Efficient Retrieval of Top-k Weighted Spatial Triangles

Ryosuke Taniguchi, Daichi Amagata, and Takahiro Hara

Osaka University, Osaka, Japan
{taniguchi.ryosuke, amagata.daichi, hara}@ist.osaka-u.ac.jp

Abstract. Due to the proliferation of location-based services and IoT devices, a lot of spatial points are being generated. Spatial data analysis is well known to be an important task. As spatial data analysis tools, 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, have been receiving much attention. We focus on triangles (one of the simplest sub-graph patterns) in such graphs and address the problem of retrieving the top-k weighted spatial triangles. This problem has important real-life applications, e.g., group search, urban planning, and co-location pattern mining. However, 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 solve this challenge, we propose an efficient algorithm that returns the exact result. Our experimental results on real datasets show the efficiency of our algorithm.

Keywords: Spatial points · Weighted graph · Top-k retrieval.

1 Introduction

Due to the proliferation of location-based services and IoT devices, a lot of spatial (or geo-location) points are being generated nowadays. Analyzing such spatial points yields useful observations. Many spatial point processing techniques [1–4, 9] and systems [10, 13, 15] have therefore been devised. Recently, as spatial point analysis tools, graph-based approaches have been receiving attention [6, 14, 16].

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. Triangles are particularly considered in graph contexts, because triangle is one of the simplest yet important primitive sub-graph patterns (e.g., clique) having many applications [8, 11]. For example, spatial triangles can be utilized in group search [7], co-location pattern mining [16], and urban planning [6]. Note that the number of triangles in a spatial neighbor graph is generally

huge. Enumerating all of them is therefore not feasible, and the output size should be controllable (by a user-specified parameter k) [8]. In spatial databases, given a subset of points in P (e.g., that form triangles), the cohesiveness of the subset is a factor in measuring its importance [16].

Motivated by the above applications and observations, this paper addresses 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 $dist(p_x, p_y) + dist(p_y, p_z) + dist(p_x, p_{zy})$, where $dist(\cdot, \cdot)$ measures the Euclidean distance between two points, which takes into account the cohesiveness. Then, given P and the output size k, this problem retrieves k spatial triangles with the minimum weight among all triangles in the spatial neighbor graph of P. This problem is computationally challenging, as seen below. A straightforward solution for this problem is to enumerate all triangles and then output k triangles with the minimum weight. The number of triangles in the spatial neighbor graph is $O(\binom{n}{2})$, where $n = |P|$, so this solution is not feasible. To alleviate this computational cost, we can use DHL [8], which is a heuristic algorithm and was proposed originally for graph databases. DHL assumes that edges are sorted by weights, and it greedily accesses the edges in this order, so as to avoid enumerating triangles with large weights. However, to employ DHL, we face substantial time incurred by building a spatial neighbor graph of P and sorting a large amount of edges.

To solve the above issues, we propose an efficient algorithm that returns the exact answer. We find an observation that a subset of the spatial neighbor graph, which usually contains the top-k weighted triangles, can be built offline. Besides, 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, resulting in improvement of online computation. Thanks to these observations, our algorithm does not need to correctly build the spatial graph and sort all edges. To summarize, our main contributions are as follows:

- We address the problem of retrieving the top-k weighted spatial triangles. To our knowledge, we are the first to tackle this problem in spatial databases.
- We propose a simple, efficient, and exact solution for this problem.
- We conduct experiments on real datasets, and the results show that our solution for static data is up to *three orders of magnitude faster* than a baseline algorithm.

2 Preliminary

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 $\in \mathbb{R}^2$. We use $dist(p, p')$ to denote the Euclidean distance between p and p'. We assume that P is memory resident.

Given a distance threshold r, where r is a tolerable distance between points to regard them as being located close to each other, 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:*

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

Then, our problem in Section 3 is 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*[1].

3 Our Solution

Main idea. To efficiently retrieve k triangles with the minimum weight, it is desirable to prune points that do not contribute to the top-k result. 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.

