IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received May 12, 2022, accepted May 20, 2022, date of publication May 23, 2022, date of current version May 27, 2022.

Digital Object Identifier 10.1109/ACCESS.2022.3177620

Efficient Retrieval of Top-k Weighted Triangles on Static and Dynamic Spatial Data

RYOSUKE TANIGUCHI, DAICHI AMAGATA, (Member, IEEE), AND TAKAHIRO HARA, (Senior Member, IEEE)
Graduate School of Information Science and Technology, Osaka University, Suita 565-0871, Japan
Corresponding author: Ryosuke Taniguchi (taniguchi.ryosuke@ist.osaka-u.ac.jp)

This work was supported in part by JST PRESTO under Grant JPMJPR1931, in part by JSPS Grant-in-Aid for Scientific Research (A) under Grant 18H04095, and in part by JST CREST under Grant JPMJCR21F2.

ABSTRACT Due to the proliferation of location-based services, spatial data analysis becomes more and more important. We consider graphs consisting of spatial points, where each point has edges to its nearby points and the weight of each edge is the distance between the corresponding points, as they have been receiving attention as spatial data analysis tools. We focus on triangles in such graphs and address the problem of retrieving the top-k weighted spatial triangles. This problem is computationally challenging, because the number of triangles in a graph is generally huge and enumerating all of them is not feasible. To overcome this challenge, we propose an algorithm that returns the exact result efficiently. We moreover consider two dynamic data models: (i) fully dynamic data that allow arbitrary point insertions and deletions and (ii) streaming data in a sliding-window model. They often appear in location-based services. The results of our experiments on real datasets show the efficiency of our algorithms for static and dynamic data.

INDEX TERMS Dynamic data, spatial points, top-k retrieval, weighted graph.

I. INTRODUCTION

Due to the proliferation of location-based services and IoT devices, many geo-location points are being generated nowadays. Useful observations can be obtained by analyzing such spatial points, thereby existing works devised many spatial point processing techniques [2]–[5], [20] and systems [21], [28], [32] so far. As spatial point analysis tools, graph-based approaches have recently been receiving much attention [9], [11]–[13], [30], [34].

A. MOTIVATION AND CHALLENGE

Given a set P of spatial points and a distance threshold r, a spatial neighbor graph of P consists of a set of vertices that correspond to points in P and a set of edges where an edge is created between two points iff the distance between them is not larger than r and the weight of this edge is the distance. Graph-based structures provide intuitive relationships between spatial points, so techniques that mine some patterns (i.e., sub-graphs) from spatial neighbor graphs are often required. In graph contexts, triangles are particularly considered as triangle is one of the simplest yet important primitive sub-graph patterns having many applications [17], [22]. For instance, spatial triangles can be utilized in group search [14], co-location pattern mining [34], and urban planning [12], [13]. Note that the number of triangles in a spatial neighbor graph is generally huge. It is not feasible to enumerate all of them, and the output size should be controllable (by a user-specified parameter k) [13], [17]. In spatial databases, given a subset of points in P, the cohesiveness of the subset is a factor in measuring its importance [14], [34].

The above applications and observations motivate us to address the problem of retrieving the top-k weighted spatial triangles. The weight of the triangle formed by points p_x, p_y, and p_z is defined as $dist(p_x, p_y) + dist(p_y, p_z) + dist(p_x, p_z)$, where $dist(\cdot, \cdot)$ measures the Euclidean distance between two points, which takes into account the cohesiveness. Then, given P and k, this problem retrieves k spatial triangles with the minimum weight among all triangles in the spatial neighbor graph of P. For example, this problem formulation yields the following observation:

Example 1: We ran the above problem on a real dataset Places, a set of POIs in the U.S.A., by setting $k = 100$, and found some co-location patterns. First, we observed

The associate editor coordinating the review of this manuscript and approving it for publication was Chong Leong Gan.

an intuitive pattern: industrial and precision manufacturing facilities exist near machine shops. We secondly observed that ⟨dentist, psychologist, consultant⟩ appears multiple times in the top-k triangles, suggesting that (psychological) consulting services tend to exist near clinics (or hospitals). In addition, we found that consultant services tend to exist near capital and risk management services, such as investment and stocks, in the top-k result.

As seen above, this problem helps analysts and experts mine (i) relationships between points and (ii) patterns/knowledge hidden in spatial datasets, and they also help consider where to open a new store (or service).

However, this problem is computationally challenging. A straightforward approach is to enumerate all triangles and then output k triangles with the minimum weight. The number of triangles in the spatial neighbor graph is exponential to the dataset size, suggesting the infeasibility of this approach. To alleviate this cost, DHL [17], which is a heuristic algorithm and was proposed originally for graph databases, can be used. DHL needs to sort edges in order of weights, because it greedily accesses the edges in this order to avoid enumerating triangles with large weights. However, if we employ DHL, we face substantial time incurred by sorting a large amount of edges of the spatial neighbor graph of P.

B. CONTRIBUTION

To solve the above issues, we propose an efficient algorithm that returns the exact answer. We observe that a subset of the spatial neighbor graph, which usually contains the top-k weighted triangles, can be built offline. From this partial graph, for each point $p \in P$, we can enumerate a triangle having p with a small weight in $O(1)$ time offline. These n triangles provide a tight threshold for the top-k result, which helps filter unnecessary points and triangles, accelerating online computation. Thanks to these observations, our algorithm does not need to correctly build the spatial graph and sort all edges.

We moreover consider insertions of new points and deletions of existing points, because this case often appears in location-based services [2], [10]. In this case, the top-k result may change, thus we need to efficiently update the result whenever we have an update (insertion or deletion). We show that our filtering idea for static data is still effective for dynamic data. Furthermore, we consider a sliding-window model for applications that focus only on *recently generated points* [2], [3], [19], [20]. We also design an efficient and exact algorithm for this case.

We summarize our main contributions below.

- We address the problem of retrieving the top-k weighted spatial triangles. To the best of our knowledge, this is the first work to tackle this problem in spatial databases.
- We propose a simple yet efficient algorithm for solving this problem exactly.
- We show how to deal with fully dynamic data to efficiently update the top-k result.
- We design an efficient and exact algorithm for monitoring the top-k result under a sliding-window model.
- We conduct experiments on real datasets, and the results show that (i) our solution for static data is up to *three orders of magnitude faster* than a baseline algorithm and (ii) our solutions for dynamic data can quickly update the top-k result.

This article significantly extends our conference paper [26]. Compared with this paper, this article provides

- more detailed explanations of our solution for static data with examples and pseudo codes,
- an exact algorithm for fully dynamic data,
- an exact algorithm for streaming data in a sliding-window model,
- a detailed performance statistics of our solution for static data,
- experimental results of our solutions for dynamic data, and
- surveys about related works.

C. ORGANIZATION

The rest of this article is organized as follows. Section II introduces preliminary information. We present our solutions for static data, fully dynamic data, and sliding-window data, in Sections III, IV, and V, respectively. We report our experimental results in Section VI. We review related work in Section VII. Finally, in Section VIII, we conclude this article.

