Osaka University Knowledge Archive

Title	Retrieving Top-N Weighted Spatial k-cliques
Author(s)	Taniguchi, Ryosuke; Amagata, Daichi; Hara, Takahiro
Citation	Proceedings - 2022 IEEE International Conference on Big Data, Big Data 2022. 2022, p. 4952-4961
Version Type	AM
URL	https://hdl.handle.net/11094/92857
rights	© 2022 IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes, creating new collective works, for resale or redistribution to servers or lists, or reuse of any copyrighted component of this work in other works.
Note	

Osaka University Knowledge Archive : OUKA

https://ir.library.osaka-u.ac.jp/

Osaka University

Retrieving Top-N Weighted Spatial k-cliques

Ryosuke Taniguchi
Osaka University
Osaka, Japan
taniguchi.ryosuke@ist.osaka-u.ac.jp

Daichi Amagata
Osaka University
Osaka, Japan
amagata.daichi@ist.osaka-u.ac.jp

Takahiro Hara
Osaka University
Osaka, Japan
hara@ist.osaka-u.ac.jp

***Abstract*—Spatial data analysis is a classic yet important topic because of its wide range of applications. Recently, as a spatial data analysis approach, a neighbor graph of a set P of spatial points has often been employed. This paper also considers a spatial neighbor graph and addresses a new problem, namely top-N weighted spatial k-clique retrieval. This problem searches for the N minimum weighted cliques consisting of k points in P, and it has important applications, such as community detection and co-location pattern mining. Recent spatial datasets have many points, and efficiently dealing with such big datasets is one of the main requirements of applications. A straightforward approach to solving our problem is to try to enumerate all k-cliques, which incurs $O(n^k k^2)$ time. Since $k \geq 3$, this approach cannot achieve the main requirement, so computing the result without enumerating unnecessary k-cliques is required. This paper achieves this challenging task and proposes a simple practically-efficient algorithm that returns the exact answer. We conduct experiments using two real spatial datasets consisting of million points, and the results show the efficiency of our algorithm, e.g., it can return the exact top-N result within 1 second when $N \leq 1000$ and $k \leq 7$.**

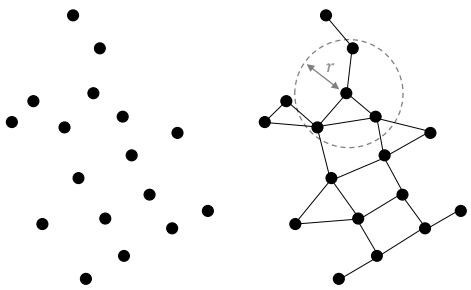
***Index Terms*—spatial data, top-N retrieval, k-clique, algorithm**

I. Introduction

Nowadays, IoT devices, such as smartphones, tablets, and wearable watches, are everywhere. Through these devise, we usually receive location-based services [1]–[5], and, at the same time, many geo-spatial data are generated (e.g., via GPS) [6]–[8]. For example, even 10 years ago, Google was generating approximately 25 PB of data per day, and most of them were spatial data [9]. Another example is NASA, which generates approximately 4 TB of spatial data per day [10]. It is well known that useful observations and knowledge are hidden in such big spatial data. Dealing with big spatial data is, however, not a trivial task, so many works developed efficient algorithms for analysing such data [11]–[18]. Recently, as one of spatial data analysis approaches, graph-based mining has receiving attention [19]–[25]. This paper also considers graph-based spatial data analysis.

A. Motivation

Let P be a set of geo-spatial (i.e., 2-dimensional) points. We use $dist(p, p')$ to denote the distance between two points $p, p' \in P$. Furthermore, let r be a distance threshold. A spatial neighbor graph G_r consists of P (i.e., a point in P is regarded as a node) and E, where an edge exists between two nodes p and p' iff $dist(p, p') \leq r$. Fig. 1 illustrates a concrete example. The intuition of using the spatial neighbor graph is to focus only on near points, because there is some relationship between points existing near each other [26]. Also, by considering P as a graph structure, we can find some interesting inner structures (i.e., sub-graphs). These structures are useful for important applications, e.g., community detection [27], motif detection [28], and co-location pattern mining [29]. As an inner structure, some works considered spatial cliques [24], [30], [31], because the clique is a fundamental structure for the above applications. For example, [20] demonstrated that non-trivial co-location patterns are observable by retrieving triangles (i.e., cliques consisting of three nodes).



(a) Point set P (b) Neighbor graph G_r

Fig. 1. A neighbor graph G_r of P (r is a given distance threshold)

As shown above, finding spatial cliques has important applications. Simply enumerating all cliques is, however, not helpful for these applications, for two reasons. First, there is usually a significant number of cliques in a spatial neighbor graph G when G has many points (nodes). Such a significant output size overwhelms users and makes interesting knowledge hard to observe. Second, when analyzing an output, the size of cliques should be fixed; otherwise, the output may contain cliques consisting of many points. Such large cliques are not easy to analyze, as there are too many point combinations. These two reasons indicate that *both the output size and clique size should be controllable*. In other words, it is desirable that users can specify the output size N and clique size k. Motivated by this, we consider the problem of retrieving the top-N weighted spatial k-cliques, which finds the N minimum weighted cliques consisting of k points. (The formal definition appears in Section II.) To the best of our knowledge, this is the first work to address this problem.

978-1-6654-8045-1/22/$31.00 © 2022 IEEE

B. Challenge

One of the main requirements of spatial data analysis is efficiency, because it has to deal with many spatial points. Considering our problem, this is not easy to achieve. A straightforward algorithm that solves our problem is to consider all possible combinations of points that form k-cliques. This algorithm requires $O(n^k k^2)$ time. Because $k \geq 3$, this algorithm is not feasible in practice, even for moderate-sized datasets. Therefore, we need to focus only on points (or k-cliques) that can be in the top-N result by ignoring unnecessary points.

To achieve this, we consider a threshold-based strategy. Given a threshold τ, which is the weight of (intermediate) top N-th k-clique, we can filter points forming only k-cliques with larger weights than τ, thus can ignore unnecessary points. However, to make this strategy function well, we have to overcome the following two challenges: (i) τ needs to be tight, and (ii) τ needs to be computed efficiently. The first challenge is a trivial requirement. If τ is loose, the filtering efficiency decreases, and we still have to enumerate many unnecessary k-cliques. The second challenge implies that computing a tight τ with long running time does not make sense.

C. Contribution

We overcome the above challenges and propose a practically-efficient algorithm that returns the exact answer. We show that, by using an i-nearest neighbor graph, which is built offline, a tight threshold is efficiently obtained. Our algorithm exploits this threshold τ and employs an iterative filter-and-verify approach. In this approach, we filter unnecessary points by using τ and focus on only points that are not filtered. Then, we verify whether these points can form k-cliques with smaller weights than τ. We repeat this operation while updating the top-N result and τ until we have no more candidate points. In addition to this, we propose some techniques that avoid enumerating unnecessary k-cliques in the verification step to further improve the efficiency. Our empirical results show that our algorithm functions well on million-scale datasets.

To summarize, this paper makes the following contributions.