This idea brings an important advantage: the spatial neighbor graph of P needs to be built online (since it depends on r), whereas the t-NN graph of P can be built *offline* (for $t = O(1)$). Furthermore, if we have the t-NN graph of P, for each $p \in P$, we can enumerate a promising triangle having p, i.e., the triangle formed by p and its 2 nearest neighbors, in the same offline step. Even if these triangles are not included in the top-k result, they usually have small weights, yielding a tight threshold for online computation in practice. This threshold helps prune unnecessary points (and thus triangles), so the above ideas improve the efficiency of online computation.

Our algorithm is designed based on the above ideas and consists of a one-time offline computation and online computation. In the next subsections, we present how to prepare these triangles and how to compute the exact top-k result in detail.

[1] When r is too small, the spatial neighbor graph of P can be very sparse and there may be less than k triangles in the graph. In this case, this problem is easily solved, thus we assume that r is reasonably specified and there are many triangles in the graph.

Offline processing. 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. 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 [5]. 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. The kd-tree of P is built in $O(n \log n)$ time. For a fixed B (i.e., $B = O(1)$), the B-NNs of $p_x \in P$ are retrieved in $O(Bn^{1-1/d}) = O(\sqrt{n})$ time [12]. We can therefore build the B-NN graph in $O(n^{1.5})$ time. Last, sorting triangles in T needs $O(n \log n)$ time. Our offline algorithm hence needs $O(n^{1.5})$ time.

Building the spatial neighbor graph of P incurs $O(n(\sqrt{n} + s_{avg}))$ time, where s_{avg} is the average number of edges held by each point. Compared with this, our offline algorithm is cheaper, and it is general to any k and r. We exploit the B-NN graph of P and the set T of triangles to efficiently retrieve the top-k weighted spatial triangles.

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

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. Consider a triangle $\triangle_{x,x_{i+2},y}$. We have $w(e_{x,x_{i+2}}) \leq w(e_{x,y}) + w(e_{x_i,y})$ from triangle inequality. Equation (2) shows that θ is the sum of the weights of two edges of the (intermediate) top k-th triangle. Therefore, if $w(e_{x,x_{i+2}}) > \theta$, the weights of any triangles that have $e_{x,x_{i+2}}$ are larger than τ. □

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 p_x can be safely removed from P_{cand}. REDUCE-CANDIDATES(P_{cand}, i, θ) does this point removal. (For a point $p_x \in P_{cand}$, if we do not have $e_{x,x_{i+2}}$, we update the B-NN graph by increasing B before REDUCE-CANDIDATES(P_{cand}, i, θ).)

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.

• *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 $j \in [1, ..., i+1]$, while updating the top-k result R, τ, and θ.

W.r.t. p_{x_j}, we access it in order of $p_{x_1}, ..., p_{x_{i+1}}$. 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. We analyze the theoretical performance of our online algorithm. Step 1 needs $O(1)$ time, since T is sorted offline. Consider the i-th iteration of step 2, and let n_i be the size of P_{cand} in this iteration. (Notice that n_i is affected by k.) In step 2, for each $p_x \in P_{cand}$, we check $e_{x,x_{i+2}}$. It hence needs $O(n_i)$ time[2]. Next, consider the i-th iteration of step 3. Let n'_i be the size of P_{cand} in this iteration. Notice that $n'_i \leq n_i$, since step 2 reduces the size of P_{cand}. In step 3, for each $p_x \in P_{cand}$, we enumerate triangles formed by p_x, $p_{x_{i+2}}$, and p_{x_j}. Although we can early terminate this enumeration, its worst number is $i+1$. That is, we need $O(i)$ time for p_x, thus step 3 requires $O(i \cdot n'_i)$ time. Consequently, 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.

[2] When we need to update the B-NN graph, we need $O(n_i\sqrt{n})$ additional time.

4 Experiment

This section introduces our experimental results. All experiments were conducted on a machine with 3.6GHz Intel Core i9-9900K CPU and 128GB RAM. In addition, all algorithms tested were single threaded and compiled by g++ 9.3.0 with -O3 flag.

We compared it with DHL [8], which can compute the exact answer from the spatial neighbor graph of P. For DHL, we used the original implementation[3].

Dataset. We used two real datasets, CaStreet[4] and Places[5]. 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.