II. PROBLEM DEFINITION

Let P be a set of spatial (or geo-location) points in a Euclidean space. A spatial point $p \in P$ has 2-dimensional coordinates $\in \mathbb{R}^2$. The Euclidean distance between p and p' is denoted by $dist(p, p')$. Given a distance threshold r, we can build a spatial neighbor graph of P defined below:

*Definition 1 (*Spatial Neighbor Graph*): Given a set P of points and a distance threshold r, the spatial neighbor graph of P is an undirected graph consisting of a set of vertices that correspond to the points in P and a set of edges where an edge is created between p_i and p_j iff $dist(p_i, p_j) \leq r$. The edge between p_i and p_j is represented as $e_{i,j}$ and has a weight $w(e_{i,j})$ where $w(e_{i,j}) = dist(p_i, p_j)$.*

In the spatial neighbor graph, there are triangles consisting of three points fully connected to each other. We define their weight:

*Definition 2 (*Weight of a Triangle*): Given a triangle $\triangle_{x,y,z}$ consisting of three points p_x, p_y, and p_z, the weight of this triangle, $w(\triangle_{x,y,z})$, is:*

$$w(\triangle_{x,y,z}) = dist(p_x, p_y) + dist(p_y, p_z) + dist(p_x, p_z). \quad (1)$$

Section III addresses the problem defined as follows:

*Definition 3 (*Top-k Weighted Triangle Retrieval Problem*): Given a set P of points, an output size k, and a distance threshold r, this problem is to retrieve at most k triangles in the spatial neighbor graph of P with the minimum weight.*

We assume that r is reasonably specified so that there are many triangles in the graph.

TABLE 1. Summary of notations.

Notation	Meaning
p	a 2-dimensional (geo-spatial) point
P	a set of n points
$dist(p, p')$	the Euclidean distance between p and p'
$\triangle_{x,y,z}$	the triangle formed by p_x, p_y, and p_z
$w(\triangle_{x,y,z})$	the weight of $\triangle_{x,y,z}$
k	a result size
r	a distance threshold
B	a batch size
T	a set of triangles
τ	a weight threshold
θ	an edge weight threshold
R	a top-k result set
$N(p)$	a set of neighbors of p
W	a sliding window size

When P is a dynamic set of spatial points, it is required to update the top-k result. This problem, which we address in Section IV, is formally defined as follows:

Definition 4 (Top-k Weighted Triangle Monitoring Problem on Fully Dynamic Data): Given a dynamic set P of points, an output size k, and a distance threshold r, this problem is to monitor (or update) at most k triangles in the spatial neighbor graph of P with the minimum weight, whenever P has updates (insertions and/or deletions of points).

Last, when P is a set of streaming points, a sliding-window model, which takes only the most recent W points into account, is usually employed [2], [3], [19], [20]. Section V assumes this case and addresses the following problem.

Definition 5 (Top-k Weighted Triangle Monitoring Problem on a Sliding-Window Model): Given a set P of streaming points, an output size k, a windows size W, and a distance threshold r, this problem is to monitor (or update) at most k triangles in the spatial neighbor graph of P_W with the minimum weight, where P_W contains the W most recently generated points in P.

Table 1 summarise notations used frequently in this article.

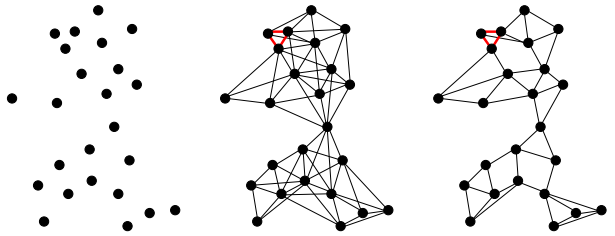
III. OUR SOLUTION FOR STATIC DATA

This section presents our proposed solution. Section III-A introduces our main idea. In Sections III-B and III-C, we detail our offline and online algorithms, respectively.

A. MAIN IDEA

To efficiently output the result, pruning points that do not contribute to the top-k result is important. Assume that triangle $\triangle_{x,y,z}$ is included in the top-k result. From Equation (1) and Definition 3, it is intuitively seen that, for p_x, edges $e_{x,y}$ and $e_{x,z}$ would be (two of) the t nearest neighbors (t-NNs) of p_x, where t is a small constant. This suggests that the top-k triangles can be retrieved from the t-NN graph and that correct building of the spatial neighbor graph of P is not necessary.

Example 2: Figure 1 depicts our idea. Consider an example of P shown in Figure 1(a). Figures 1(b) and 1(c) respectively show the spatial neighbor graph of P and 3-NN graph of P. The red edges show the top-1 weighted spatial triangle. It can be seen that we can obtain the result from a sparser graph than the spatial neighbor graph.



(a) Example of P (b) Spatial neighbor graph of P (c) 3-NN graph of P

FIGURE 1. Illustration of our idea. Red edges form the top-1 weighted spatial triangle.

Now assume that we have the t-NN graph of P, then, we can enumerate a promising triangle having p, i.e., the triangle formed by p and its 2 nearest neighbors offline, for each $p \in P$. Even if these triangles are not included in the top-k result, they have small weights, leading to a tight threshold for online computation that helps prune unnecessary points (and triangles). Our algorithm is designed based on the above ideas and consists of a one-time offline computation and online computation.

B. OFFLINE PROCESSING

Algorithm 1 describes our offline algorithm. The objectives of this offline processing are to (i) build a B-NN graph of P, where $B \geq 3$ is a batch size, and (ii) enumerate triangles with small weights. The batch size B is tuned empirically, and we show that a small constant (e.g., 10) is enough in Section VI-A. We use $p.E$ to denote the set of edges held by a point $p \in P$.

Given P and B, for each $p_x \in P$, we compute the B-NNs of p_x in $P \backslash \{p_x\}$ by using a kd-tree [6]. The B-NNs are maintained in $p.E$ and sorted in ascending order of weight (i.e., distance). Moreover, for each $p_x \in P$, we compute the triangle $\triangle_{x,y,z}$, where p_y and p_z are respectively the NN and 2-NN of p_x. This triangle is maintained in T, so T has at most n triangles (we remove duplicated triangles). Last, we sort the triangles in T in ascending order of weight.

Remark: Our offline algorithm needs $O(n^{1.5})$ time [26]. Let s_{avg} be the average number of edges held by each point. Building the spatial neighbor graph of P incurs $O(n(\sqrt{n} + s_{avg}))$ time. Our offline algorithm is hence cheaper, and it is general to any k and r.