- We address the problem of retrieving the top-N weighted spatial k-cliques. This is the first work to tackle this problem.
- We propose a simple, efficient, and exact algorithm for this problem.
- We conduct extensive experiments using two real spatial datasets. The results demonstrate the efficiency of our algorithm. For example, when $N \leq 1000$ and $k \leq 7$, our algorithm returns the answer within 1 second.

Organization. The rest of this paper is organized as follows. Section II introduces preliminary information for presenting our algorithm. We propose our algorithm in Section III, and then report our experimental results in Section IV. We review related studies in Section V. Finally, in Section VI, we conclude this paper.

II. Preliminary

In this section, we define some notations and the problem we address. Table I summarizes the notations frequently used in this paper. Let P be a set of n geo-spatial points, and we assume that P is static and memory-resident. Each point $p \in P$ is represented as a 2-dimensional coordinate $\langle x, y \rangle$ in Euclidean space $\mathbb{R}^2$. Given two points $p, p' \in P$, the Euclidean distance between p and p' is shown by $dist(p, p')$.

Before defining our problem, we consider a spatial neighbor graph of P.

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 $G_r = (P, E_r)$. Each node of G_r is a point in P. In addition, E_r is a set of undirected weighed edges, and there is an edge $e(p, p')$ between two nodes (i.e., points) p and p' iff $dist(p, p') \leq r$. The weight of edge $e(p, p')$, denoted by $w(e(p, p'))$, is $dist(p, p')$.*

Next, we define k-clique.

Definition 2 (k-clique). *Given a spatial neighbor graph G_r of P, a k-clique C is a set of k points in P such that there is an edge $\in E_r$ between any two points in C.*

In this paper, we design the weight of a k-clique as follows.

Definition 3 (Weight of k-clique). *Given k-clique C, define $E(C)$ as $E(C) = \{e(p, p') \mid p, p' \in C\}$. The weight of C, denoted by $w(C)$, is defined as*

$$w(C) = \max_{E(C)} w(e(p, p')). \tag{1}$$

It is important to notice that a k-clique C having small $w(C)$ consists of points existing near each other, because a small $w(C)$ means that the farthest pair of two points in C is still close. That is, $w(C)$ effectively represents the cohesiveness of C, and it is an important measure for evaluating a set of spatial points [22], [24].

Based on this concept[1], we rank k-cliques in G_r and define our problem.

Definition 4 (Top-N weighted spatial k-clique retrieval problem). *Given a set P of points, a distance threshold r, an output size N, and a clique size k, this problem retrieves N k-cliques with the smallest weight in G_r.*

The objective of this paper is to solve the problem in Definition 4 exactly in a practically-efficient manner. In Section III, we propose an in-memory algorithm that achieves this objective.

III. Our Algorithm

A straightforward algorithm for solving the problem in Definition 4 builds a spatial neighbor graph G_r of P, enumerates all k-cliques in G_r, and then computes N k-cliques with the minimum weight among the enumerated ones. This is not feasible in practice, since the number of k-cliques in G_r

[1]The other definitions of the weight of k-clique are out of the scope of this paper and can be addressed in a future work.

TABLE I
OVERVIEW OF NOTATIONS

Notation	Description
P	set of n spatial points
p	spatial (2-dimensional) point
$e(p, p')$	edge between p and p'
$w(e(p, p'))$	weight of $e(p, p')$
C	k-clique
$w(C)$	weight of k-clique
k	clique size
N	output size
G_r	spatial neighbor graph
$\mathcal{G}_i$	i-nearest neighbor graph
p^k	k-nearest neighbor point of p in $P \backslash \{p\}$
τ	threshold: weight of (intermediate) top N-th k-clique

is $O(n^k)$ and $k \geq 3$. Trivially, we have $N \ll O(n^k)$, so enumerating all k-cliques in G_r is too redundant.

A. Main idea

Equation (1) suggests that the points in k-cliques have to be located near each other so as to be included in the top-N result. This observation derives an intuition that, for each point p in a k-clique C with a small $w(C)$, $p' \in C$ $(p \neq p')$ is in the k-nearest neighbors (k-NNs) of p among the points in P. Then, we see that an i-NN graph, which is defined below, is useful to retrieve k-cliques with small weights.

DEFINITION 5 (i-NEAREST NEIGHBOR GRAPH). *Given a set P of points and an integer $i \geq 1$, the i-NN graph of P, $\mathcal{G}_i$, consists of all points in P and $i \cdot n$ edges. Specifically, given a point $p \in P$, for each $j \in [1, i]$, there is an edge between p and p^j, where p^j is the j-th nearest neighbor of p in $P \backslash \{p\}$, and $w(e(p, p^j))$ is $dist(p, p^j)$.*

It is important to notice that $\mathcal{G}_i$ needs only $O(n)$ space and is independent of r, so it can fit into main-memory and can be built offline. By setting a sufficiently large value as i ($i = 30$ in our experiments) such that $i > k$ for reasonable values of N and k in applications, we can find k-cliques with small weights, *without building* G_r. Furthermore, in practice, a point $p \in P$ and its $(k-1)$-NN points form a k-clique. Thanks to $\mathcal{G}_i$, such k-cliques are easy to enumerate, and they have small weights, yielding a tight threshold τ with a small computational cost. This tight threshold contributes to filtering many unnecessary points and k-cliques.

B. Overview

Based on the above main idea, we devise an efficient algorithm for our problem. Our algorithm has offline and online processing.

- *Offline processing.* We build $\mathcal{G}_i$ by using a tree-structure, e.g., a kd-tree [32], offline. This processing is done only once in general.

- *Online processing.* Given N, k, and r, our algorithm runs its online processing, which is described in Algorithm 1. Our online algorithm has the following steps:

Algorithm 1: PROPOSED-ALGORITHM

Input: $\mathcal{G}_i$ (i-NN graph of P), r (distance threshold), N (output size), and k (clique size)

1 $\tau \leftarrow r$ ▷ initialize threshold τ
2 $R \leftarrow \varnothing$ ▷ initialize the result set R
3 $P_{cand} \leftarrow P$ ▷ initialize a candidate set P_{cand}
4 $\mathcal{G}_{i,r} \leftarrow \mathcal{G}_i$
5 Remove all edges $e(p, p')$ in $\mathcal{G}_{i,r}$ s.t. $w(e(p, p')) > r$
6 $f \leftarrow$ Get-Threshold ▷ use Algorithm 2, 3, or 4
7 $j \leftarrow 0$
8 **if** $(f = 0) \wedge$ (*Algorithm 3 or 4 is used*) **then**
9 $\quad j \leftarrow -1$
10 **while** $P_{cand} \neq \varnothing$ **do**
11 $\quad$ Filter(k, j, τ, P_{cand})
12 $\quad$ Verify$(\mathcal{G}_{i,r}, k, j, \tau, P_{cand}, N, R)$
13 $\quad j \leftarrow j + 1$
14 $\quad$ **if** $k + j = i + 1$ **then**
15 $\quad\quad$ **return** $\varnothing$ ▷ abort case
16 **return** R

1) It first initializes the threshold τ, the result set R, and the candidate point set P_{cand} (lines 1–3).
2) Next, it makes a copy of $\mathcal{G}_i$, denoted by $\mathcal{G}_{i,r}$, and removes all edges $e(p, p')$ in $\mathcal{G}_{i,r}$ such that $w(e(p, p')) > r$ (lines 4–5).
3) Then, it computes intermediate τ and R through Get-Threshold (line 6).
4) After that, it filters unnecessary points through Filter and enumerates additional k-cliques while updating τ and R in Verify. At the first iteration of this step, we have $j = 0$ and Verify considers k-cliques having p and its $(k + j)$-NN for each $p \in P_{cand}$. (In some cases, j is initialized as -1, not to miss k-cliques that have not been considered so far, see line 8.) At the end of the iteration, j is incremented by one. This step is iterated until $P_{cand} = \varnothing$[2].