Parameter. We set $n = 1,000,000$ (via random sampling) and $k = 100$ by default. In all experiments, $r = 0.01$ (and it did not affect the performance of our algorithm). We set $B = 10$.

Impact of k. Next, we investigate the impact of the result size k, and Fig. 1 shows our experimental result. Our algorithm is significantly faster than DHL. For example, when $k = 100$, our algorithm is 2021 and 1465 times faster than DHL on CaStreet and Places, respectively. DHL suffers from the overhead cost incurred by dealing with the spatial neighbor graph (while we do not have this drawback.)

It can be also observed that the tendency of our algorithm is different between CaStreet and Places. We found that Places is denser than CaStreet. Due to this feature, compared with CaStreet case, our algorithm needed to enumerate more triangles and update the top-k result more frequently on Places. However, it still returns the result in 1.13 [sec] even when $k = 1000$. Recall that it needed 0.16 [sec] when $k = 100$. We therefore see that our algorithm scales linearly to k.

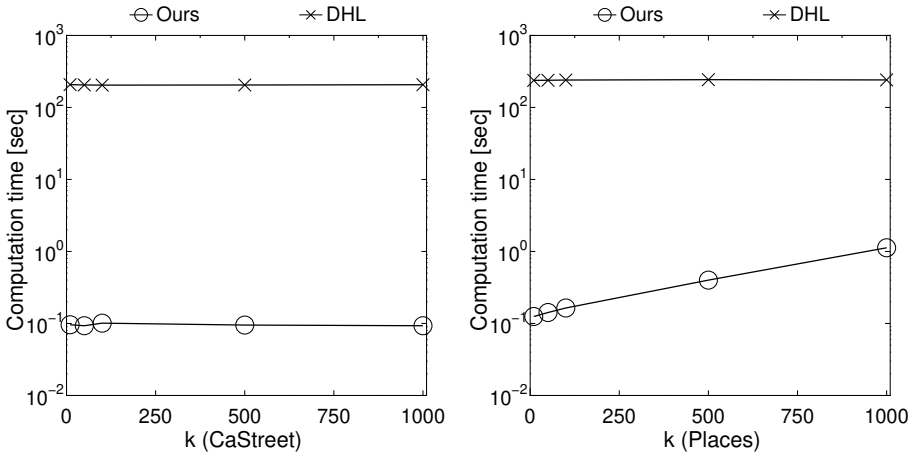
Impact of n. Fig. 2 studies the scalability of our algorithm to the cardinality of dataset n. Our algorithm has a linear scalability to n, while DHL is superlinear w.r.t. n. This clarifies the advantage of our algorithm. When we used all points of CaStreet and Places, our algorithm is 2807 and 6193 times faster than DHL on CaStreet and Places, respectively.

5 Conclusion

The number of location-based services is increasing, and a lot of spatial points are being generated nowadays. This fact strengths the importance of analyzing spatial points, and much efforts have been made to devise techniques for spatial point analysis.

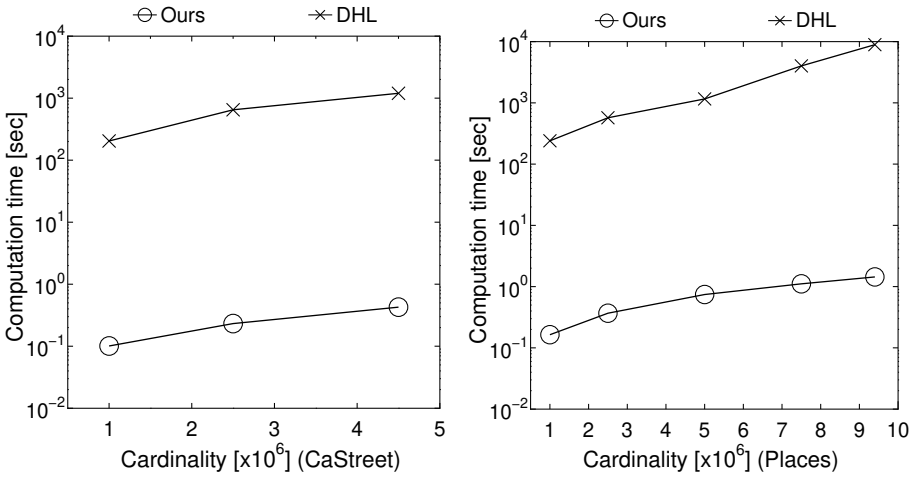
As a spatial point analysis tool, we proposed the problem of retrieving the top-k weighted spatial triangles. Because the number of triangles in a set of