C. ONLINE PROCESSING

To efficiently retrieve the top-k weighted spatial triangles, we consider *edge access order*. Let τ be an intermediate threshold of the top-k result (i.e., the weight of the intermediate top k-th triangle). From τ and triangle inequality, for any edges, we can obtain a weight θ that has to be satisfied to form the top-k weighted spatial triangles. That is, any triangles that have edges with weights larger than θ do not have to

Algorithm 1 Offline Processing

Require: P (set of points) and B (batch size)

1: $T \leftarrow \varnothing$ // a set of triangles
2: **for** each $p_x \in P$ **do**
3: $p_x.E \leftarrow \{e_{x,x'} \mid p_{x'} \in B$-NNs of p_x in $P \backslash \{p_x\}\}$ // edges are sorted in ascending order of weight
4: $T \leftarrow T \cup \{\triangle_{x,y,z}\}$ where p_y and p_z are respectively the NN and 2-NN of p_x
5: **end for**
6: Sort the triangles $\triangle \in T$ in ascending order of $w(\triangle)$

be enumerated. We exploit this observation along with the triangles in T and the B-NN graph obtained offline.

Algorithm 2 overviews our online algorithm. Let P_{cand} be the set of points that may form top-k triangles, and $P_{cand} = P$ at initialization. Our online algorithm has the following steps:

1) We first initialize the top-k result R and the threshold τ from the n triangles obtained offline in DETERMINE-THRESHOLD (P_{cand}, r). Then, from τ, we compute a threshold θ for edges. As seen later, any edges with weights larger than θ cannot form top-k triangles.
2) (If necessary, we update the B-NN graph by increasing B.) In REDUCE-CANDIDATES (P_{cand}, i, θ), we remove points with no edges satisfying θ any more from P_{cand}.
3) For each point in P_{cand}, we additionally enumerate triangles that could be in the top-k result and update R if necessary.
4) We repeat steps 2 and 3 until we have $P_{cand} = \varnothing$, and then R is returned.

Below, we detail steps 1, 2, and 3.

• *Step 1:* Recall that T is a sorted set of triangles obtained offline. Each triangle in T is formed by a point p, its NN, and 2-NN. (We remove all triangles in T that have edges with weights larger than r.) In DETERMINE-THRESHOLD (P_{cand}, r), we initialize R by the first k triangles in T, and τ is the weight of the k-th triangle. Let $\triangle_{x,y,z}$ be the k-th triangle. We set the threshold θ for edges as follows:

$$\theta = \tau - \max\{dist(p_x, p_y), dist(p_y, p_z), dist(p_x, p_z)\}. \quad (2)$$

This is used in the next step.

• *Step 2:* We next filter unnecessary points in P_{cand} by using θ. Let p_{x_j} be the j-th NN of p_x. Consider the i-th iteration of REDUCE-CANDIDATES (P_{cand}, i, θ). For $p_x \in P_{cand}$, if $w(e_{x,x_{i+2}}) > \theta$, triangles including $e_{x,x_{i+2}}$ can be ignored. (Recall that NN and 2-NN were considered in the offline processing.)

Proposition 1: For a point $p_x \in P_{cand}$, if $w(e_{x,x_{i+2}}) > \theta$, any triangles that have $e_{x,x_{i+2}}$ cannot be the top-k weighted spatial triangles.

Proof: See [26]. □

From this observation, we see that, if $w(e_{x,x_{i+2}}) > \theta$, all unseen triangles having p_x do not have to be enumerated and

Algorithm 2 Online Processing

Require: P (set of points), k (output size), r (distance threshold), B (batch size), and T (a set of triangles)
Ensure: R (set of k triangles with the minimum weight)

1: $b \leftarrow B$
2: $P_{cand} \leftarrow P$
3: $R \leftarrow$ DETERMINE-THRESHOLD (P_{cand}, r)
4: $\triangle_{x,y,z} \leftarrow$ triangle with the k-th smallest weight in R
5: $\tau \leftarrow w(\triangle_{x,y,z})$
6: $\theta \leftarrow \tau - \max\{dist(p_x, p_y), dist(p_y, p_z), dist(p_x, p_z)\}$
7: $i \leftarrow 1$
8: **while** $P_{cand} \neq \varnothing$ **do**
9: **if** $i = b - 1$ **then**
10: $b \leftarrow b + B$
11: Build b-NN graph of P_{cand}
12: **end if**
13: REDUCE-CANDIDATES (P_{cand}, i, θ)
14: $R \leftarrow$ ENUMERATE-TRIANGLES (P_{cand}, r, i)
15: Execute lines 4–6
16: $i \leftarrow i + 1$
17: **end while**

p_x can be safely removed from P_{cand}. REDUCE-CANDIDATES (P_{cand}, i, θ) does this point removal.

The triangles enumerated offline practically have small weights, as they are based on NN and 2-NN. Therefore, τ and θ are tight even when i is small, and we can effectively reduce the size of P_{cand} in early iterations.

Example 3: We use Figure 2 to understand our point filtering. Assume that DETERMINE-THRESHOLD (P_{cand}, r) returns the triangle formed by the red edges, and θ is also obtained as depicted in this figure. Focus on p_x and p_y that are described by green and blue, respectively. The edge between p_x (p_y) and its 3-NN is described by the same color, and its weight is shown in the right part of this figure. We have $w(e_{x,x_3}) > \theta$ and $w(e_{y,y_3}) > \theta$, and unseen triangles that have p_x or p_y cannot be the top-k result. Therefore, we can remove them from P_{cand}.

• *Step 3:* After filtering unnecessary points in the above step, we enumerate triangles that may become the top-k result in ENUMERATE-TRIANGLES (P_{cand}, r, i). Consider the i-th iteration of this step. For each $p_x \in P_{cand}$, we enumerate triangles formed by $p_x, p_{x_{i+2}}$, and p_{x_j}, where $j \in [1, \ldots, i+1]$, while updating the top-k result R, τ, and θ.

W.r.t. p_{x_j}, we access it in order of $p_{x_1}, \ldots, p_{x_{i+1}}$. Then, it is important to notice that $w(e_{x,x_j}) + w(e_{x,x_{i+2}})$ monotonically increases. When we have $w(e_{x,x_j}) + w(e_{x,x_{i+2}}) \geq \tau$, we see that triangles with these edges cannot be the top-k result, thus we can stop enumerating triangles without losing correctness.

Analysis: Let n_i be the size of P_{cand} at the i-th iteration of step 2. In addition, let n'_i be the size of P_{cand} at the i-th iteration of step 3. Our online algorithm needs $O(\sum_{i=1}^{I}(n_i + i \cdot n'_i))$ time, where I is the number of iterations of step 3. (The detail appears in [26].) In Section VI-A, we show that our algorithm

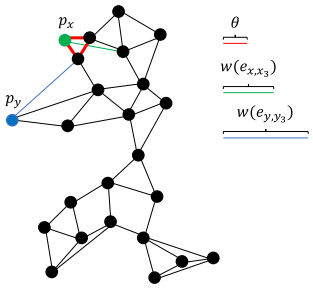


FIGURE 2. Illustration of REDUCE-CANDIDATES (P_{cand}, i, θ).

has a small n'_i and I in practice, yielding $O(\sum_{i=1}^{I}(n_i+i\cdot n'_i)) \approx O(n)$. This suggests that our algorithm practically beats any approaches that build the spatial neighbor graph of P, as they need at least $\Omega(n^{1+\epsilon})$ time where $\epsilon > 0$.