Notice that $\mathcal{G}_{i,r} \subseteq G_r$, so it is clear that the top-N result in $\mathcal{G}_{i,r}$ is the one in G_r.

The efficiency of Algorithm 1 depends on (i) that of Get-Threshold and (ii) the tightness of τ obtained in Get-Threshold, because a tight τ can filter many points in Filter and enumerate less k-cliques in Verify. In the following subsections, we elaborate these functions.

C. Threshold Computation

To get a tight threshold τ, considering a k-clique consisting of a point p and its $(k-1)$-NN points is promising. The number of such k-cliques is at most $n = |P|$, and we usually have $N \ll n$. Hence, enumerating all of them is redundant.

[2]If we find that the top-N result cannot be obtained in $\mathcal{G}_{i,r}$ during the iterations, we abort the retrieval. In this case, we need to build an i'-NN graph, where $i' > i$. (In our experimental setting, this abort case did not occur.)

Algorithm 2: Get-Threshold-with-Sorting

Input: $\mathcal{G}_{i,r}$, N, k, τ, P_c, and R

1 $E \leftarrow$ a set of all edges $e(p, p^{k-1})$ where $p \in P_{cand}$
2 Sort every $e(p, p^{k-1}) \in E$ in ascending order of $w(e(p, p^{k-1}))$
3 $\mathcal{C} \leftarrow \varnothing$
4 $f \leftarrow 0$
5 **for** *each* $e(p, p^{k-1}) \in E$ **do**
6 **if** $\tau \leq w(e(p, p^{k-1}))$ **then**
7 $f \leftarrow 1$
8 **break**
9 $C \leftarrow k$-clique consisting of p and its $(k-1)$-NNs
10 **if** $(w(C) < \tau) \wedge (C \notin \mathcal{C})$ **then**
11 $\mathcal{C} \leftarrow \mathcal{C} \cup \{C\}$
12 Update R and τ
13 **return** f

To alleviate this redundancy, we propose three heuristic approaches for Get-Threshold. (Our empirical studies compare the performances of these approaches.) Note that all the heuristic approaches need only $O(n)$ space.

• *Sorting-based approach.* We first propose a sorting-based heuristic, which is shown in Algorithm 2. For each $p \in P$, we focus on the edge between p and its $(k-1)$-th NN, i.e., $e(p, p^{k-1})$. If $w(e(p, p^{k-1}))$ is small, the k-clique consisting of p and its $(k-1)$-NN points would have a small weight. On the other hand, if $w(e(p, p^{k-1})) \geq \tau$, p is not necessary.

PROPOSITION 1. *If $w(e(p, p^{k-1})) \geq \tau$, any k-cliques C having p have $w(C) \geq \tau$.*

PROOF. From Definitions 2 and 3, we have $w(C) \geq w(e(p, p^{k-1}))$. Then, this proposition immediately holds. □

Based on the above observations, we maintain such edges in E in ascending order of weight. Then, we access each edge in E in this order. Assume that we now access $e(p, p^{k-1}) \in E$. If $\tau \leq w(e(p, p^{k-1}))$, p cannot form any k-cliques that can be in the top-N result from Proposition 1. As E is regarded as an ordered array, the subsequent points fall into the same case, so Algorithm 2 terminates. Otherwise, we consider p and its $(k-1)$-NN points. If these points form a k-clique C and this clique has not been considered so far, we use C to update R and τ.

Algorithm 2 assumes that at least N distinct k-cliques can be enumerated by using some points and their $(k-1)$-NN points. Our experiments confirm that this assumption holds in practice. Even if this assumption does not hold, extending this approach so that we can enumerate N distinct k-cliques is still straightforward. Specifically, we can replace Algorithm 2 with Algorithm 6, which considers $(k+j)$-NN points ($j \geq 0$). Its detail is shown in Section III-E. (This discussion holds for the other two heuristic approaches.)

• *Sampling-based approach.* The second approach is a simple variant of Algorithm 2. Notice that Algorithm 2 needs to sort n edges and needs to consider at most n k-cliques. We alleviate this cost via random sampling.

Algorithm 3: Get-Threshold-with-Sampling

Input: $\mathcal{G}_{i,r}$, N, k, τ, P_{cand}, R, and s (sample size)

1 $P_{sample} \leftarrow$ a set of s randomly sampled points in P_{cand}
2 Get-Threshold-with-Sorting w/o line 7

Algorithm 4: Get-Threshold-with-Priority-Queue

Input: $\mathcal{G}_{i,r}$, N, k, τ, P_c, R, and q (queue size)

1 $E \leftarrow$ a set of q distinct edges $e(p, p^{k-1})$ with the smallest $w(e(p, p^{k-1}))$, where $p \in P_{cand}$
 ▷ `implemented by a priority queue`
2 Run lines 2–12 of Algorithm 2

Algorithm 3 describes the second approach. It randomly samples s points from $P = P_{cand}$ and then runs Get-Threshold-with-Sorting (i.e., Algorithm 2) on a set of the s samples.

• *Priority queue-based approach.* Although the second approach can reduce the cost of computing τ, it may yield a looser threshold than the one obtained by the first approach. To mitigate this drawback (with a small sacrifice in computational cost), we use only $q \ll n$ edges (or points) with the smallest weight, in the expectation that these points form k-cliques with small weights.

Algorithm 4 shows the last approach, which is also a variant of Get-Threshold-with-Sorting. Consider the edge set E computed in Algorithm 2, and we have $|E| = n$. In Algorithm 4, we use only q edges with the smallest weight among E. This can be done efficiently by using a priority queue (i.e., q is the queue size). Then, we run the same operations as in Get-Threshold-with-Sorting on the q edges (or points).

D. Candidate Point Removal

After getting (or updating) τ, we remove unnecessary points in Filter. We note that Filter confirms whether $p \in P_{cand}$ can form k-cliques having p and its $(k+j)$-th nearest neighbor, where $j \geq -1$. (Recall that we run Filter and Verify in an iterative manner by using the variable j, see Algorithm 1.) If p cannot do it, p is not necessary any more, which is trivially derived from Proposition 1. Based on this rationale, Algorithm 5 removes points, which are unnecessary for retrieving the correct top-N result, from P_{cand}.

E. Additional k-clique Enumeration

When $P_{cand} \neq \varnothing$, the points in P_{cand} may form a k-clique C such that $w(C) < \tau$. To find such k-cliques, given $p \in P_{cand}$ and j (see Algorithm 1), we enumerate k-cliques having p and p^{k+j} for each $p \in P_{cand}$. Because, for p, we consider k-cliques consisting of p and some of its $(k+j)$-nearest neighbor points, the number of k-cliques having p and p^{k+j} is $\binom{k+j-1}{k-2}$.

