $D_{sh}[6] = 16$ and $D_{sh}[10] = 15$. Thus $maxD = 16$. The query location is $v_0$, $d_c = 20$, $dist(pt_{01}) = 10$, $v_1$ is a border vertex of $G_{2_1}$, $d_B^{min}(v_1, G_{2_1}) = 15$ and $d_p^{min}(v_1, G_{2_1}) = 8$. Here, $dist(pt_{01}) + d_B^{min}(v_1, G_{2_1})$ is greater than $d_c$ and $dist(pt_{01}) + d_p^{min}(v_1, G_{2_1})$ is greater than $maxD$ but not $d_c$. Thus, in this case path $pt_{01}$ is pruned using Pruning Rule 4.

*2) Algorithm:* Algorithm 1, Ct-$k$SNN, shows the pseudocode to find $k$SNNs in the road network using the $Ct$-tree.

The algorithm uses two priority queues $Q_{cur}$ and $Q_{next}$, where $Q_{cur}$ is used for the path expansion in $G_{s_t}$, and $Q_{next}$ stores the paths that will be expanded WHILE searching the parent subgraph of $G_{s_t}$. Each entry of these queues represents a path, the PSS and the path's length. The entries in a queue are ordered in descending order based on their PSSs. The

**Algorithm 1** *Ct*-tree-*k*SNN($v_l, k, d_c$)

1: $Initialize(Q_{cur}, Q_{next}, D_{sh})$
2: $G_{s_t} \leftarrow Contains(v_l)$
3: $Enqueue(Q_{cur}, v_l, \infty, 0)$
4: **while** $Q_{cur} != \emptyset$ **do**
5: $pt_{li}, pss(pt_{li}), dist(pt_{li}) \leftarrow Dequeue(Q_{cur})$
6: **if** $dist(pt_{li}) < D_{sh}[i]$ **then**
7: $D_{sh}[i] \leftarrow dist(pt_{li})$
8: **end if**
9: **if** $isPOI(v_i)$ and $!Include(A, v_i)$ **then**
10: $A \leftarrow A \cup \{pt_{li}\}$
11: **if** $A.size() == k$ **then**
12: **return** $A$
13: **end if**
14: **end if**
15: **if** $v_i \in B_{s_t}$ **then**
16: $Enqueue(Q_{next}, pt_{li}, pss(pt_{li}), dist(pt_{li}))$
17: **end if**
18: **for** $e_{ij} \in E_{s_t}$ **do**
19: **if** $!PrunePath(pt_{lj}, d_c, D_{sh}, G_{s_t})$ **then**
20: $Enqueue(Q_{cur}, pt_{lj}, pss(pt_{lj}), dist(pt_{lj}))$
21: **end if**
22: **end for**
23: **if** $Q_{cur} = \emptyset$ **then**
24: $G_{s_t} \leftarrow GetParent(G_{s_t})$
25: $Q_{cur} \leftarrow Q_{next}$
26: $Q_{next} \leftarrow \emptyset$
27: **end if**
28: **end while**
29: **return** $A$

algorithm uses an array $D_{sh}$, where $D_{sh}[i]$ stores the distance of the shortest dequeued path from $v_l$ to $v_i$.

The algorithm starts with initializing two priority queues $Q_{cur}$ and $Q_{next}$ to $\emptyset$ and $D_{sh}[i]$ for each $v_i$ to $d_c$ using Function *Initialize* (Line 1). Then, starting from the root node, Function *Contains* recursively traverses the child nodes using the stored pointers of $v_l$ until it identifies the smallest subgraph $G_{s_t}$ of a *Ct*-tree node that includes $v_l$ and has at least $k$ POIs (Line 2). The $v_l$ and the corresponding path information are enqueued to $Q_{cur}$. The algorithm iteratively processes the entries in $Q_{cur}$ until it becomes empty or $k$SNNs are found (Lines 4–28).

In every iteration, the algorithm dequeues a path $pt_{li}$ from $Q_{cur}$ and updates $D_{sh}[i]$ of the last vertex $v_i$ of the dequeued path if $dist(pt_{li}) < D_{sh}[i]$ (Lines 5–8). If $v_i$ represents a POI that is not already present in $A$, then $pt_{li}$ is added to $A$, and if $A$ includes $k$ entries, the answer is returned (Lines 9–14).

If $v_i$ is a border vertex of $G_{s_t}$, then there is at least one outgoing edge from $v_i$ with safety score smaller than or equal to $s$, which might need to be later considered if $k$SNNs are not found in current $G_{s_t}$. Thus, $pt_{li}$ and the corresponding path information are enqueued to $Q_{next}$ (Lines 15–17). Note that the ESS of $G_{s_t}$ are larger than $s$.

Next, for each outgoing edge $e_{ij}$ of $v_i$ in $G_{s_t}$, the algorithm checks whether the newly formed path $pt_{lj}$ by adding $e_{ij}$ at the end of $pt_{li}$ can be pruned using *PrunePath* function (*PrunePath* is elaborated below). If the path is not pruned, then $pt_{lj}$ and the corresponding path information are enqueued to $Q_{cur}$ (Lines 18–22).

At the end of the iteration, the algorithm checks whether the exploration of $G_{s_t}$ is complete, i.e., $Q_{cur}$ is empty. If this condition is true, then it means that the safest paths having distances less than $d_c$ from $v_l$ to $k$SNNs are not included in $G_{s_t}$. Thus, the algorithm sets the parent of $G_{s_t}$ as $G_{s_t}$, assigns $Q_{next}$ to $Q_{cur}$ and resets $Q_{next}$ to $\emptyset$ (Lines 23-27).

***PrunePath***. Algorithm 2 returns *true* if path $pt_{ij}$ can be pruned by any of our pruning criteria and $false$ otherwise. It first checks whether $pt_{ij}$ can be pruned using the criteria in Pruning Rules 1 or 2. Note that an entry $D_{sh}[x]$ for a vertex $v_x$ is initialized to $d_c$ and later it gets updated once a path to $v_x$ is dequeued from the queue (see Line 7 in Algorithm 1).

If $pt_{ij}$ is not pruned, Function $checkBorder(G_{s_t}, v_j)$ checks whether $v_j$ is a border vertex of $G_{s_t}$ or one of its descendants. If so, it returns *true* for *IsBorder* and the node $G_{s'_{t'}}$ for which $v_j$ is a border vertex. Otherwise, *IsBorder* is set to $false$. By construction of the *Ct*-tree, a border vertex of a *Ct*-tree node is also the border vertex of its descendants. Thus, there may be multiple nodes for which $v_j$ is a border vertex, in which case, $G_{s'_{t'}}$ is chosen to be the highest node in the *Ct*-tree for which $v_j$ is a border vertex.