IV. OUR SOLUTION FOR FULLY DYNAMIC DATA

We next consider the case where P is subjective to updates (point insertions and deletions), and address the problem defined in Definition 4. In this case, the top-k result R may change because of the update of P. We below consider how to minimize the result update cost while keeping the correct answer, and show that our approach in Section III-C can actually deal with point insertions and deletions flexibly. Hereinafter, we assume that the top-k result R is initialized by our algorithm in the previous section.

A. INSERTION CASE

Assume that we have a new point p_x. It is important to note that *triangles which can newly become the top-k result are limited to the ones having* p_x. We use this observation to incrementally update the top-k result.

1) Given p_x, we run a range search on a kd-tree where its query point is p_x and radius is r, to update the B-NN graph. (Observe that the points whose B-NNs may be updated exist within the distance r from p_x, due to the constraint of r.) For each point p_y in this range search result, if p_x becomes a new B-NN of p_y, we add p_x into the edge set $p_y.E$. Also, for p_x, we make $p_x.E$ from this range search result.
2) We next consider the triangle $\triangle_{x,x_1,x_2}$. If $w(e_x, e_{x_1}) > \theta$ or $w(e_x, e_{x_2}) > \theta$, the weights of new triangles having p_x are larger than τ. Hence, we terminate the update.
3) Otherwise, we run lines 7–16 of Algorithm 2 by setting $P_{cand} = \{p_x\}$.

The main cost of this case is incurred by the range search, which needs $O(\sqrt{n}+s)$, where s is the size of the range search result. The second operation needs $O(1)$ time. Also, the third operation needs a trivial cost $\ll O(\sqrt{n})$ because it has a few iteration numbers in practice.

B. DELETION CASE

We next assume that a point p_x is removed from P. We have two cases incurred by this point removal.

1) NO TRIANGLES ARE REMOVED FROM R

If no triangles having p_x are in the top-k result, it is trivial to see that the top-k result does not change. In this case, we simply remove the edges corresponding to p_x from the B-NN graph.

2) SOME TRIANGLES ARE REMOVED FROM R

In this case, we need to update the top-k result. Note that *this case is essentially the same as the static case*, because R has less than k triangles. Therefore, to update R, we update the B-NN graph, update R via DETERMINE-THRESHOLD (P, r),[1] and then verify R through lines 4–16 of Algorithm 2.

Clearly, the former case needs $O(1)$ time. The cost of the latter case is the same as our online algorithm in Section III-C. It is intuitively seen that *the latter case rarely occurs* for datasets with a large n. This implies that the amortized update cost for a deletion can come close to the former cost.

V. OUR SOLUTION FOR SLIDING-WINDOW MODEL

This section addresses the problem in Definition 5. Different from the fully dynamic case in Section IV, we need to consider insertion and deletion at the same time in the sliding-window model. This is because a window slide removes the oldest point and inserts a new point. Therefore, under this model, the top-k result has to be updated when

1) the weights of triangles having a new point are less than the threshold of the current top-k result and
2) the removed point has triangles included in the current top-k result.

To efficiently deal with these cases, we maintain the following triangle for each point $p \in P_W$. (Recall that P_W is a set of points in the current window.)

Definition 6 ($\triangle_p^{\min}$): Consider a point $p \in P_W$, and $\triangle_p^{\min}$ represents the triangle that has the minimum weight among the set of triangles having p but not being included in the top-k result.

Although $\triangle_p^{\min}$ may be updated when the window slides, it supports efficient top-k result update. For case 1), we can focus only on points p having $w(\triangle_p^{\min}) < \tau$ (recall that τ is the threshold of the top-k result) and can ignore the other points. For case 2), by adding $\triangle_p^{\min}$ into an intermediate top-k result, we can obtain a tight τ, which also supports pruning unnecessary points. Since we maintain only a single triangle $\triangle_p^{\min}$ for each point $p \in P_W$, the space complexity is only $O(W)$. Below, we show how to maintain $\triangle_p^{\min}$ when p_x is removed from and is added to the window.

A. DEALING WITH REMOVED POINT

When p_x is removed from the window, we confirm whether $p_x \in \triangle^{\min}$ of some points in P_W. Let $N(p_x)$ be a set of neighbors of p_x. If $p_x \in \triangle_y^{\min}$ for a point $p_y \in P_W$, we have

[1] In this dynamic data case, for each point $p \in P$, we enumerate the triangle p, its NN, and 2NN *online*, because it may be updated by past and current point insertions and deletions.

Algorithm 3 Removal

Require: P_W, p_x (removed point), and R
1: **if** R contains triangles having p_x **then**
2: Remove all such triangles from R
3: **end if**
4: **for** each $p_y \in N(p_x)$ such that $p_x \in \triangle_y^{\min}$ **do**
5: $\triangle_y^{\min} \leftarrow$ UPDATE-$\triangle^{\min}(P_W, p_y)$
6: **end for**

Algorithm 4 Insertion

Require: P_W, p_x (new point), and r
1: $\triangle_x^{\min} \leftarrow$ UPDATE-$\triangle^{\min}(P_W, p_x)$
2: **for** each $p_y \in N(p_x)$ **do**
3: $\triangle_y^{\min} \leftarrow$ UPDATE-$\triangle^{\min}(P_W, p_y)$
4: **end for**

to update $\triangle_y^{\min}$. To achieve this, we need to enumerate triangles containing p_y, and this can be done by essentially the same operation in step 3 of our algorithm for static data, see Section III-C.

Algorithm 3 describes how to deal with p_x when it is removed from the window. We first remove invalid triangles from the current top-k result R. Then, we update $\triangle_y^{\min}$ for each $p_y \in N(p_x)$ in the way explained above, which corresponds to UPDATE-$\triangle^{\min}(P_W, p_y)$.

B. DEALING WITH NEW POINT

When a new point p_x is inserted into the window, we evaluate whether triangles having p_x and p_y can be $\triangle_y^{\min}$ for each $p_y \in N(p_x)$. (We retrieve $N(p_x)$ through a range search.) Algorithm 4 describes this procedure. We first compute $\triangle_x^{\min}$. Then, for each $p_y \in N(p_x)$, we update $\triangle_y^{\min}$ if necessary. How to enumerate triangles follows the same way as in Section V-A.

C. TOP-k RESULT UPDATE

Recall that we need to update the top-k result R when (i) $w(\triangle_x^{\min}) < \tau$ where p_x is a new point and (ii) triangles having p_y, where p_y is a removed point, are included in R. Taking into account this fact, we update R in the following three steps, which are summarized in Algorithm 5.

1) We first remove invalid triangles from R in Algorithm 3. Then, if $|R| < k$, we add $(k - |R|)$ triangles $\triangle^{\min}$ with the minimum $w(\triangle^{\min})$ to R to obtain an intermediate top-k result with a (probably) tight threshold.
2) If $\triangle_z^{\min}$ was inserted into R in the previous step, there may exist other triangles $\triangle$ having p_z such that $w(\triangle) < \tau$. We hence enumerate such triangles and update $\triangle_z^{\min}$ and R in lines 11–12.
3) Last, due to the update of τ, there may exist other points $p \in P_W$ such that $w(\triangle_p^{\min}) < \tau$. If so, we do the same operations in the second step for p in lines 16–20.