Algorithm 5: Filter

Input: k, j, τ, and P_{cand}

for *each* $p \in P_{cand}$ *s.t.* $w(e(p, p^{k+j})) \geq \tau$ **do**

  $P_{cand} \leftarrow P_{cand} \backslash \{p\}$

Enumerating this number of k-cliques incurs a non-negligible cost, particularly when j increases. Therefore, we propose two techniques that can avoid brute-force enumeration.

Pruning $p \in P_{cand}$. Given a point $p \in P_{cand}$ and j, to avoid enumerating unnecessary k-cliques having p and p^{k+j}, we consider the following point set:

$$P_{k+j} = \{p^l \mid p^l \in P, l \in [1, k+j-1], dist(p^l, p^{k+j} < \tau\}.$$

Given p and p^{k+j}, to form a k-clique C having these two points, we select $(k-2)$ points from the set of $(k+j-1)$-nearest neighbor points of p. In addition, to have $w(C) < \tau$, it is necessary to have $dist(p^l, p^{k+j}) < \tau$ where $l \in [1, k+j-1]$. Note that, for p, P_{k+j} is a set of points that satisfy these constraints. Now focus on the size of P_{k+j}. If $|P_{k+j}| < k-2$, we have $|P_{k+j} \cup \{p, p^{k+j}\}| < k$, meaning that there are no k-cliques C having p and p^{k+j} such that $w(C) < \tau$. Therefore, if p has this case, we can skip enumerating $\binom{k+j-1}{k-2}$ k-cliques without losing correctness. Because computing P_{k+j} needs only $O(k+j)$ time and $O(k+j) \ll \binom{k+j-1}{k-2}$, this pruning is efficient.

Early termination of k-clique enumeration. Even when we cannot prune p in the above technique, we still can skip enumerating $\binom{k+j-1}{k-2}$ k-cliques *by using an updated* τ. The rationale is the same as that of Algorithm 2. Consider that every $p \in P_{cand}$ is sorted in ascending order of $w(e(p, p^{k+j}))$. By using the sorted access approach, we (i) enumerate additional k-cliques to update R and τ, and (ii) ignore all p such that $w(e(p, p^{k+j})) \geq \tau$.

Algorithm 6 uses the above techniques to enumerate additional k-cliques while skipping redundant enumerations. Its overall approach is the same as that of Algorithm 2 (so Algorithm 6 also needs only $O(n)$ space). The main difference is that it employs the pruning technique before enumerating k-cliques, see lines 6–7.

F. Discussion

Our algorithm uses data structures with $O(n)$ space (the i-NN graph and the data structure for Get-Threshold), so the space complexity of our algorithm is $O(n)$. As our filtering techniques are heuristic, its worst time complexity is still $O(n^k)$. However, different from this theoretical result, our empirical studies show that our algorithm scales well to both n and k.

Last, recall that, for each $p \in P_{cand}$, our algorithm considers k-cliques having p, p^{k+j}, and $(k-2)$ points in its $(k+j-1)$-nearest neighbors. Since (1) j is initialized as -1, (ii) j is incremented by one, and (iii) p is filtered iff $w(p, p^{k+j}) \geq \tau$, our algorithm does not miss any k-cliques that are in the top-N result. This clarifies the correctness of our algorithm.

Algorithm 6: Verify

Input: $\mathcal{G}_{i,r}$, N, k, j, τ, P_{cand}, and R

1 $E \leftarrow$ a set of all edges $e(p, p^{k+j})$ where $p \in P_{cand}$

2 Sort every $e(p, p^{k+j}) \in E$ in ascending order of $w(e(p, p^{k+j}))$

3 **for** *each* $e(p, p^{k+j}) \in E$ **do**

4 **if** $\tau \leq w(e(p, p^{k+j}))$ **then**

5 **break**

6 $P_{k+j} \leftarrow \{p^l \mid p^l \in P, l \in [1, k+j-1], dist(p^l, p^{k+j}) < \tau\}$

7 **if** $|P_{k+j}| \geq k-2$ **then**

8 $\mathcal{C} \leftarrow$ a set of all k-cliques having p, p^{k+j}, and $(k-2)$ points in P_{k+j}

9 **for** *each* $C \in \mathcal{C}$ *s.t.* $(w(C) < \tau) \wedge (C \notin R)$ **do**

10 Update R and τ

IV. Experiment

In this section, we report our experimental results. All experiments were conducted on a Ubuntu 20.04 LTS machine equipped with Intel Core i9-9900K CPU@3.6GHz and 128GB RAM.

A. Setting

Algorithms. We evaluated the following algorithms.

- SA: Algorithm 1 employing Get-Threshold-with-Sorting in line 6.
- SSA: Algorithm 1 employing Get-Threshold-with-Sampling in line 6. We used two samples sizes, i.e., $s = 0.01 \cdot |P|$ and $s = 0.1 \cdot |P|$.
- PQA: Algorithm 1 employing Get-Threshold-with-Priority-Queue in line 6. We set $q = 5N$.
- Triangle: the algorithm proposed in [19]. This algorithm can deal with only $k = 3$, as it is specific to triangles. We therefore used this algorithm as a baseline in the case where $k = 3$.
- Brute-force: an algorithm that builds the spatial neighbor graph G_r, enumerates all k-cliques with a state-of-the-art enumeration algorithm DDegCol [33], and computes the top-N k-cliques among them. For the k-clique enumeration, we used the original implementation of [33].

The above algorithms were implemented in C++, compiled by g++ 9.3.0 with -O3 optimization, and single-threaded.

Dataset. We used two real spatial datasets, CaStreet[3] and Places[4]. CaStreet consists of the minimum bounding rectangles of road segments in the U.S.A. We used bottom-left and upper-right points, and obtained 4,499,454 points. Then, we removed the duplicated points. Places consists of 9,356,750

[3] http://chorochronos.datastories.org/?q=node/59

[4] https://archive.org/details/2011-08-SimpleGeo-CC0-Public-Spaces

TABLE II
RUNNING TIME COMPARISON IN SECONDS AT THE DEFAULT PARAMETER SETTING

Algorithm	CaStreet	Places
SA	0.167	0.226
SSA ($s = 0.01\|P\|$)	5.164	0.767
SSA ($s = 0.1\|P\|$)	0.164	0.228
PQA	0.068	0.153
Brute-force	> 10000	> 10000

coordinates of public places in the U.S.A. As with CaStreet, we removed the duplicated points in Places.

Default parameter. We set $r = 0.01$ and $i = 30$, and they were fixed. This is because $r = 0.01$ yields a sufficiently large spatial neighbor graph (more than 50 million edges both on CaStreet and Places), and $i = 30$ was also able to consider large (resp. reasonable) values of N (resp. k).

The default values of N and k were 1,000 and 5, respectively. For each dataset, we ran random sampling to obtain 1 random million points, and we used them as the default dataset. When we investigated the impact of a given parameter, we fixed the other parameters.

B. Result

The evaluated algorithms except for Brute-force share the same offline processing. (Brute-force has no pre-processing.) This section hence focuses on the online running time.