**Algorithm 2** $PrunePath(pt_{lj}, d_c, D_{sh}, G_{s_t})$

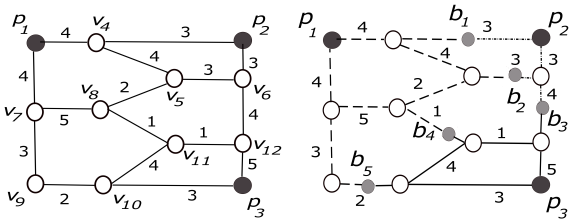
1: **if** $dist(pt_{lj}) \geq D_{sh}[j]$ **then**
2: **return** true
3: **end if**
4: $isBorder G_{s'_{t'}} \leftarrow checkBorder(G_{s_t}, v_j)$
5: **if** $isBorder == true$ and $dist(pt_{lj}) + d_B^{min}(v_j, G_{s'_{t'}}) \geq d_c$ **then**
6: **if** $dist(pt_{lj}) + d_p^{min}(v_j, G_{s'_{t'}}) \geq maxD$ **then**
7: **return** true
8: **end if**
9: **end if**
10: **return** false

If $v_j$ is a border vertex then the algorithm checks whether $pt_{ij}$ can be pruned using Pruning Rules 3 or 4 (Lines 5–8). One of the pruning conditions that uses the minimum border distance is the same in both Pruning Rules 3 and 4, which is checked in Line 5. The left part of the other condition, adding the minimum POI distance with $dist(pt_{ij})$ is also same in both pruning rules. The right part is $d_c$ for Pruning Rule 3 and *maxD* for Pruning Rule 4, where *maxD* represents the maximum of the current shortest distances of $v_l$ to the POIs in $G_{s'_{t'}}$. The shortest distance of every vertex (including POIs) is initialized to $d_c$ (Line 1 of Algorithm 1). Thus, *maxD* is initially $d_c$ and later may become less than $d_c$ when the paths to POIs in $G_{s'_{t'}}$ are dequeued from the priority queue (Lines 5–7 of Algorithm 1). Since the minimum border distance and minimum POI distance of a *Ct*-tree node are at least equal to those of its descendants, we do not need to check these pruning conditions for $v_j$ for the descendants of $G_{s'_{t'}}$ separately.

In full version [41] of this paper, we provide complexity analysis of the algorithm and discuss how to update *Ct*-tree when there are updates to the road network and POIs.

## VI. Safety Score Based Network Voronoi Diagram

Network Voronoi diagram (NVD) [44], [45] has been shown as an effective method for faster processing of $k$NN queries [18], [46]. Traditional NVD is computed using distances and does not apply for processing $k$SNN queries. In Section VI-A, we present *safety score based network Voronoi diagram (SNVD)* and its properties. Our efficient technique to compute the $k$SNNs is composed of two phases: preprocessing (Section VI-B) and query processing (Section VI-C).




(a) Original Graph (b) SNVD

Fig. 7: An example of SNVD

### A. SNVD Properties

An SNVD divides the road network graph into subgraphs such that each subgraph contains a single POI which is the *unconstrained safest neighbor* for every point of the subgraph. We differentiate the unconstrained safest neighbor from the safest nearby neighbor (SNN) by considering *unconstrained safest path*. For an unconstrained safest path, we assume a sufficiently large value for $d_c$ such that every simple path in the graph has length less than $d_c$, e.g., $d_c$ is just bigger than the sum of the lengths of all edges in the graph. The subgraphs are called Voronoi cells and a point that defines the boundary between two adjacent Voronoi cells is a called a border vertex. The Voronoi cell of POI $p_i$ is denoted as $VC_i$ and the border vertex set for $VC_i$ is denoted as $B_i$.

Figure 7 shows an example of an SNVD. Black vertices represent the generator POIs $p_1$, $p_2$ and $p_3$, white vertices represent the road intersections, and grey vertices represent the border vertices.

Unlike traditional distance based NVD [46], [47], a border vertex $b$ between two adjacent Voronoi cells $VC_i$ and $VC_j$ may not have equal PSS for its unconstrained safest paths to $p_i$ and $p_j$. Figure 8 shows examples of such scenarios. When the PSSs for $b$'s unconstrained safest paths to $p_i$ and $p_j$ differ, the unconstrained safest neighbor of $b$ is one of the POIs $p_i$ and $p_j$ for which $b$ has the higher PSS.

The following lemmas shows the properties of SNVD that we exploit in our solution.

**Lemma 3.** *Let $p_1, p_2, \ldots, p_{k-1}$ represent the $k-1$ unconstrained safest POIs of $v_l$, respectively. The unconstrained safest path from $v_l$ to its $k^{th}$ unconstrained safest POI $p_k$ can only go through $\{VC_1, VC_2, \ldots, VC_{k-1}, VC_k\}$.*

*Proof.* Assume that the unconstrained safest path from $v_l$ to $p_k$ goes through a vertex $v$ of $VC_t$, where $t \notin \{1,2,\ldots,k\}$. By construction of SNVD, POI $p_t$ is safer than POI $p_k$ for $v$. Since the unconstrained safest path from $v_l$ to $p_k$ goes through $v$, POI $p_t$ is also safer than POI $p_k$ for $v_l$. Thus $t \in \{1,2,\ldots,k\}$, which contradicts our assumption. □

**Lemma 4.** *Let $v_1, v_2, \ldots, v_{k-1}$ represent the $k-1$ unconstrained safest POIs of $v_l$, respectively. The $k^{th}$ unconstrained safest POI $p_k$ of $v_l$ lies in one of the adjacent Voronoi cells of $VC_1, VC_2, \ldots, VC_{k-1}$.*

*Proof.* If the $k^{th}$ unconstrained safest POI $v_k$ of $v_l$ does not lie in one of the adjacent Voronoi cells of $VC_1, VC_2, \ldots, VC_{k-1}$, then the unconstrained safest path between $v_l$ and $v_k$ must go through a vertex which does not lie in $VC_1, VC_2, \ldots, VC_{k-1}, VC_k$, which contradicts Lemma 3. Thus, the $k^{th}$ unconstrained safest POI of $v_l$ lies in one of the adjacent Voronoi cells of $VC_1, VC_2, \ldots, VC_{k-1}$. □

### B. Prepossessing

The following information will be precomputed and stored for the faster processing of $k$SNN queries:

- Voronoi cells, where each Voronoi cell $VC_i$ has the POI $p_i$ as the generator, a set $B_i$ of border vertices, and a list $L_i$ of pointers to adjacent Voronoi Cells
- For every vertex $v \in B_i$, the unconstrained safest path between $v$ and $p_i$ and its PSS
- For every pair of border vertices $v_i \in B_i$ and $v_j \in B_i$, the unconstrained safest path between $v_i$ and $v_j$ that lies completely inside $VC_i$ and its PSS
- For every $v_x \in B_i$ the minimum border distance and the minimum POI distance, where the minimum border distance of a border vertex is defined as $argmin_{v_y \in B_i \setminus v_x} dist(pt_{xy}^{sh})$ and the minimum POI distance is $dist(pt_{xi}^{sh})$ and the shortest paths for the minimum border distance and the minimum POI distance are computed by considering only the subgraph represented by Voronoi cell $VC_i$
- For every vertex $v$ in the road network graph $G$, a pointer to the Voronoi cell whose POI generator is the unconstrained safest neighbour of $v$

To construct the SNVD, we first apply parallel Dijkstra algorithm [48] to find the unconstrained safest POI for each road network vertex. Starting from a vertex, the Dijkstra algorithm keeps tracks of the unconstrained safest path for every visited vertex. When the safest path's distance is unconstrained, no path gets pruned and thus unlike INE, it is sufficient to only expand the unconstrained safest path from a vertex.

Then we compute the border vertices as follows: (i) if the unconstrained safest paths from $v$ to two POIs $p_i$ and $p_j$ have the same PSS, then $v$ is considered as a border vertex of Voronoi cells $VC_i$ and $VC_j$, and (ii) if two end vertices of an edge have different unconstrained safest POIs, say $p_i$ and $p_j$, and the Edge Safety Score (ESS) associated with the edge is $s$, then the point $b$ of the edge that provides equal distance associated with $s$ in both of the $b$'s safest paths to $p_i$ and $p_j$ is identified as a border vertex of Voronoi cells $VC_i$ and $VC_j$. Figure 8 shows three example scenarios. In case 1, the end vertices of the edge with ESS 1 has different unconstrained safest POIs and the distance of the edge is 4. There is no other

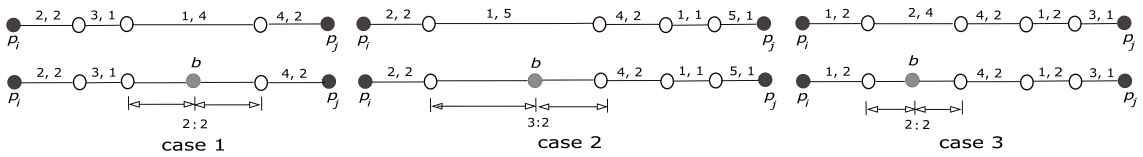



Fig. 8: Example border vertices of an SNVD, weights associated with each edge are $w_{ij}^{ss}$, $w_{ij}^{d}$, respectively

edge in $pt_{ij}$ with ESS 1. Thus $b$ is placed at the midpoint of the edge so that both of the $b$'s safest paths to $p_i$ and $p_j$ have equal distance for ESS 1. For case 2, $b$ is not the midpoint of the edge as there is another edge in $pt_{ij}$ with ESS 1. Note that in case 3, $b$ is placed at the mid of the edge with ESS 2, which is not the minimum ESS in $pt_{ij}$. This is because both of the $b$'s safest paths to $p_i$ and $p_j$ already have an edge with the same distance for ESS 1.

The resulted distances of the splitted edges with a border vertex $b$ may become less than 1. This occurs when the length of the edge that is splitted by $b$ is 1 and no other edge in $pt_{ij}$ has the same ESS as the splitted edge. Our PSS measure requires the edge weight to be greater than or equal to 1. Thus, after the construction of the whole SNVD, if the distance of any splitted edge becomes less than 1, we scale up the distances of all edges in the road network to ensure that the distances of the splitted edges are greater than or equal to 1 and recompute the SNVD.

To compute the unconstrained safest path between $v_i$ and $v_j$ that lies completely inside $VC_i$ for every pair of border vertices $v_i \in B_i$ and $v_j \in B_i$, we apply Dijkstra algorithm within $VC_i$.

### C. Query Processing

According to the construction of SNVD, a POI $p_i$ is the unconstrained safest neighbor for every location $v$ in a Voronoi cell $VC_i$. However, an unconstrained safest neighbor is not necessarily the SNN for a location $v$ if the distance of the unconstrained safest path from $v$ to $p_i$ is greater than or equal to the distance constraint $d_c$ specified in the query. Hence, unlike the existing NVD based nearest neighbor search algorithms [18], we cannot simply return the POI of the Voronoi cell that includes the query location as the SNN. Since $d_c$ is a query specific parameter, it is also not possible to consider $d_c$ while computing the Voronoi cells of the SNVD. In Sections VI-C1, we discuss our SNVD based solution to find $k$SNNs and in Section VI-C2, we elaborate our efficient (unconstrained and constrained) safest path computation techniques by exploiting SNVD properties.

*1) kSNN search:* Our SNN search technique using SNVD incrementally finds the unconstrained safest neighbors and adds each of these neighbors in a candidate set. The incremental search stops when the candidate set contains at least $k$ POIs that have unconstrained safest paths with length less than $d_c$. Let $k+m$ be the size of the candidate set where $m \geq 0$. The k SNNs are guaranteed to be among the candidate set. Then, we compute the PSSs of the safest paths from $v_l$ to the remaining $m$ POIs in the candidate set to determine the query answer. The POIs whose safest paths from $v_l$ among $k+m$ candidates have $k$ largest PSSs and have distances less than $d_c$ are identified as $k$SNNs.

Algorithm 3 shows the steps to find $k$SNNs in the road network using the SNVD. The algorithm's inputs are $v_l$, $k$ and $d_c$. The algorithm initializes $it$ and $j$ to 1, finds the first unconstrained safest neighbor using the stored pointer (Function $FindUSN$) and adds it to the candidate POI set $C_P$ (Lines 1–2). Then the algorithm iterates in a loop until $C_P$ includes $k$SNNs. In every iteration, the algorithm checks whether the distance of the unconstrained safest path from $v_l$ to the $j^{th}$ unconstrained safest neighbor is less than $d_c$. If yes, Function $IsConstrained$ returns true and $it$ is incremented by 1 (Lines 4–6)). Irrespective of the returned value (true or false) of Function $IsConstrained$, the algorithm increments $j$ by 1 and finds the next ($j^{th}$) unconstrained safest neighbor using Function $NextUCN$ (Lines 7–8). When $it$ becomes greater than $k$, then the loop ends and it is guaranteed that $C_P$ includes $k$SNNs. Finally, $FindA$ identifies $k$SNNs from $C_P$ in $A$ (Line 10), and Algorithm 3 returns $A$ as the $k$SNN query answer (Line 11).