Algorithm 5 Update-Top-k

Require: P_W, r, T (set of W triangles with $\triangle^{\min}$) and R (the current top-k result)
Ensure: R
1: Run Algorithms 3 and 4 in order
2: Sort the triangles $\triangle^{\min} \in T$ in ascending order of $w(\triangle^{\min})$
3: $l \leftarrow k - |R|$
4: **if** $l > 0$ **then**
$R \leftarrow R \cup \{l$ triangles with the smallest weight in $T\}$
5: **end if**
6: $\triangle_{x,y,z} \leftarrow$ triangle with the k-th smallest weight in R
7: $\tau \leftarrow w(\triangle_{x,y,z})$
8: $i \leftarrow 1$
9: **while** $i \leq l$ **do**
10: $p \leftarrow$ point having the triangle with the i-th smallest weight in T
11: $R \leftarrow$ UPDATE-TOP-k-TRIANGLES(P_W, p)
12: $\triangle_p^{\min} \leftarrow$ UPDATE-$\triangle_p^{\min}(P_W, p)$
13: Execute lines 6–7
14: $i \leftarrow i + 1$
15: **end while**
16: $\triangle_q^{\min} \leftarrow$ triangle with the smallest weight in T
17: $i \leftarrow 1$
18: **while** $w(\triangle_q^{\min}) < \tau$ **do**
19: Execute lines 11–14
20: $\triangle_q^{\min} \leftarrow$ triangle with the i-th smallest weight in T
21: **end while**

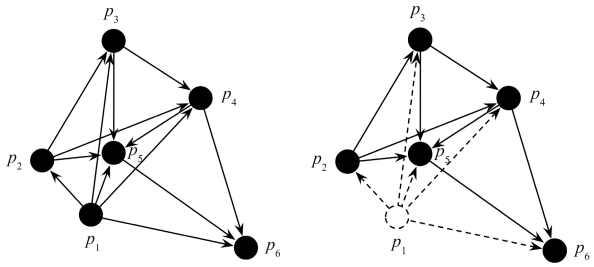
The number of triangles enumerated in Algorithm 5 cannot be bounded (and the worst case can be similar to our static algorithm) because it depends on data distributions. Nevertheless, it is practically small because the top-k result does not change so frequently. In Section VI-C, we show that Algorithm 5 never reaches the worst case.

D. OPTIMIZATION

Assume that, for a point $p_a \in P_W$, $\triangle_a^{\min} = \triangle_{a,b,c}$. It is possible that $\triangle_b^{\min} = \triangle_{a,b,c}$ and $\triangle_c^{\min} = \triangle_{a,b,c}$. If $\triangle_{a,b,c}$ is newly included in the top-k result, we have to update $\triangle_a^{\min}$, $\triangle_b^{\min}$, and $\triangle_c^{\min}$. Furthermore, if p_a is removed from the window, we have to update $\triangle_b^{\min}$ and $\triangle_c^{\min}$. These degrade the performance of Algorithm 5. To avoid such redundancy, we employ a *directed* spatial neighbor graph.

*Definition 7 (*Directed Spatial Neighbor Graph*): Assume that points in P_W are maintained by the generation order, and $o(p_i) \prec o(p_j)$ shows that p_i was generated before p_j. Then, given P_W and r, in the directed spatial neighbor graph of P_W, there is a direct edge $e_{i,j}$ between p_i and p_j if and only if $dist(p_i, p_j) < r$ and $o(p_i) \prec o(p_j)$, where $w(e_{i,j}) = dist(p_i, p_j)$.*

From this definition, hereinafter, $N(p_i)$ is also re-defined as a set of points p_j such that $dist(p_i, p_j) < r$ and $o(p_i) \prec o(p_j)$. Below, we present why this structure can remove the redundancy.



(a) Directed spatial neighbor graph (b) Removal of p_1

FIGURE 3. Example of the case where p_x is removed.

1) WHEN p_x IS REMOVED

Recall that the sliding-window model removes the oldest point, so it is important to notice that $o(p_x) \prec o(p)$ for every $p \in P_W$. We then see that $p_x \notin N(p)$ and $\triangle_p^{\min}$ never contains p_x for every $p \in P_W$. Therefore, when p_x is removed, we do not have to update $\triangle_p^{\min}$ for every $p \in P_W$.

Example 4: We explain this observation by using Figure 3. Figure 3(a) illustrates a directed spatial neighbor graph consisting of $P_W = \{p_1, p_2, p_3, p_4, p_5, p_6\}$, where $o(p_i) \prec o(p_j)$ for $i < j$. Now assume that the window slides and p_1 is removed. As shown in Figure 3(b), the other points do not have direct edges to p_1 and do not change $N(p)$. Hence, $\triangle_p^{\min}$ also does not change.

2) WHEN p_y IS ADDED

In this case, we update the directed spatial neighbor graph by using a range search. Then, for each $p_x \in P_W$ such that $dist(p_x, p_y) < r$, we update $\triangle_y^{\min}$ by enumerating triangles that have both p_x and p_y (if necessary). Note that we have $\triangle_y^{\min} \neq \triangle_z^{\min}$ for p_y and p_z such that $o(p_y) \prec o(p_z)$, since $N(p_z)$ does not contain p_y.

3) TOP-k RESULT UPDATE

Thanks to the above optimization, we have no duplication w.r.t. $\triangle^{\min}$ thus can avoid unnecessary triangle enumerations. We incorporate this optimization into Algorithms 3–5.

VI. EXPERIMENT

For experiments, we used a Ubuntu machine equipped with 3.6GHz Intel Core i9-9900K CPU and 128GB RAM. In addition, all algorithms were compiled by g++ 9.3.0 with −O3 flag and ran in a single thread mode.

A. EVALUATION ON STATIC DATA

This section evaluates our algorithm for static data. We compared it with DHL [17], which can compute the exact answer from the spatial neighbor graph of P. As mentioned in Sections I-A and VII, DHL is the only existing algorithm that can deal with our problem. For DHL, we used the original implementation.[2]

[2]https://github.com/raunakkmr/Retrieving-top-weighted-triangles-in-graphs

TABLE 2. Impact of r on our algorithm.

	Computation time [sec]	
r	CaStreet	Places
0.01	0.10	0.16
0.015	0.09	0.17
0.02	0.09	0.16
0.025	0.11	0.17

1) DATASET

We used two real large datasets, CaStreet[3] and Places,[4] to investigate how efficiently our algorithm runs on large datasets. Recall that one of our objectives is to design an efficient (and exact) algorithm for the problem defined in Definition 3. CaStreet consists of the minimum bounding rectangles of road segments in the U.S.A. We used bottom-left and upper-right points, and its cardinality is 4,499,454. Places consists of the geo-locations of public places in the U.S.A, and its cardinality is 9,356,750.

2) PARAMETER

We set $n = 1,000,000$ (via random sampling), $k = 100$, and $r = 0.01$ by default.