Comparison and ablation studies. We first compare the performances of the evaluated algorithms. TABLE II shows the running time result. (Triangle cannot deal with $k \geq 4$, so it cannot work at the default parameter setting.) It clarifies that SSA with $s = 0.01|P|$ needs the longest time among the three Get-Threshold functions proposed in Section III-C. This is reasonable, because a small number of samples yields a loose threshold, which increases the number of k-cliques enumerated. Recall that the number of k-clique candidates can be $O(n^k)$. Nevertheless, SA, SSA with $s = 0.1|P|$, and PQA are significantly fast: they need only a few hundred milliseconds to retrieve top-1000 5-cliques. Among them, we see that PQA is the fastest. It computes a tight threshold efficiently, so its the other time is the shortest. It is also clear that Brute-force is significantly slower than the other algorithms. We terminated Brute-force before it finishes enumerating all k-cliques, because Brute-force did not do this within reasonable time. Actually, just building a spatial neighbor graph already incurred more than 25 seconds. It can be seen that PQA is x147000 (x65000) faster than Brute-force on CaStreet (Places). Hence, we did not use Brute-force as a baseline in the subsequent experiments.

In Fig. 2, we investigate the efficiency of the pruning and early termination techniques proposed in Section III-E. Note that "time to compute threshold" and "the other time" respectively represent the running time of line 6 of Algorithm 1 and that of lines 10–15. We used SA and PQA to do this, and SA (PQA) w/o pruning represents SQ (PQA) that does not employ these techniques. Fig. 2 shows that the total running time of SA (PQA) w/o pruning is longer than that of SA (PQA). This result demonstrates that the techniques contribute to the efficiency improvement.

Impact of k. We show how the clique size k affects the performance of each algorithm in Fig. 3. First, we focus on the $k = 3$ case. On CaStreet, Triangle has a similar performance to that of PQA, whereas our algorithms are much faster than Triangle on Places. For example, PQA is one order of magnitude faster than Triangle. A possible reason for this result is that our pruning and early termination techniques function better on Places than on CaStreet. Since PQA is general to any k, its advantage (against Triangle) is clear.

We observe that each algorithm does not have an exponential growth of running time with increase of k. In addition, we see that PQA is the most robust against k and fastest in most cases. On CaStreet, its running time is around 100 [msec] for $k \in [3, 7]$. On Places, on the other hand, its running time scales only *linearly* to k (note that the plot is log-scale). Furthermore, by comparing SA (PQA) with SA (PQA) w/o pruning, we see that the impact of our pruning and early termination techniques becomes larger when k is relatively large (e.g., k is 6 and 7).

Impact of $|P|$. We next study the scalablity to the cardinality of a given dataset by using random sampling. Fig. 4 shows that, on CaStreet (Places), SA, SSA with $s = 0.1|P|$, and PQA scale linearly (or sub-linearly) to $|P|$. This result clarifies the efficiencies of these algorithms. (Actually, this result is theoretically reasonable, because, for a fixed k, i.e., $k = O(1)$, G_r has $O(n^k) = O(n)$ k-cliques.)

On the other hand, SSA with $s = 0.01|P|$ has a different behavior. For example, its running time on CaStreet with 1 million points is longer than that on CaStreet with 2.6 million points. Since this algorithm uses only a small number of samples, it cannot obtain a tight threshold when $|P|$ is small. Because of this, SSA with $s = 0.01|P|$ took this counter-intuitive behavior.

Impact of N. Last, we study the influence of the output size N. Fig. 5 shows our experimental results.

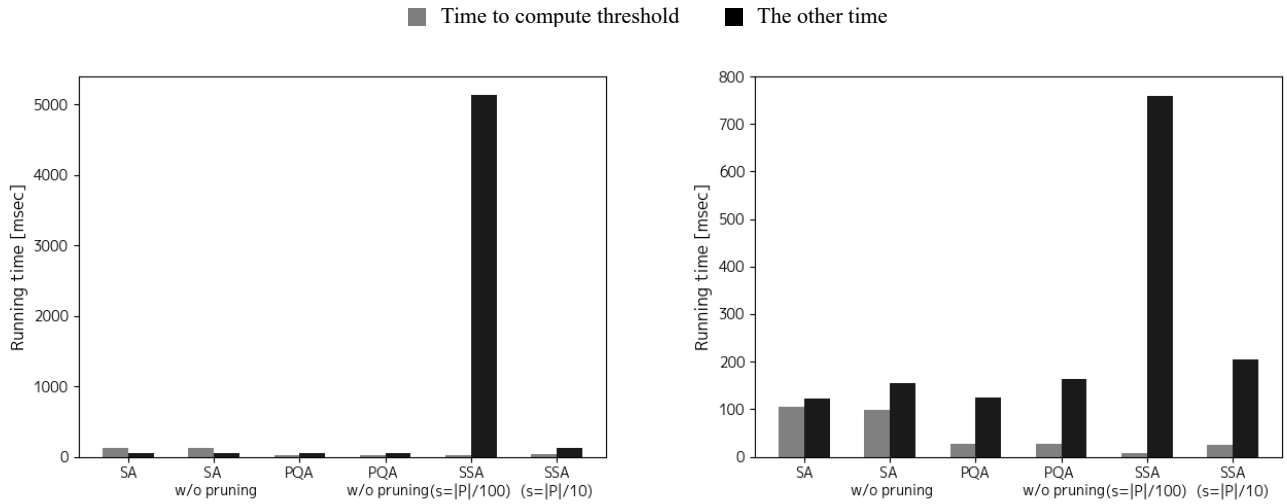
When $N = 100$, PQA and SSA have similar running times. However, as N increases, the performance difference between PQA and SSA becomes larger, suggesting that the priority-queue-based threshold computation yields a clear advantage. On the other hand, SA is (much) slower than PQA when N is small. This is because, when N is small, the main bottleneck of SA is Get-Threshold-with-Sorting, which incurs a larger cost than Get-Threshold-with-Priority-Queue.

Let us focus on a large N. In this case, SA and PQA show similar performances. This is also reasonable, because when N is large, the main bottleneck of these algorithms is Verify.

V. RELATED WORK

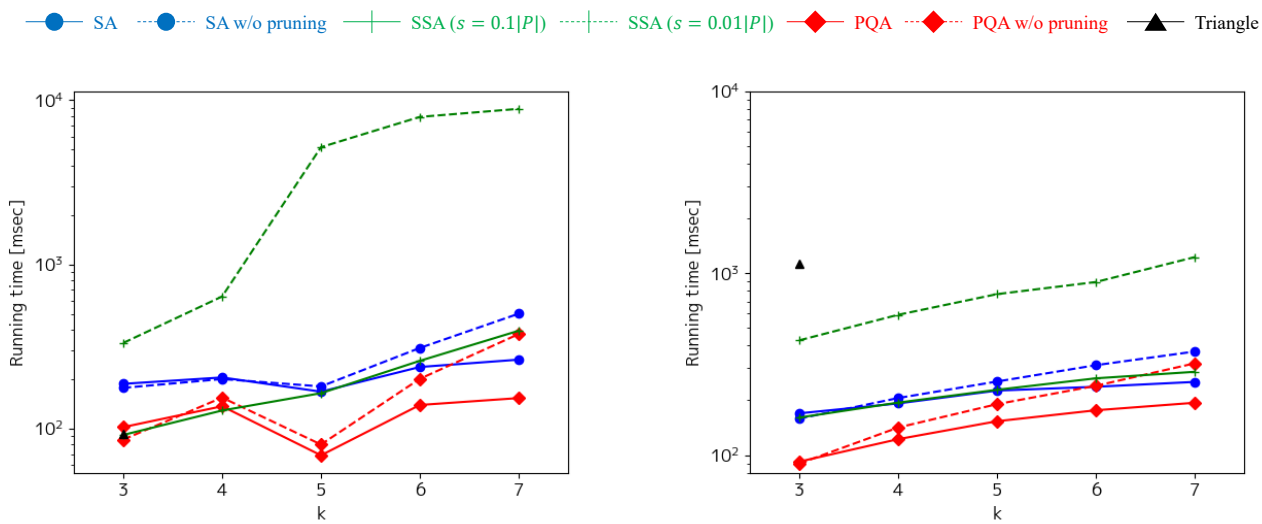
A. Spatial data analysis

As mentioned in Section I, spatial datasets are generated in many environments. To find interesting patterns, knowledge,