**Algorithm 3** SNVD-$k$SNN($v_l, k, d_c$)

```
1:  it, j ← 1
2:  C_P ← FindUSN(v_l)
3:  while it ≤ k do
4:      if IsConstrained(C_P, j, d_c) then
5:          it ← it + 1
6:      end if
7:      j ← j + 1
8:      C_P ← C_P ∪ NextUSN(v_l, j)
9:  end while
10: A ← FindA(C_P)
11: return A
```

***Incremental computation of unconstrained safest neighbors.*** Function $FindUSN$ locates Voronoi cell $VC_q$ that includes $v_l$ and returns the corresponding POI $p_q$ as the first unconstrained safest neighbor. $VC_q$ becomes the first candidate Voronoi cell of the set $C_{VC}$. To compute the second unconstrained safest neighbor, Function $NextUSN$ adds the adjacent Voronoi cells of $VC_q$ to $C_{VC}$ as they include the second unconstrained safest POI of $v_l$ (Lemma 4). The POI of one of these candidate Voronoi cells in $C_{VC}$ whose unconstrained safest path to $v_l$ has the second best PSS is identified as the second unconstrained safest neighbor of $v_l$ (please see Section VI-C2 for unconstrained safest path computation technique). Similarly, to find the $j^{th}$ unconstrained safest neighbor of $v_l$, $NextUSN$ adds the adjacent Voronoi cells of the $(j-1)^{th}$ unconstrained safest neighbor's Voronoi cell in $C_{VC}$, if they are not already included in $C_{VC}$. The $j^{th}$ unconstrained safest neighbor of $v_l$ is the POI that has the $j^{th}$ best PSS for the unconstrained safest path from $v_l$ among the POIs of the

candidate Voronoi cells.

*2) Safest Path Computation:* Our PSS measure and the property shown in Lemma 3 allow us to adapt the shortest path computation technique proposed in [18] for finding the unconstrained safest paths from $v_l$ to a POI of the candidate Voronoi cell in $C_{VC}$. After *FindUSN* locates Voronoi cell $VC_q$ that includes $v_l$, it computes the unconstrained safest paths from $v_l$ to $p_q$ and the border vertices of $VC_q$ using Dijkstra algorithm. Then similar to [18], we use precomputed unconstrained safest paths between border vertices for this purpose and reduce the processing overhead significantly. In experiments, we show that the number of border vertices of an SNVD is small with respect to the total number of vertices and thus, the overhead for storing the unconstrained safest paths for the border vertices is negligible.

On the other hand, we cannot use precomputed unconstrained safest paths and adapt [18] to compute the safest path between $v_l$ and a POI while identifying $k$SNNs from candidate $k+m$ POIs. We improve the INE based safest path search technique discussed in Section IV by incorporating novel pruning techniques for search space refinement using SNVD properties and apply it to compute the safest paths that have distances less than $d_c$. Pruning Rules 5 and 6 use the minimum border distances and minimum POI distances of the border vertices, respectively, in the pruning condition:

**Pruning Rule 5.** *Let the safest path between $v_l$ and POI $p_y$ needs to be determined. A path $pt_{lj}$ can be pruned if $dist(pt_{lj}) + d_B^{min}(v_j, VC_x) \geq d_c$, where $v_j$ is a border vertex of Voronoi cell $VC_x$, $d_B^{min}(v_j, VC_x)$ represents the minimum border distance of $v_j$ and $d_c$ represent the distance constraint.*

**Pruning Rule 6.** *Let the safest path between $v_l$ and POI $p_x$ needs to be determined. A path $pt_{lj}$ can be pruned if $dist(pt_{lj}) + d_p^{min}(v_j, VC_x) \geq d_c$, where $v_j$ is a border vertex of Voronoi cell $VC_x$, $d_p^{min}(v_j, VC_x)$ represents the minimum POI distance of $v_j$ and $d_c$ represent the distance constraint.*

We have $s_k$, the upper bound of the PSS of the safest path between $v_l$ and its $k^{th}$ SNN, once $k$ unconstrained safest neighbors of $v_l$ that have distances less than $d_c$ from $v_l$ are identified. Later if a safest path from $v_l$ to a new POI is identified that has the distance less than $d_c$ and PSS higher than current $s_k$, then $s_k$ is updated to the new PSS. By exploiting Property 2 of PSS and $s_k$, we develop a new pruning technique as follows:

**Pruning Rule 7.** *A path $pt_{lj}$ can be pruned if $pss(pt_{lj}) < s_k$, where $s_k$ represent the upper bound of the PSS of the safest path between $v_l$ and its $k^{th}$ SNN.*

Thus, in addition to Pruning Rules 1 and 2, our SNVD based solution checks whether a path can be pruned using Pruning Rules 5, 6 and 7 to find the safest paths that have distances less than $d_c$.

In full version [41] of this paper, we provide complexity analysis of the algorithm and discuss how to update SNVD when there are updates to the road network and POIs.