3) IMPACT OF BATCH SIZE B

We first empirically tune the batch size (a hyper-parameter), because it can affect the efficiency of our algorithm. When $B = 5$, the running time was 0.07 (0.18) [sec] on CaStreet (Places). Also, when $B = 10$ and $B = 15$, on CaStreet (Places), those were respectively 0.10 and 0.10 (0.16 and 0.17) [sec]. From this result, it can be seen that the running time of our algorithm is almost not affected by B. This means that the top-k result can be retrieved from a B-NN graph, where B is small. This result justifies our idea in Section III-A, i.e., it is not necessary to build the spatial neighbor graph of P correctly. From the result, we set $B = 10$ in the remaining experiments.

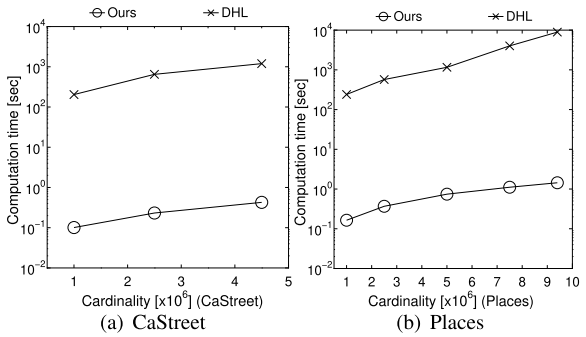
4) IMPACT OF r

Note that as r increases, the number of neighbors also increases. Table 2 shows the result of our experiment with varying r. The computation time of our algorithm is essentially the same even when we use a larger r than the default one. This result shows the robustness of our algorithm against r.

5) OFFLINE TIME

We report the offline time of our algorithm at the default parameter. On CaStreet and Places, our offline algorithm took 21.05 and 26.54 seconds, respectively. Since our offline algorithm is general for any k and r, the offline time is reasonable. (Actually, even if our algorithm begins from offline computation, it took less time to return the answer than DHL.)

[3]http://chorochronos.datastories.org/?q=node/59
[4]https://archive.org/details/2011-08-SimpleGeo-CC0-Public-Spaces



(a) CaStreet (b) Places

FIGURE 4. Impact of cardinality of dataset.

TABLE 3. Statistics of our algorithm in each iteration on Places ($k = 100$).

	$n = 1,000,000$	
Iteration	$\lvert P_{cand}\rvert$	#triangles enumerated
$i = 1$	102	203
$i = 2$	18	54
$i = 3$	4	16
$i = 4$	0	0
	$n = 9,356,750$	
Iteration	$\lvert P_{cand}\rvert$	#triangles enumerated
$i = 1$	316	610
$i = 2$	30	84
$i = 3$	4	16
$i = 4$	-	-

6) IMPACT OF n

Figure 4 studies the scalability of our algorithm to the cardinality of dataset n. Our algorithm has a linear scalability to n, while DHL is superlinear w.r.t. n. This clarifies the advantage of our algorithm. When we used all points of CaStreet and Places, our algorithm is 2807 and 6193 times faster than DHL on CaStreet and Places, respectively.

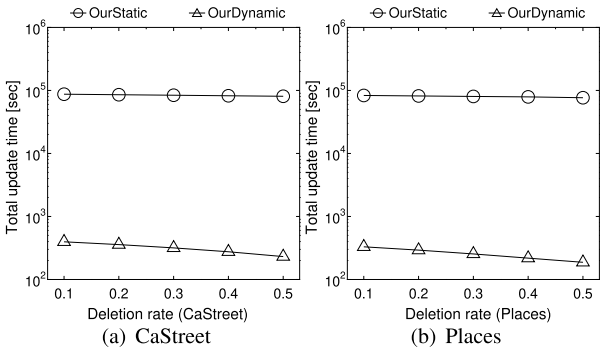
To understand the linear scalability of our algorithm, we investigated the size of P_{cand} and the number of triangles enumerated in each iteration. Table 3 shows the result on Places when $n = 1,000,000$ and $n = 9,356,750$. (We omit the result on CaStreet, because it is similar to the one in Table 3.) It is important to note that the numbers of iterations and triangles enumerated are both very small. This also clarifies the effectiveness of our idea. Recall that the time complexity of our online algorithm is $O(\sum_{i=1}^{I}(n_i + i \cdot n'_i))$. In practice, I and n'_i are sufficiently small. In addition, when $i \geq 2$, $n_i = n'_{i-1}$ and n_i is also sufficiently small. Notice that $n_1 = n$, then we have $O(\sum_{i=1}^{I}(n_i + i \cdot n'_i)) \approx O(n)$. Now it is clear why we have the linear scalability.

B. EVALUATION ON FULLY DYNAMIC DATA

This section evaluates our solution for fully dynamic data. Because no existing works have addressed this problem so far, we compared our solution with our static algorithm that computes the result from scratch whenever we have an update.

1) DATASET

We used the same datasets as the ones in Section VI-A, and we used 1,000,000 points for initialization.



(a) CaStreet (b) Places

FIGURE 5. Impact of deletion rate.

2) WORKLOAD

We used 10,000 updates as a workload. This workload consisted of $(1-\alpha)\times 10,000$ insertions and $\alpha \times 10,000$ deletions. (Given an update is a deletion, we removed a random point in P.) To investigate the result update efficiency of our solution, we conducted experiments with varying α (i.e., deletion rate). We set $k = 100$.

3) RESULT

We measured the time to complete the workload, and Figure 5 depicts the result. Due to the incremental update, our algorithm for dynamic data, which is represented by "OurDynamic", completes the workload significantly faster than the algorithm for static data (represented by "OurStatic"). For example, in the case of CaStreet and $\alpha = 0.1$, OurDynamic completes the workload in about 400 seconds. Its average update time per an update is hence about 40 milliseconds, whereas that of OurStatic is about 9000 milliseconds.[5]

When α is larger, the performance difference becomes more bigger. We see that, as α increases, OurDynamic needs less time to complete the workload. In most deletion cases, the top-k result did not change, meaning that OurDynamic incurs only $O(1)$ time in each of these cases. We had these cases more as the deletion rate increases, thus its time becomes shorter.

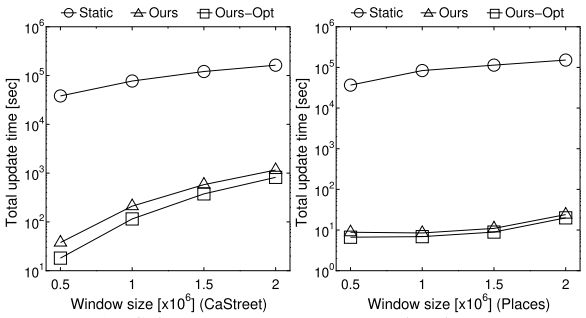
C. EVALUATION ON SLIDING-WINDOW MODEL

Last, we evaluate our algorithms for the sliding-window model. This problem also has no existing works, so we compared our algorithms with our static algorithm that computes the result from scratch whenever the window slides. We use "Ours", "Ours-Opt", and "Static" to respectively denote Algorithm 5 without the optimization in Section V-D, Algorithm 5 with the optimization, and the static algorithm.