(a) CaStreet (b) Places

Fig. 2. Decomposed time [msec]



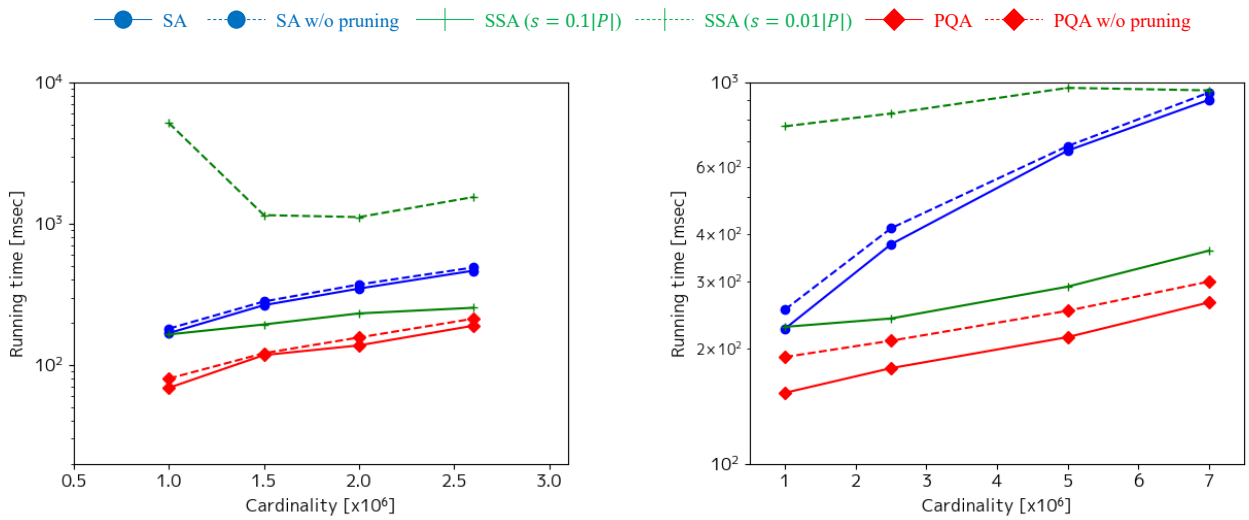
(a) CaStreet (b) Places

Fig. 3. Impact of k

and emerging events hidden in spatial datasets, existing works developed many problems, such as query processing [34], [35], minimum spanning tree construction [36], [37], and visualization [38], [39]. Moreover, to support efficient processing of big spatial data, some works designed (distributed) systems [6], [8], [11], [40] and machine learning models [41]–[44]. As this paper considers a spatial neighbor graph to find interesting patterns, we review existing works on spatial graphs. (Some studies, e.g., [26], [45], propose efficient spatial graph building approaches, but they do not consider finding patterns in spatial graphs. In addition, studies on road networks, e.g., [46], are different from our work, because road networks have different constraints.)

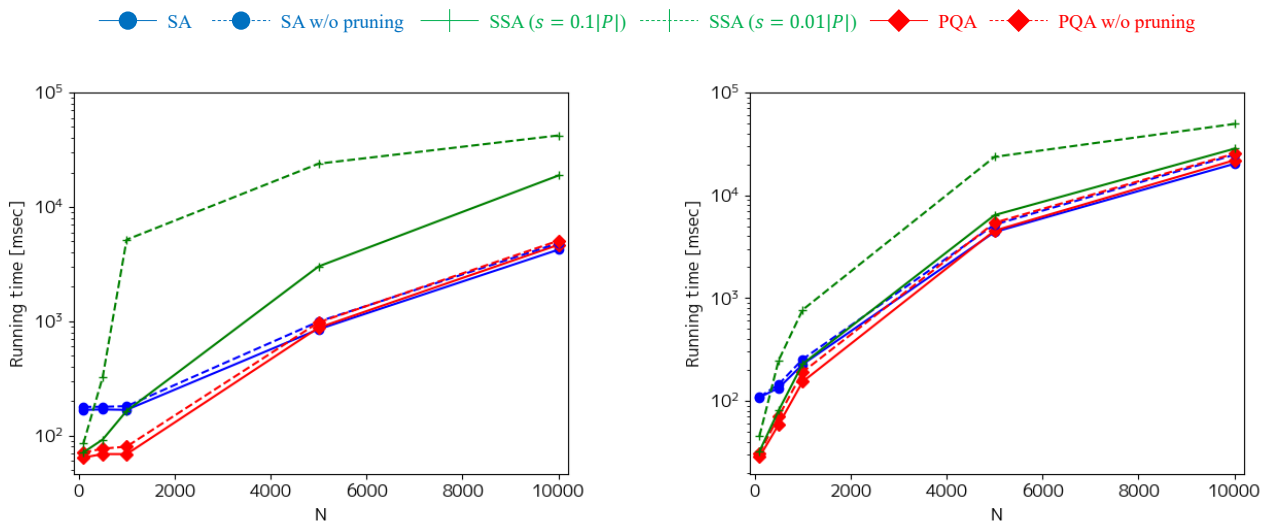
The problem of enumerating all maximal spatial cliques in a spatial neighbor graph G_r was addressed in [24]. Different from our work, this problem cannot control both output and clique sizes and cannot be solved in polynomial time. This may overwhelm users. For example, the experimental results of [24] report that the output size can be million-scale, and it is not trivial to deal with such a large-scale output (particularly for ordinal users). Literature [30] also considers spatial cliques but under a different assumption (i.e., it considers fuzzy model). Similar to [24], the problem in [30] cannot control both output and clique sizes.

Given a sub-graph, [47] (resp. [23]) tackled the problem of finding all (resp. top-N) subsets of P that match the given



(a) CaStreet (b) Places

Fig. 4. Impact of $|P|$



(a) CaStreet (b) Places

Fig. 5. Impact of N

sub-graph pattern. This problem is also NP-hard, so [23], [47] devised practically-efficient algorithms. Since this problem requires a graph pattern (i.e., sub-graph) as an input, some interesting (or required) pattern needs to be known in advance. It is clear, from this fact, that this pattern matching problem targets different applications.