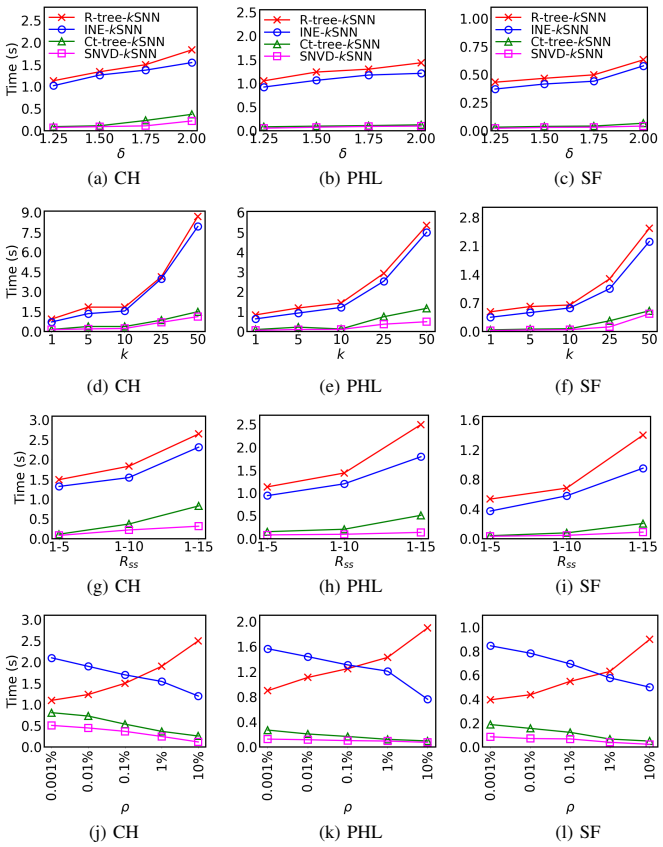



Fig. 9: Effect of varying $d_c$, $k$, $R_{ss}$ and $\rho$ for the three datasets

## VII. Experiments

### A. Setup

There is no existing work to compute $k$SNNs and thus, in this paper, we compare our algorithms *Ct*-tree-$k$SNN and SNVD-$k$SNN against two baselines: *R*-tree-$k$SNN and INE-$k$SNN. *R*-tree-$k$SNN, indexes POIs using an *R*-tree and uses techniques in [49] to find the candidate $k$SNN POIs whose Euclidean distances from the query location are less than $d_c$. Then, *R*-tree-$k$SNN applies the most recent technique [9] to find the safest valid paths to these candidate POIs to determine the $k$SNNs. INE-$k$SNN is the extension of INE for evaluating $k$SNN queries (Section IV).

***Datasets:*** We used real-world datasets of three cities that differ in terms of the road network size and crime statistics. Specifically, we extract the road network of Chicago, Philadelphia and San Francisco, using OpenStreetMap (Table III). All edge weights representing distances were scaled up and integer values were taken.

*Edge $w_{ij}^{ss}$ Computation.* We used crime datasets of Chicago [50], Philadelphia [51] and San Francisco [52] to assign weights ($w_{ij}^{ss}$) representing realistic Edge Safety Score (ESS)s of the respective road network graphs. We generated the ESSs of edges separately for three different ESS ranges $[1, s^{max}]$ denoted as $R_{ss}$ (see Table IV). The crime datasets include locations (latitude and longitude) of the crime incidents like robbery, motor vehicle theft, sexual offense, weapons violation, burglary and criminal damage to vehicle. For each edge, we count the number of crime incidents within 1km and

set it as the crime count of the edge. These crime counts are normalized to integers between the range $[1, s^{max}]$ representing crime scores. The crime score of each edge is converted to a safety score as $s^{max} + 1 - crimescore$, e.g., if $s^{max}$ is 10 and the crime score for an edge is 10, the safety score of the edge is 1. The PSS of a path is computed based on the ESSs included in the path (Definition 2) and can be a float value.

TABLE III: Road Network Dataset

| Dataset | # Vertices | # Edges |
|---|---|---|
| Chicago (**CH**) | 125,344 | 200,110 |
| Philadelphia (**PHL**) | 80,558 | 120, 581 |
| San Francisco (**SF**) | 40,528 | 72,819 |

***Parameters:*** The distance constraint $d_c$ is a user specified parameter. Although our solution works for any $d_c$, in our experiments, we set $d_c$ to a value that is large enough to find the $k$SNNs for each query. Let $d^k$ be the distance between query location $v_l$ and its $k^{th}$ closest POI. We set $d_c = \delta \times d^k$ where $\delta > 1$ is a parameter in the experiments varied from 1.25 to 2. Existing studies involving other parameterized queries also adapt similar settings to ensure query results are not empty [53] and contain at least $k$ POIs [54]. We evaluate the algorithms by varying $\delta$, ESS range ($R_{ss}$), $k$, and the density of POIs ($\rho$), where density is the total number of POIs divided by the total number of vertices in the graph. The range and default values of the parameters are shown in Table IV. The justification of the parameter values can be found in [41].

TABLE IV: Experiment Parameters

| Parameters | Range | Default |
|---|---|---|
| $k$ | 1, 5, 10, 25, 50 | 10 |
| $\delta$ | 1.25, 1.5, 1.75, 2 | 2 |
| $R_{ss}$ | 1-5, 1-10, 1-15 | 1-10 |
| $\rho$ | 0.001%, 0.01%, 0.1%, 1%, 10% | 1% |

***Measures:*** For each experiment, we randomly select 100 vertices as query locations and report the average processing time to evaluate a $k$SNN query. We measure the preprocessing and storage cost for $Ct$-tree and SNVD in terms of the construction time, index size and the percentage of border vertices. All algorithms are implemented in C++ and the experiments are run on a 64-bit Windows 7 machine with an Intel Core i3-2350M, 2.30GHz processor and 4GB RAM.

### *B. Results*

***Effect of $\delta$:*** Figures 9(a)–9(c) show that our algorithms can find $k$SNNs in real time. The query processing time slowly increases with the increase of $\delta$ (and consequently $d_c$) because more paths need to be explored for larger $\delta$. For $Ct$-tree-$k$SNN and SNVD-$k$SNN, we observe on average 11.6 times and 13.8 times faster processing time than that of R-tree-$k$SNN and on average 9.7 times and 12.7 times faster processing time than that of INE-$k$SNN, respectively.

***Effect of $k$:*** Figures 9(d)–9(f) show that the time increases with the increase of $k$, which is expected. The increase rate is low for $Ct$-tree-$k$SNN and SNVD-$k$SNN and shows almost a linear growth for varying $k$ from 1 to 50, whereas in case of INE-$k$SNN and R-tree-$k$SNN, the time increases significantly.