1) DATASET

We used the same datasets in Section VI-B.

[5]This is slower than the computation time in the case of static data. When datasets are static, for each point $p \in P$, OurStatic can enumerate the triangle consisting of p, its NN, and 2NN *offline*. However, when datasets are dynamic, it needs to do this *online*, because these triangles may be updated.



(a) CaStreet (b) Places

FIGURE 6. Impact of window size.

2) WORKLOAD

After the first W points were contained in the window, we ran 10,000 slides. We set $r = 0.1$ and $k = 100$.

3) RESULT

We measured the total time to deal with 10,000 window slides. Figure 6 shows the result of experiments with different window sizes. We observe that larger W needs a longer time. However, Ours and Ours-Opt keep short update time. When the window size is 1,000,000, Ours is about 700 (10,000) times faster than Static on CaStreet (Places). Furthermore, even when the window size is 2 million, Ours-Opt needs only 82 [msec] and 2 [msec] on average to update the top-k result per slide on CaStreet and Places, respectively, suggesting that it scales well to large window sizes. It is also seen that Ours-Opt is always faster than Ours, thanks to the optimization.

VII. RELATED WORK

This section reviews existing works that relate to the problem of retrieving the top-k weighted spatial triangles.

A. GRAPH-BASED SPATIAL DATA MINING

Graph is a simple yet effective structure for representing relationships between data. Spatial points usually have relationships if they locate in close positions. Therefore, graph-based spatial data mining has been receiving attention. (Note that our work is different from works for road networks, e.g., [8], because these assume that graphs are given and P is constrained by the road networks.)

Literature [12] considers spatial pattern matching. Given P and a query that is a graph pattern, it finds all subsets of P that matches the query. Different from our problem, this spatial pattern matching requires to specify a sub-graph of P. Clearly, the graph structure of P is not pre-known, so it is not an easy task to specify a concrete query. Moreover, the query result size is not controllable. Literature [13] considers a top-k version of spatial pattern matching, but it still has the former drawback. Spatial maximal clique in the spatial neighbor graph of P is considered in [34]. Since a triangle is a 3-clique, this problem is similar to ours. The authors of [34] found that the finding a spatial maximal clique corresponds to doing a maximal convex polygon. Their solution is based on this observation, and they do not consider the weight of polygons. Therefore, their technique cannot be employed for finding the top-k weighted spatial triangles.

Given a set W of location-based service providers and a set U of users with locations, [27] tackled the bipartite matching between W and U. Unlike the above works that try to ''mine interesting sub-graphs'', this problem focuses on ''building a graph''. Recently, [30] designed a system that builds spatial proximity graphs (e.g., a k-NN graph and Delaunay graph) from a given set P of points for multicore processors. It also supports other operations, such as clustering and computing minimum spanning trees on spatial proximity graphs. However, retrieving the top-k weighted spatial triangles is not supported, and we are the first to study this problem in spatial databases.

B. SPATIAL DATA ANALYSIS

Because spatial point analysis is well known to be important, much efforts have been made to develop query processing techniques, machine-learning models [23], [29], and systems [32], [33]. We below review some examples of analytical techniques.

The problem of maximizing range sum queries was addressed in [11]. Given a rectangle, this problem finds the location of the rectangle that maximizes the weight of points enclosed by the rectangle. A streaming version of this problem was also considered in [2], [3]. Such location selection problems have been extensively studied, e.g., in [15]. The interaction between spatial points was addressed in [4]. Some works considered spatial data visualization. In [16], to achieve interactive visualization of spatial points, the authors proposed an efficient algorithm that incrementally updates the visualization result from the previous one. Moreover, [7] proposed an efficient bounding technique for kernel density visualization.

C. TRIANGLE ENUMERATION/COUNTING

Because the problem of triangle enumeration/counting is one of the classic problems in graph databases, many works tackled it. State-of-the-art algorithms for static and dynamic graphs can be found in [1], [18], [24], [25], [31]. Unfortunately, existing works for graph databases generally assume unweighted graphs and do not consider any ranking of triangles.

Similarly to our problem, DHL addressed the problem of retrieving the top-k weighted triangles in graph databases. It was originally proposed for weighted graphs, so it can deal with our problem by building the spatial neighbor graph of P. (DHL originally retrieves k triangles with the maximum weight, but it is straightforward to focus on triangles with the minimum weight.) However, because of the overhead incurred by dealing with the spatial neighbor graph, DHL is significantly outperformed by our algorithm.

VIII. CONCLUSION

This paper addressed the problems of retrieving and monitoring top-k weighted spatial triangles. As there are many

triangles in a spatial neighbor graph, enumerating all triangles is costly. We hence proposed an efficient algorithm that returns the exact answer. Based on this algorithm, we showed how to deal with fully dynamic data. Furthermore, we designed an algorithm for streaming data in a sliding-window model. The results of our experiments on real datasets demonstrate the efficiencies of our algorithms for static and dynamic data.