Literature [19] is the work most related to this paper. It addressed the problem of retrieving top-N weighted spatial triangles. To efficiently solve this problem, [19] proposed an algorithm that exploits a pruning technique based on triangle inequality. Its theoretical analysis (under a practical assumption) demonstrates that the algorithm scales linearly to $n = |P|$. It is trivial that this problem is a special case of the problem in our paper, as triangles are 3-cliques. (In other words, our problem generalizes the problem of [19].) Recall that we compared the performance of our algorithm with that of the algorithm proposed in [19], and the result shows the advantage of our algorithm (i.e., PQA). Literature [20] assumes that P is dynamic and addressed the problem of monitoring top-N weighted spatial triangles. We consider such a dynamic dataset case as future work.

B. Clique enumeration in graph databases

Similar to [24], the problem of clique enumeration has been studied extensively under sequential [48] and parallel computation setting [49], because it has important applications, e.g.,

social network analysis [50]. Many works focus on maximal clique or k-clique enumeration.

Because of the NP-hardness, maximal clique enumeration is a computationally-expensive task. Therefore, parallel enumeration algorithms have been extensively developed [49], [51], [52]. As noted earlier, even if we have a spatial neighbor graph G_r of P, these algorithms are not available to compute the top-N k-cliques.

Similar to maximal cliques, k-cliques are also important for finding dense sub-graphs [53], [54]. There are some studies that enumerate all k-cliques [33], [55]–[57]. If we run one of these algorithms after building a spatial neighbor graph G_r of P, we can compute the top-N k-cliques. However, building G_r already incurs a larger computational cost. In addition, all k-clique enumeration incurs a larger computational cost, even when we use the stat-of-the-art [33]. This observation suggests that this approach is not appropriate for efficiently solving our problem. Our experimental result confirmed this, see Section IV-B.

VI. Conclusion & Future Work

Due to the importance of big spatial data analysis, many works proposed (i) problems that effectively analyze/mine spatial data and (ii) efficient algorithms for the problems. This paper also contributed to this trend. Specifically, we addressed a new problem, namely top-N weighted spatial k-clique retrieval, and proposed a practically-efficient algorithm. This algorithm exploits a threshold-based iterative filter-and-verify approach. To compute a tight threshold efficiently, we proposed three techniques. Furthermore, to enable efficient verification, we proposed two techniques that avoid unnecessary k-clique enumeration. We conducted experiments using two real spatial datasets, and the results show that (i) our algorithm is efficient, (ii) priority queue-based threshold computation is effective, and (ii) our verification technique improves the efficiency.

As this paper is the first work for the top-N weighted spatial k-clique retrieval problem, there are some future works. In this paper, we focued on *practical* implementations to solve the problem, and designing theoretical algorithms that can reduce the upper-bound time complexity is an open problem. Moreover, this paper considered a static dataset, but some scenarios need to deal with dynamic (or streaming) spatial data. In such a dynamic data case, we need to monitor the up-to-date result in real-time. How to deal with this case efficiently is also an open problem.

Acknowledgements

This research is partially supported by JST PRESTO Grant Number JPMJPR1931, JSPS Grant-in-Aid for Scientific Research (A) Grant Number 18H04095, and JST CREST Grant Number JPMJCR21F2.

References

[1] D. Amagata and T. Hara, "Monitoring maxrs in spatial data streams." in *EDBT*, 2016, pp. 317–328.

[2] S. Nishio, D. Amagata, and T. Hara, "Geo-social keyword top-k data monitoring over sliding window," in *DEXA*, 2017, pp. 409–424.

[3] D. Amagata and T. Hara, "A general framework for maxrs and maxcrs monitoring in spatial data streams," *ACM Transactions on Spatial Algorithms and Systems*, vol. 3, no. 1, pp. 1–34, 2017.

[4] V. Pandey, A. Kipf, T. Neumann, and A. Kemper, "How good are modern spatial analytics systems?" *PVLDB*, vol. 11, no. 11, pp. 1661–1673, 2018.

[5] N. Taguchi, D. Amagata, and T. Hara, "Geo-social keyword skyline queries," in *DEXA*, 2017, pp. 425–435.

[6] A. Eldawy and M. F. Mokbel, "Spatialhadoop: A mapreduce framework for spatial data," in *ICDE*, 2015, pp. 1352–1363.

[7] Y. Nakayama, D. Amagata, and T. Hara, "An efficient method for identifying maxrs location in mobile ad hoc networks," in *DEXA*, 2016, pp. 37–51.

[8] J. Yu and M. Sarwat, "Geosparkviz: a cluster computing system for visualizing massive-scale geospatial data," *The VLDB Journal*, vol. 30, no. 2, pp. 237–258, 2021.

[9] R. R. Vatsavai, A. Ganguly, V. Chandola, A. Stefanidis, S. Klasky, and S. Shekhar, "Spatiotemporal data mining in the era of big spatial data: algorithms and applications," in *SIGSPATIAL Workshop*, 2012, pp. 1–10.

[10] S. V. Mehta, S. Sodhani, and D. Patel, "Spatial co-location pattern mining-a new perspective using graph database," *arXiv preprint arXiv:1810.09007*, 2018.

[11] J. Yu, J. Wu, and M. Sarwat, "Geospark: A cluster computing framework for processing large-scale spatial data," in *SIGSPATIAL*, 2015, pp. 1–4.

[12] D. Amagata, T. Hara, and S. Nishio, "Distributed top-k query processing on multi-dimensional data with keywords," in *SSDBM*, 2015, pp. 10:1–10:12.

[13] K. Feng, G. Cong, C. S. Jensen, and T. Guo, "Finding attribute-aware similar region for data analysis," *PVLDB*, vol. 12, no. 11, pp. 1414–1426, 2019.

[14] Y. Nakayama, D. Amagata, and T. Hara, "Probabilistic maxrs queries on uncertain data," in *DEXA*, 2017, pp. 111–119.

[15] S. Tsuruoka, D. Amagata, S. Nishio, and T. Hara, "Distributed spatial-keyword knn monitoring for location-aware pub/sub," in *SIGSPATIAL*, 2020, pp. 111–114.

[16] Y. Chen, Z. Chen, G. Cong, A. R. Mahmood, and W. G. Aref, "Sstd: a distributed system on streaming spatio-textual data," *PVLDB*, vol. 13, no. 12, pp. 2284–2296, 2020.

[17] D. Amagata, S. Tsuruoka, Y. Arai, and T. Hara, "Feat-sksj: Fast and exact algorithm for top-k spatial-keyword similarity join," in *SIGSPATIAL*, 2021, pp. 15–24.

[18] D. Amagata, Y. Sasaki, T. Hara, and S. Nishio, "Probabilistic nearest neighbor query processing on distributed uncertain data," *Distributed and Parallel Databases*, vol. 34, no. 2, pp. 259–287, 2016.

[19] R. Taniguchi, D. Amagata, and T. Hara, "Efficient retrieval of top-k weighted spatial triangles," in *DASFAA*, 2022, pp. 224–231.

[20] ——, "Efficient retrieval of top-k weighted triangles on static and dynamic spatial data," *IEEE Access*, vol. 10, pp. 55 298–55 307, 2022.

[21] Y. Fang, R. Cheng, X. Li, S. Luo, and J. Hu, "Effective community search over large spatial graphs," *PVLDB*, vol. 10, no. 6, pp. 709–720, 2017.

[22] Y. Fang, Z. Wang, R. Cheng, X. Li, S. Luo, J. Hu, and X. Chen, "On spatial-aware community search," *IEEE Transactions on Knowledge and Data Engineering*, vol. 31, no. 4, pp. 783–798, 2018.