TABLE V: Effectiveness of Pruning Rules (PRules) on CH dataset. The Basic algorithm in each approach only applies Pruning Rule 1. Each % value $x$ shown in a parenthesis indicates that the value in this cell is $x$% of the value of the Basic of the same approach.

| | Setting | Time (ms) | $\|V_a\|$ | # of Valid Paths |
|---|---|---|---|---|
| INE | Basic | 2225 | 2708 | 18223 |
| | PRule 2 | 1787 (80.31%) | 2545 (93.9%) | 14534 (79.75%) |
| $Ct$-tree | Basic | 682 | 1601 | 4531 |
| | PRule 2 | 485 (71.11%) | 1056 (65.95%) | 3287 (72.65%) |
| | PRule 3 | 460 (67.44%) | 942 (58.83%) | 2721 (60.05%) |
| | PRule 4 | 471 (69.06%) | 955 (59.65%) | 2913 (64.31 % ) |
| | All | 377 (55.27%) | 875 (54.65 % ) | 2621 (57.84%) |
| SNVD | Basic | 442 | 391 | 942 |
| | PRule 2 | 349 (78.95%) | 341 (87.21%) | 699 (74.20 % ) |
| | PRules 5, 6 | 364 (82.35%) | 356 (91.04%) | 719 (76.32 % ) |
| | PRule 7 | 379 (85.74%) | 371 (94.88%) | 772 (81.95%) |
| | All | 229 (51.81%) | 162 (58.56%) | 623 (66.13%) |

***Effect of $R_{ss}$:*** Figures 9(g)–9(i) show that the time increases for R-tree-$k$SNN, INE-$k$SNN and $Ct$-tree-$k$SNN for larger $R_{ss}$, whereas for SNVD-$k$SNN the required time is the least among four algorithms and remains almost constant for all $R_{ss}$s. A larger $R_{ss}$ increases the height of the $Ct$-tree which in turn increases the number of subgraphs accessed by $Ct$-tree-$k$SNN. On the other hand, the number of subgraphs represented by SNVD cells does not vary with different $R_{ss}$s.

***Effect of $\rho$:*** Figures 9(j)–9(l) show that the time decreases for INE-$k$SNN, Ct-$k$SNN and SNVD-$k$SNN with the increase of the POI density, $\rho$, whereas the time sharply increases for R-tree-$k$SNN. For INE-$k$SNN, a larger $\rho$ increases the possibility of finding the required POIs with less amount of incremental network expansion. For $Ct$-tree-$k$SNN, a larger $\rho$ increases the number of POIs in a $Ct$-tree node which in turn decreases the need of search in a larger subgraph. For SNVD-$k$SNN, the decrease curve is less steeper, since in SNVD-$k$SNN, the search time mainly depends on the number of SNVD cells that need to be explored rather than the total number of SNVD cells in the SNVD. On the other hand, for R-tree-$k$SNN, the number of candidates for $k$SNNs increases with the increase in $\rho$, which also causes increase in the number of independent safest path computations.

***Effect of Datasets and Performance Scalability:*** In our experiments, we used three datasets that vary in the number of road network vertices, edges, POIs and crime distributions. Irrespective of the datasets, both $Ct$-tree-$k$SNN and SNVD-$k$SNN outperform R-tree-$k$SNN and INE-$k$SNN with a large margin. In addition, the query processing time linearly increases with increase of the dataset size in terms of the number of road network vertices, edges and POIs. Thus, our algorithms are scalable and applicable for all settings. In default parameter setting, the average time for Ct-kSNN is 371 ms, 123 ms and 66 ms for CH, PHL, and SF datasets, respectively. The average time for SNVD-kSNN is 223 ms, 96 ms and 41 ms for for CH, PHL, and SF datasets, respectively.

***Pruning Effectiveness:*** We investigate the effectiveness of our pruning rules for the search space refinement. Table V summarizes the experiment results in the default setting of the parameters in terms of the query processing time, number of network vertices accessed $\|V_a\|$ and number of valid paths

TABLE VI: For $k = 10$, comparison of $k$NN and $k$SNN queries on CH dataset showing: average # of common POIs in $k$NNs and $k$SNNs; probability that 1SNN is one of the POIs in $k$NNs; average PSS of the $k$ paths returned by $k$NN queries vs $k$SNN queries, and their ratios (i.e., $k$NN:$k$SNN); and average length of the $k$ paths returned by $k$NN vs $k$SNN queries, and their ratios.

| Measure | | $\delta$=1.25 | $\delta$ =1.5 | $\delta$ = 1.75 | $\delta$ = 2 |
|---|---|---|---|---|---|
| #common POIs | | 4.71 | 4.12 | 2.23 | 1.34 |
| probability of 1SNN in $k$NN | | 0.19 | 0.15 | 0.13 | 0.12 |
| PSS | $k$NN | 0.00029 | 0.00029 | 0.00029 | 0.00029 |
| | $k$SNN | 0.00091 | 0.0018 | 0.0024 | 0.0048 |
| | ratio | 1:3.1 | 1:6.2 | 1:8.3 | 1:16.5 |
| Length | $k$NN | 1311 | 1311 | 1311 | 1311 |
| | $k$SNN | 1507 | 1717 | 1941 | 2084 |
| | ratio | 1:1.15 | 1:1.3 | 1:1.5 | 1:1.6 |

explored in road networks for different approaches when different pruning rules are applied. The Basic Algorithm includes only Pruning Rule 1(i.e., considers only valid paths). The performance of our algorithms are the best when all pruning rules are applied, which in turn means that each of the pruning rule contributes in improving the efficiency of our algorithms. Table V also explains why SNVD is faster than *Ct*-tree and *Ct*-tree is faster than INE based approach.

***Query Effectiveness:*** We compare $k$SNNs with traditional $k$ nearest neighbors ($k$NNs) on CH dataset using default settings but varying $\delta$. The results on the other datasets can be found in [41]. We generate 100 query locations and run a kNN query and a kSNN query for each of these query locations, and report average results. We compare $k$SNNs with traditional $k$ nearest neighbors ($k$NNs) using default settings but varying $\delta$. Specifically, for each data set, we generate 100 query locations and run a kNN query and a kSNN query for each of these query locations, and report average results. Table VI shows that $k$SNNs are significantly different from $k$NNs (see number of common POIs). Furthermore, the traditional $k$NN queries are not likely to include a POI that is also the first SNN. Finally, although the path lengths to $k$SNNs are longer, these paths are much safer. Specifically, the paths to $k$SNNs are up to 1.6 times longer on average but they are up to 16 times safer than the paths to $k$NNs. The number of common POIs among $k$SNNs and $k$NNs decreases and the safety (PSS) of the paths from $v_l$ to $k$SNNs increases with the increase of $\delta$. Thus, in real-world scenarios where safety may be important for users, an application may show some $k$SNNs as well as some $k$NNs to give users more options to choose from.

***Preprocessing and Index Size:*** We also evaluated the construction time and index size of *Ct*-tree and SNVD. The detailed results can be found in [41]. We observe that the construction times for *Ct*-tree and SNVD are comparable whereas SNVD is up to 1.6 times bigger in size than *Ct*-tree. Specifically, for default settings (i.e., $R_{ss} = 1 - 10$, $\rho = 1\%$), the construction time for *Ct*-tree for the three datasets ranges from $54 - 123$ minutes as compared to $50 - 122$ minutes for SNVD. On the other hand, the index size of *Ct*-tree for the three datasets ranges from $63 - 114$ MB compared to $101 - 168$ MB for SNVD.