REFERENCES

[1] M. A. Hasan and V. S. Dave, "Triangle counting in large networks: A review," *WIREs Data Mining Knowl. Discovery*, vol. 8, no. 2, p. e1226, Mar. 2018.
[2] D. Amagata and T. Hara, "Monitoring MaxRS in spatial data streams," in *Proc. EDBT*, 2016, pp. 317–328.
[3] D. Amagata and T. Hara, "A general framework for MaxRS and MaxCRS monitoring in spatial data streams," *ACM Trans. Spatial Algorithms Syst.*, vol. 3, no. 1, pp. 1–34, Mar. 2017.
[4] D. Amagata and T. Hara, "Identifying the most interactive object in spatial databases," in *Proc. IEEE 35th Int. Conf. Data Eng. (ICDE)*, Apr. 2019, pp. 1286–1297.
[5] D. Amagata, S. Tsuruoka, Y. Arai, and T. Hara, "Feat-SKSJ: Fast and exact algorithm for top-k spatial-keyword similarity join," in *Proc. 29th Int. Conf. Adv. Geographic Inf. Syst.*, Nov. 2021, pp. 15–24.
[6] J. L. Bentley, "Multidimensional binary search trees used for associative searching," *Commun. ACM*, vol. 18, no. 9, pp. 509–517, 1975.
[7] T. N. Chan, R. Cheng, and M. L. Yiu, "QUAD: Quadratic-bound-based kernel density visualization," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, Jun. 2020, pp. 35–50.
[8] T. N. Chan, Z. Li, L. H. U, J. Xu, and R. Cheng, "Fast augmentation algorithms for network kernel density visualization," *Proc. VLDB Endowment*, vol. 14, no. 9, pp. 1503–1516, May 2021.
[9] S. Chatterjee, M. Connor, and P. Kumar, "Geometric minimum spanning trees with GEOFILTERKRUSKAL," in *Proc. Int. Symp. Exp. Algorithms*, 2010, pp. 486–500.
[10] Y. Chen, Z. Chen, G. Cong, A. R. Mahmood, and W. G. Aref, "SSTD: A distributed system on streaming spatio-textual data," *Proc. VLDB Endowment*, vol. 13, no. 12, pp. 2284–2296, Aug. 2020.
[11] D.-W. Choi, C.-W. Chung, and Y. Tao, "A scalable algorithm for maximizing range sum in spatial databases," *Proc. VLDB Endowment*, vol. 5, no. 11, pp. 1088–1099, Jul. 2012.
[12] Y. Fang, R. Cheng, G. Cong, N. Mamoulis, and Y. Li, "On spatial pattern matching," in *Proc. IEEE 34th Int. Conf. Data Eng. (ICDE)*, Apr. 2018, pp. 293–304.
[13] Y. Fang, Y. Li, R. Cheng, N. Mamoulis, and G. Cong, "Evaluating pattern matching queries for spatial databases," *VLDB J.*, vol. 28, no. 5, pp. 649–673, Oct. 2019.
[14] Y. Fang, Z. Wang, R. Cheng, X. Li, S. Luo, J. Hu, and X. Chen, "On spatial-aware community search," *IEEE Trans. Knowl. Data Eng.*, vol. 31, no. 4, pp. 783–798, Apr. 2019.
[15] K. Feng, G. Cong, C. S. Jensen, and T. Guo, "Finding attribute-aware similar regions for data analysis," *Proc. VLDB Endowment*, vol. 12, no. 11, pp. 1414–1426, Jul. 2019.
[16] T. Guo, K. Feng, G. Cong, and Z. Bao, "Efficient selection of geospatial data on maps for interactive and visualized exploration," in *Proc. Int. Conf. Manage. Data*, May 2018, pp. 567–582.
[17] R. Kumar, P. Liu, M. Charikar, and A. R. Benson, "Retrieving top weighted triangles in graphs," in *Proc. 13th Int. Conf. Web Search Data Mining*, Jan. 2020, pp. 295–303.
[18] D. Lee, K. Shin, and C. Faloutsos, "Temporal locality-aware sampling for accurate triangle counting in real graph streams," *VLDB J.*, vol. 29, no. 6, pp. 1501–1525, Nov. 2020.
[19] S. Nishio, D. Amagata, and T. Hara, "Geo-social keyword top-k data monitoring over sliding window," in *Proc. DEXA*, 2017, pp. 409–424.
[20] S. Nishio, D. Amagata, and T. Hara, "Lamps: Location-aware moving top-k pub/sub," *IEEE Trans. Knowl. Data Eng.*, vol. 34, no. 1, pp. 352–364, Jan. 2022.
[21] V. Pandey, A. Kipf, T. Neumann, and A. Kemper, "How good are modern spatial analytics systems? *Proc. VLDB Endowment*, vol. 11, no. 11, pp. 1661–1673, 2018.
[22] H.-M. Park, S.-H. Myaeng, and U. Kang, "PTE: Enumerating trillion triangles on distributed systems," in *Proc. 22nd ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, Aug. 2016, pp. 1115–1124.
[23] J. Qi, G. Liu, C. S. Jensen, and L. Kulik, "Effectively learning spatial indices," *Proc. VLDB Endowment*, vol. 13, no. 12, pp. 2341–2354, Aug. 2020.
[24] P. Ribeiro, P. Paredes, M. E. P. Silva, D. Aparicio, and F. Silva, "A survey on subgraph counting: Concepts, algorithms, and applications to network motifs and graphlets," *ACM Comput. Surv.*, vol. 54, no. 2, pp. 1–36, Mar. 2022.
[25] K. Shin, S. Oh, J. Kim, B. Hooi, and C. Faloutsos, "Fast, accurate and provable triangle counting in fully dynamic graph streams," *ACM Trans. Knowl. Discovery Data*, vol. 14, no. 2, pp. 1–39, Apr. 2020.
[26] R. Taniguchi, D. Amagata, and T. Hara, "Efficient retrieval of top-k weighted spatial triangles," in *Proc. DASFAA*, 2022, pp. 224–231.
[27] Y. Tong, J. She, B. Ding, L. Chen, T. Wo, and K. Xu, "Online minimum matching in real-time spatial data: Experiments and analysis," *Proc. VLDB Endowment*, vol. 9, no. 12, pp. 1053–1064, 2016.
[28] S. Tsuruoka, D. Amagata, S. Nishio, and T. Hara, "Distributed spatial-keyword kNN monitoring for location-aware pub/sub," in *Proc. 28th Int. Conf. Adv. Geographic Inf. Syst.*, Nov. 2020, pp. 111–114.
[29] T. Vu, A. Belussi, S. Migliorini, and A. Eldway, "Using deep learning for big spatial data partitioning," *ACM Trans. Spatial Algorithms Syst.*, vol. 7, no. 1, pp. 1–37, Mar. 2021.
[30] Y. Wang, S. Yu, L. Dhulipala, Y. Gu, and J. Shun, "GeoGraph: A framework for graph processing on geometric data," *ACM SIGOPS Operating Syst. Rev.*, vol. 55, no. 1, pp. 38–46, Jun. 2021.
[31] X. Yang, C. Song, M. Yu, J. Gu, and M. Liu, "Distributed triangle approximately counting algorithms in simple graph stream," *ACM Trans. Knowl. Discovery Data*, vol. 16, no. 4, pp. 1–43, Aug. 2022.
[32] J. Yu and M. Sarwat, "GeoSparkViz: A cluster computing system for visualizing massive-scale geospatial data," *VLDB J.*, vol. 30, no. 2, pp. 237–258, Mar. 2021.
[33] J. Yu, J. Wu, and M. Sarwat, "GeoSpark: A cluster computing framework for processing large-scale spatial data," in *Proc. 23rd SIGSPATIAL Int. Conf. Adv. Geographic Inf. Syst.*, Nov. 2015, pp. 1–4.
[34] C. Zhang, Y. Zhang, W. Zhang, L. Qin, and J. Yang, "Efficient maximal spatial clique enumeration," in *Proc. IEEE 35th Int. Conf. Data Eng. (ICDE)*, Apr. 2019, pp. 878–889.

RYOSUKE TANIGUCHI is currently pursuing the master's degree with the Department of Multimedia Engineering, Graduate School of Information Science and Technology, Osaka University, Osaka, Japan. His research interest includes spatial query processing.

DAICHI AMAGATA (Member, IEEE) received the B.E., M.Sc., and Ph.D. degrees from Osaka University, Osaka, Japan, in 2012, 2014, and 2015, respectively. He is currently an Assistant Professor with the Department of Multimedia Engineering, Graduate School of Information Science and Technology, Osaka University. His research interests include fast algorithms for databases and AI technologies.

TAKAHIRO HARA (Senior Member, IEEE) received the B.E., M.E., and Dr.E. degrees in information systems engineering from Osaka University, Osaka, Japan, in 1995, 1997, and 2000, respectively. Currently, he is a Full Professor with the Department of Multimedia Engineering, Osaka University. His research interests include distributed databases, peer-to-peer systems, mobile networks, and mobile computing systems. He is a Distinguished Scientist of ACM and a member of three other learned societies.