[3] https://github.com/raunakkmr/Retrieving-top-weighted-triangles-in-graphs
[4] http://chorochronos.datastories.org/?q=node/59
[5] https://archive.org/details/2011-08-SimpleGeo-CC0-Public-Spaces



(a) CaStreet (b) Places

Fig. 1. Impact of k



(a) CaStreet (b) Places

Fig. 2. Impact of cardinality of dataset

spatial points can be huge, simply enumerating triangles is time-consuming. To avoid this issue, we proposed an efficient algorithm that returns the exact answer. We conducted experiments on real datasets, and the results demonstrate the efficiencies of our solution.

Acknowledgements

This research is partially supported by JSPS Grant-in-Aid for Scientific Research (A) Grant Number 18H04095, JST CREST Grant Number J181401085, and JST PRESTO Grant Number JPMJCR21F2.

References

1. Amagata, D., Hara, T.: Monitoring maxrs in spatial data streams. In: EDBT. pp. 317–328 (2016)
2. Amagata, D., Hara, T.: A general framework for maxrs and maxcrs monitoring in spatial data streams. ACM Transactions on Spatial Algorithms and Systems (TSAS) **3**(1), 1–34 (2017)
3. Amagata, D., Hara, T.: Identifying the most interactive object in spatial databases. In: ICDE. pp. 1286–1297 (2019)
4. Amagata, D., Tsuruoka, S., Arai, Y., Hara, T.: Feat-sksj: Fast and exact algorithm for top-k spatial-keyword similarity join. In: SIGSPATIAL. pp. 15–24 (2021)
5. Bentley, J.L.: Multidimensional binary search trees used for associative searching. Communications of the ACM **18**(9), 509–517 (1975)
6. Fang, Y., Li, Y., Cheng, R., Mamoulis, N., Cong, G.: Evaluating pattern matching queries for spatial databases. The VLDB Journal **28**(5), 649–673 (2019)
7. Fang, Y., Wang, Z., Cheng, R., Li, X., Luo, S., Hu, J., Chen, X.: On spatial-aware community search. IEEE Transactions on Knowledge and Data Engineering **31**(4), 783–798 (2018)
8. Kumar, R., Liu, P., Charikar, M., Benson, A.R.: Retrieving top weighted triangles in graphs. In: WSDM. pp. 295–303 (2020)
9. Nishio, S., Amagata, D., Hara, T.: Lamps: Location-aware moving top-k pub/sub. IEEE Transactions on Knowledge and Data Engineering **34**(1), 352–364 (2022)
10. Pandey, V., Kipf, A., Neumann, T., Kemper, A.: How good are modern spatial analytics systems? PVLDB **11**(11), 1661–1673 (2018)
11. Park, H.M., Myaeng, S.H., Kang, U.: Pte: Enumerating trillion triangles on distributed systems. In: KDD. pp. 1115–1124 (2016)
12. Toth, C.D., O'Rourke, J., Goodman, J.E.: Handbook of discrete and computational geometry. CRC press (2017)
13. Tsuruoka, S., Amagata, D., Nishio, S., Hara, T.: Distributed spatial-keyword knn monitoring for location-aware pub/sub. In: SIGSPATIAL. pp. 111–114 (2020)
14. Wang, Y., Yu, S., Dhulipala, L., Gu, Y., Shun, J.: Geograph: A framework for graph processing on geometric data. ACM SIGOPS Operating Systems Review **55**(1), 38–46 (2021)
15. Yu, J., Sarwat, M.: Geosparkviz: a cluster computing system for visualizing massive-scale geospatial data. The VLDB Journal **30**(2), 237–258 (2021)
16. Zhang, C., Zhang, Y., Zhang, W., Qin, L., Yang, J.: Efficient maximal spatial clique enumeration. In: ICDE. pp. 878–889 (2019)