## VIII. Conclusions

In this paper, we introduced $k$ Safest Nearby neighbor ($k$SNN) queries in road networks and formulated the measure of path safety score. We proposed novel indexing structures, *Ct*-tree and a safety score based Voronoi diagram (SNVD), to efficiently evaluate $k$SNN queries. We adapt the INE based technique and the *R*-tree based technique to develop two baselines to find $k$SNNs. Our extensive experimental study on three real-world datasets shows that *Ct*-tree and SNVD based approaches are up to an order of magnitude faster than the baselines. Comparing *Ct*-tree and SNVD on default settings, both approaches have comparable construction time whereas SNVD index is around 1.6 times bigger than *Ct*-tree but is 1.2-1.7 times faster in terms of query processing cost. Thus, if the storage is not an issue, one should go for the SNVD based approach to get faster query processing performance.

## Acknowledgments

Tanzima Hashem is supported by basic research grant of Bangladesh University of Engineering and Technology. Muhammad Aamir Cheema is supported by the Australian Research Council DP230100081 and FT180100140.

## References

[1] C. H. Stubbert, S. F. Pires, and R. T. Guerette, "Crime science and crime epidemics in developing countries: a reflection on kidnapping for ransom in colombia, south america," *Crime Science*, 2015.
[2] M. Natarajan, "Crime in developing countries: the contribution of crime science," *Crime Science*, vol. 5, pp. 1–5, 2016.
[3] V. Spicer, J. Song, P. Brantingham, A. Park, and M. A. Andresen, "Street profile analysis: A new method for mapping crime on major roadways," *Applied Geography*, vol. 69, pp. 65–74, 2016.
[4] BBC. (23 October 2018) Street harassment 'relentless' for women and girls. [Online]. Available: https://www.bbc.com/news/uk-politics-45942447
[5] P. International. (20 April 2020) 1 in 5 girls have experienced street harassment during lockdown. [Online]. Available: https://plan-uk.org/media-centre/1-in-5-girls-have-experienced-street-harassment-during-lockdown
[6] S. Aljubayrin, J. Qi, C. S. Jensen, R. Zhang, Z. He, and Y. Li, "Finding lowest-cost paths in settings with safe and preferred zones," *The VLDB Journal*, vol. 26, no. 3, pp. 373–397, 2017.
[7] S. Aljubayrin, J. Qi, C. S. Jensen, R. Zhang, Z. He, and Z. Wen, "The safest path via safe zones," in *ICDE*, 2015, pp. 531–542.
[8] J. Kim, M. Cha, and T. Sandholm, "Socroutes: safe routes based on tweet sentiments," in *WWW*, 2014, pp. 179–182.
[9] F. T. Islam, T. Hashem, and R. Shahriyar, "A privacy-enhanced and personalized safe route planner with crowdsourced data and computation," in *ICDE*, 2021, pp. 229–240.
[10] A. Guttman, "R-trees: A dynamic index structure for spatial searching," in *SIGMOD*, 1984, p. 47–57.
[11] N. Beckmann, H.-P. Kriegel, R. Schneider, and B. Seeger, "The R*-tree: An efficient and robust access method for points and rectangles," *SIGMOD Rec.*, vol. 19, no. 2, p. 322–331, May 1990.
[12] K. C. K. Lee, W. Lee, B. Zheng, and Y. Tian, "ROAD: A new spatial object search framework for road networks," *IEEE TKDE*, 2012.
[13] R. Zhong, G. Li, K. Tan, L. Zhou, and Z. Gong, "G-tree: An efficient and scalable index for spatial search on road networks," *IEEE TKDE*, vol. 27, no. 8, pp. 2175–2189, 2015.
[14] Z. Li, L. Chen, and Y. Wang, "G*-tree: An efficient spatial index on road networks," in *ICDE*, 2019, pp. 268–279.
[15] T. Dan, C. Luo, Y. Li, Z. Guan, and X. Meng, "LG-tree: An efficient labeled index for shortest distance search on massive road networks," *IEEE T-ITS*, vol. 23, no. 12, pp. 23 721–23 735, 2022.

[16] X. Min, D. Pfoser, A. Züfle, and Y. Sheng, "A hierarchical spatial network index for arbitrarily distributed spatial objects," *ISPRS Int. J. Geo Inf.*, vol. 10, no. 12, p. 814, 2021.

[17] D. Papadias, J. Zhang, N. Mamoulis, and Y. Tao, "Query processing in spatial network databases," in *VLDB*, 2003, pp. 802–813.

[18] M. R. Kolahdouzan and C. Shahabi, "Voronoi-based K nearest neighbor search for spatial network databases," in *VLDB*, 2004.

[19] S. Chainey and J. Ratcliffe, *GIS and crime mapping*. John Wiley & Sons, 2013.

[20] E. Galbrun, K. Pelechrinis, and E. Terzi, "Urban navigation beyond shortest route: The case of safe paths," *Inf. Syst.*, 2016.

[21] E. W. Dijkstra, "A note on two problems in connexion with graphs," *Numerische Mathematik*, vol. 1, pp. 269–271, 1959.

[22] S. Nadi and M. R. Delavar, "Multi-criteria, personalized route planning using quantifier-guided ordered weighted averaging operators," *Int. J. Appl. Earth Obs. Geoinformation*, 2011.

[23] C. Salgado, M. A. Cheema, and D. Taniar, "An efficient approximation algorithm for multi-criteria indoor route planning queries," in *SIGSPATIAL*, 2018, pp. 448–451.

[24] B. Shen, M. A. Cheema, D. Harabor, and P. J. Stuckey, "Euclidean pathfinding with compressed path databases." in *IJCAI*, 2020.

[25] A. Lambert and D. Gruyer, "Safe path planning in an uncertain-configuration space," in *ICRA*, 2003, pp. 4185–4190.

[26] L. Leenen, A. Terlunen, and H. Le Roux, "A constraint programming solution for the military unit path finding problem," in *Mobile Intelligent Autonomous Systems*, 2012, pp. 225—-240.

[27] J. P. van den Berg and M. H. Overmars, "Planning the shortest safe path amidst unpredictably moving obstacles," in *Algorithmic Foundation of Robotics VII*, 2006, pp. 103–118.

[28] A. Anwar and T. Hashem, "Optimal obstructed sequenced route queries in spatial databases," in *EDBT*, 2017.