[23] Y. Fang, Y. Li, R. Cheng, N. Mamoulis, and G. Cong, "Evaluating pattern matching queries for spatial databases," *The VLDB Journal*, vol. 28, no. 5, pp. 649–673, 2019.

[24] C. Zhang, Y. Zhang, W. Zhang, L. Qin, and J. Yang, "Efficient maximal spatial clique enumeration," in *ICDE*, 2019, pp. 878–889.

[25] J. Kim, T. Guo, K. Feng, G. Cong, A. Khan, and F. M. Choudhury, "Densely connected user community and location cluster search in location-based social networks," in *SIGMOD*, 2020, pp. 2199–2209.

[26] Y. Wang, S. Yu, L. Dhulipala, Y. Gu, and J. Shun, "Geograph: A framework for graph processing on geometric data," *ACM SIGOPS Operating Systems Review*, vol. 55, no. 1, pp. 38–46, 2021.

[27] L. Chen, C. Liu, R. Zhou, J. Li, X. Yang, and B. Wang, "Maximum co-located community search in large scale social networks," *PVLDB*, vol. 11, no. 10, pp. 1233–1246, 2018.

[28] Y. Du and C. Yan, "An improved clique-based method for discovery of novel spatial motifs in protein structures," in *BIBE*, 2018, pp. 1–5.

[29] X. Bao and L. Wang, "A clique-based approach for co-location pattern mining," *Information Sciences*, vol. 490, pp. 244–264, 2019.

[30] Z. Hu, L. Wang, V. Tran, and H. Chen, "Efficiently mining spatial co-location patterns utilizing fuzzy grid cliques," *Information Sciences*, vol. 592, pp. 361–388, 2022.

[31] P. Yang, L. Wang, and X. Wang, "A mapreduce approach for spatial co-location pattern mining via ordered-clique-growth," *Distributed and Parallel Databases*, vol. 38, no. 2, pp. 531–560, 2020.

[32] J. L. Bentley, "Multidimensional binary search trees used for associative searching," *Communications of the ACM*, vol. 18, no. 9, pp. 509–517, 1975.

[33] R.-H. Li, S. Gao, L. Qin, G. Wang, W. Yang, and J. X. Yu, "Ordering heuristics for k-clique listing," *PVLDB*, vol. 13, no. 11, pp. 2536–2548, 2020.

[34] S. Nishio, D. Amagata, and T. Hara, "Lamps: Location-aware moving top-k pub/sub," *IEEE Transactions on Knowledge and Data Engineering*, vol. 34, no. 1, pp. 352–364, 2022.

[35] D. Amagata and T. Hara, "Identifying the most interactive object in spatial databases," in *ICDE*, 2019, pp. 1286–1297.

[36] S. Chatterjee, M. Connor, and P. Kumar, "Geometric minimum spanning trees with GEOFILTERKRUSKAL," in *SEA*, 2010, pp. 486–500.

[37] W. B. March, P. Ram, and A. G. Gray, "Fast euclidean minimum spanning tree: algorithm, analysis, and applications," in *KDD*, 2010, pp. 603–612.

[38] T. Guo, K. Feng, G. Cong, and Z. Bao, "Efficient selection of geospatial data on maps for interactive and visualized exploration," in *SIGMOD*, 2018, pp. 567–582.

[39] T. N. Chan, R. Cheng, and M. L. Yiu, "Quad: quadratic-bound-based kernel density visualization," in *SIGMOD*, 2020, pp. 35–50.

[40] Y. Fang, R. Cheng, J. Wang, L. Budiman, G. Cong, and N. Mamoulis, "Spacekey: exploring patterns in spatial databases," in *ICDE*, 2018, pp. 1577–1580.

[41] S. Wang, J. Cao, and P. Yu, "Deep learning for spatio-temporal data mining: A survey," *IEEE Transactions on Knowledge and Data Engineering*, vol. 34, no. 8, pp. 3681–3700, 2022.

[42] D. Amagata, Y. Arai, S. Fujita, and T. Hara, "Learned k-nn distance estimation," in *SIGSPATIAL*, 2022.

[43] J. Qi, G. Liu, C. S. Jensen, and L. Kulik, "Effectively learning spatial indices," *PVLDB*, vol. 13, no. 12, pp. 2341–2354, 2020.

[44] P. Li, H. Lu, Q. Zheng, L. Yang, and G. Pan, "Lisa: A learned index structure for spatial data," in *SIGMOD*, 2020, pp. 2119–2133.

[45] Y. Tong, J. She, B. Ding, L. Chen, T. Wo, and K. Xu, "Online minimum matching in real-time spatial data: experiments and analysis," *PVLDB*, vol. 9, no. 12, pp. 1053–1064, 2016.

[46] M. Zhang, L. Li, W. Hua, R. Mao, P. Chao, and X. Zhou, "Dynamic hub labeling for road networks," in *ICDE*, 2021, pp. 336–347.

[47] Y. Fang, R. Cheng, G. Cong, N. Mamoulis, and Y. Li, "On spatial pattern matching," in *ICDE*, 2018, pp. 293–304.

[48] Z. Chen, L. Yuan, X. Lin, L. Qin, and J. Yang, "Efficient maximal balanced clique enumeration in signed networks," in *The Web Conference*, 2020, pp. 339–349.

[49] A. Das, S.-V. Sanei-Mehri, and S. Tirthapura, "Shared-memory parallel maximal clique enumeration," in *HiPC*, 2018, pp. 62–71.

[50] J. Cheng, Y. Ke, A. W.-C. Fu, J. X. Yu, and L. Zhu, "Finding maximal cliques in massive networks," *ACM Transactions on Database Systems*, vol. 36, no. 4, pp. 1–34, 2011.

[51] A. Das, S.-V. Sanei-Mehri, and S. Tirthapura, "Shared-memory parallel maximal clique enumeration from static and dynamic graphs," *ACM Transactions on Parallel Computing*, vol. 7, no. 1, pp. 1–28, 2020.

[52] Y.-W. Wei, W.-M. Chen, and H.-H. Tsai, "Accelerating the bron-kerbosch algorithm for maximal clique enumeration using gpus," *IEEE Transactions on Parallel and Distributed Systems*, vol. 32, no. 9, pp. 2352–2366, 2021.

[53] C. Tsourakakis, "The k-clique densest subgraph problem," in *World Wide Web*, 2015, pp. 1122–1132.

[54] B. Sun, M. Danisch, T. H. Chan, and M. Sozio, "Kclist++: A simple algorithm for finding k-clique densest subgraphs in large graphs," *PVLDB*, vol. 13, no. 10, pp. 1628–1640, 2020.

[55] N. Chiba and T. Nishizeki, "Arboricity and subgraph listing algorithms," *SIAM Journal on computing*, vol. 14, no. 1, pp. 210–223, 1985.

[56] M. Danisch, O. Balalau, and M. Sozio, "Listing k-cliques in sparse real-world graphs," in *World Wide Web*, 2018, pp. 589–598.

[57] I. Finocchi, M. Finocchi, and E. G. Fusco, "Clique counting in mapreduce: Algorithms and experiments," *Journal of Experimental Algorithmics*, vol. 20, pp. 1–20, 2015.