[29] L. Li, M. A. Cheema, H. Lu, M. E. Ali, and A. N. Toosi, "Comparing alternative route planning techniques: A comparative user study on melbourne, dhaka and copenhagen road networks," *IEEE TKDE*, 2021.

[30] L. Li, M. A. Cheema, M. E. Ali, H. Lu, and D. Taniar, "Continuously monitoring alternative shortest paths on road networks," *PVLDB*, 2020.

[31] Z. Luo, L. Li, M. Zhang, W. Hua, Y. Xu, and X. Zhou, "Diversified top-k route planning in road network," *PVLDB*, 2022.

[32] T. Hashem, T. Hashem, M. E. Ali, and L. Kulik, "Group trip planning queries in spatial databases," in *SSTD*, 2013.

[33] M. Sharifzadeh, M. R. Kolahdouzan, and C. Shahabi, "The optimal sequenced route query," *The VLDB Journal*, 2008.

[34] H. Zhu, W. Li, W. Liu, J. Yin, and J. Xu, "Top k optimal sequenced route query with POI preferences," *Data Sci. Eng.*, 2022.

[35] K. Fu, Y. Lu, and C. Lu, "TREADS: a safe route recommender using social media mining and text summarization," in *SIGSPATIAL*, 2014.

[36] S. Shah, F. Bao, C. Lu, and I. Chen, "CROWDSAFE: crowd sourcing of crime incidents and safe routing on mobile devices," in *SIGSPATIAL*, 2011.

[37] G. Jossé, K. A. Schmid, A. Züfle, G. Skoumas, M. Schubert, and D. Pfoser, "Tourismo: A user-preference tourist trip search engine," in *SSTD*, 2015, pp. 514–519.

[38] E. Hossain, M. R. Karim, M. Hasan, S. A. Zaoad, T. Tanjim, and M. M. Khan, "Spafe: A crowdsourcing and multimodal recommender system to ensure travel safety in a city," *IEEE Access*, 2022.

[39] R. Kaur, V. Goyal, V. M. V. Gunturi, A. Saini, K. Sanadhya, R. Gupta, and S. Ratra, "A navigation system for safe routing," in *MDM*, 2021.

[40] T. Abeywickrama, M. A. Cheema, and D. Taniar, "k-nearest neighbors on road networks: A journey in experimentation and in-memory implementation," *Proc. VLDB Endow.*, 2016.

[41] P. Biswas, T. Hashem, and M. A. Cheema, "Safest nearby neighbor queries in road networks (full version)," *arXiv:2107.14122*, 2021.

[42] R. Geisberger, P. Sanders, D. Schultes, and D. Delling, "Contraction hierarchies: Faster and simpler hierarchical routing in road networks," in *WEA*, 2008, pp. 319–333.

[43] R. A. Finkel and J. L. Bentley, "Quad trees: A data structure for retrieval on composite keys," *Acta Inf.*, vol. 4, pp. 1–9, 1974.

[44] A. Okabe, B. Boots, K. Sugihara, S. N. Chiu, and D. G. Kendall, *Spatial Tessellations: Concepts and Applications of Voronoi Diagrams, Second Edition*. Wiley, 2000.

[45] A. Okabe, T. Satoh, T. Furuta, A. Suzuki, and K. Okano, "Generalized network voronoi diagrams: Concepts, computational methods, and applications," *IJGIS*, vol. 22, no. 9, pp. 965–994, 2008.

[46] M. Safar, D. Ebrahimi, and D. Taniar, "Voronoi-based reverse nearest neighbor query processing on spatial networks," *Multimedia Syst.*, vol. 15, no. 5, pp. 295–308, 2009.

[47] K. Xuan, G. Zhao, D. Taniar, B. Srinivasan, M. Safar, and M. L. Gavrilova, "Network voronoi diagram based range search," in *AINA*, 2009, pp. 741–748.

[48] M. Erwig, "The graph voronoi diagram with applications," *Networks*, vol. 36, no. 3, pp. 156–163, 2000.

[49] G. R. Hjaltason and H. Samet, "Distance browsing in spatial databases," *ACM Trans. Database Syst.*, 1999.

[50] C. D. Portal. (2001-2019) Crimes - 2001 to present. Last accessed: April 26, 2021. [Online]. Available: https://data.cityofchicago.org/Public-Safety/Crimes-2001-to-Present/ijzp-q8t2

[51] OpenDataPhilly. (2006-2019) Crime incidents. [Online]. Available: https://www.opendataphilly.org/dataset/crime-incidents

[52] DataSF. (2003-2018) Police department incident reports: Historical 2003 to may 2018. [Online]. Available: https://data.sfgov.org/Public-Safety/Police-Department-Incident-Reports-Historical-2003/tmnf-yvry

[53] L. Chen, G. Cong, C. S. Jensen, and D. Wu, "Spatial keyword query processing: An experimental evaluation," *PVLDB*, 2013.

[54] Z. Shao, M. A. Cheema, D. Taniar, H. Lu, and S. Yang, "Efficiently processing spatial and keyword queries in indoor venues," *IEEE TKDE*, 2020.


**Punam Biswas** Punam Biswas is a P.H.D. student at the Department of Computer Science and Engineering of Bangladesh University of Engineering and Technology (BUET). She completed her B.Sc. and M.Sc. degree from Bangladesh University of Engineering and Technology (BUET). Her research interest falls in the area of human-computer interaction, distributed computing and spatial database.

**Tanzima Hashem** Dr. Tanzima Hashem is a professor at the Department of Computer Science and Engineering of Bangladesh University of Engineering and Technology (BUET). She received her Ph.D. degree from the University of Melbourne, Australia in 2012. Her research interest falls in the area of ubiquitous computing, spatial databases and privacy. In 2017, she received the prestigious OWSD-Elsevier Foundation Award for Early-Career Women Scientists in the Developing World in Engineering Sciences.

**Muhammad Aamir Cheema** is an ARC Future Fellow, an Associate Professor at the Faculty of Information Technology, Monash University, Australia. He obtained his PhD from UNSW Australia in 2011. He is the recipient of 2012 Malcolm Chaikin Prize for Research Excellence in Engineering, 2013 Discovery Early Career Researcher Award, 2014 Dean's Award for Excellence in Research by an Early Career Researcher, 2018 Future Fellowship, 2018 Monash Student Association Teaching Award and 2019 Young Tall Poppy Science Award. He has also won two CiSRA best research paper of the year awards, two invited papers in the special issue of IEEE TKDE on the best papers of ICDE, and three best paper awards at ICAPS 2020, WISE 2013 and ADC 2010, respectively.