Example-based Spatial Pattern Matching

Yue Chen
Nanyang Technological University
yue004@e.ntu.edu.sg

Kaiyu Feng*
Nanyang Technological University
kfeng002@e.ntu.edu.sg

Gao Cong*
Nanyang Technological University
gaocong@ntu.edu.sg

Han Mao Kiah
Nanyang Technological University
hmkiah@ntu.edu.sg

ABSTRACT

The prevalence of GPS-enabled mobile devices and location-based services yield massive volume of spatial objects where each object contains information including geographical location, name, address, category and other attributes. This paper introduces a novel type of query termed *example-based spatial pattern matching* (EPM) query. It takes as input a set of spatial objects, each of which is associated with one or more keywords and a location. These objects serve as an example that depicts the spatial pattern that users want to retrieve. The EPM query returns all sets of objects that match the spatial pattern. The EPM query can be used for applications like urban planning, scene recognition and similar region search. We propose an efficient algorithm and three pruning techniques to answer EPM queries. Furthermore, we provide an approximation guarantee for intermediate results of the algorithm. Our experimental evaluations on four real-world datasets demonstrate the effectiveness and efficiency of our proposed algorithm and techniques.

PVLDB Reference Format:
Yue Chen, Kaiyu Feng, Gao Cong, and Han Mao Kiah. Example-based Spatial Pattern Matching. PVLDB, 15(11): 2572 - 2584, 2022.
doi:10.14778/3551793.3551815

1 INTRODUCTION

With the proliferation of GPS-enabled mobile devices and location-based services, massive volume of spatial objects are being generated rapidly. For example, Gowalla [19] and Foursquare [26] contain 2.7 million and 10.1 million points of interest (POIs), respectively. Each POI has information such as geographical location, name, address, category and other category-specific attributes. Users can pose spatial keyword queries [6–10, 16, 22, 23, 25, 27] on such databases to search for their desired POIs. For example, *What are the POIs with category "shopping mall" in this region?*

Although spatial keyword queries are useful for many applications, they would not be able to meet the need of many other occasions. A prevalent requirement is to search several objects with desired attributes and relative spatial locations. Consider the following three motivating examples.

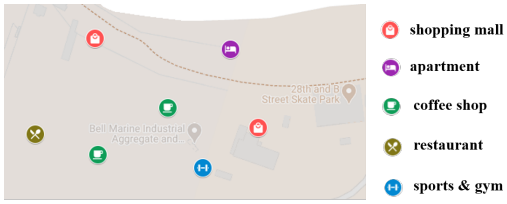


Figure 1: An example of spatial pattern

Example 1: Scene recognition [17]. It aims to find scenes when only attributes of several objects and their approximate relative locations are given, but no addresses are given. For example, Julia wants to identify a place in a foreign city that she visited many years ago. She cannot remember the name and the address, which are in foreign languages, but she still recalls that "there was a coffee shop, when she went out the coffee shop, she walked along the street around 1km and entered a Chinese restaurant, and she could see a car park on her right hand side about 300 meters away". Scene recognition may have other application scenarios, e.g., a person may read a traveling blog mentioning the general categories of the locations and their approximate relative positions but without specific location names, and she would like to find out the places.

Example 2: Similar region search [20]. It may have applications in Urban planning, where it aims to find similar layouts of facilities for a given draft layout design. For instance, inappropriate layouts of public facilities have potential danger in urban planning [1]. When government plans to develop a new district, they could pose queries on databases to check if their devised draft layouts (i.e., described by a set of facilities and their approximate relative locations) have potential danger. As another example, landscape design concerns the placement of car parks, swimming pools and other facilities in a region. Given a draft design, the designer wants to retrieve the existing designs with similar layouts for the purposes such as intellectual property protection or improving the draft design. Another example could be improving services like POI recommendation [18] and region recommendation [24] by searching the regions with similar distribution of POIs as the previous visited region of the user.

Example 3: Pattern recognition. It aims to find patterns in which the approximate relative positions of objects are preserved. For example, sports like football, basketball, hockey and rugby have tactics such as attack and defense. The positions of different players (e.g., center forward, defensive midfielder, center attacking midfielder, etc.) usually reveal their tactics. The players can review and identify their tactics by retrieving similar position layouts from the

*Corresponding authors

Proceedings of the VLDB Endowment, Vol. 15, No. 11 ISSN 2150-8097.
doi:10.14778/3551793.3551815

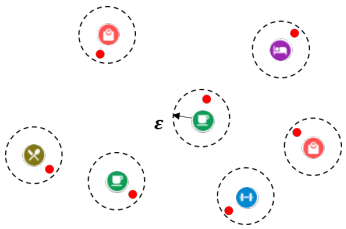


Figure 2: Matching under tolerance $\varepsilon = 0.5km$

snapshots or video frames of historical competitions [1]. As another example, in astro-photography stacking and alignment, given a set of astral images and a reference image, it aims to find all astral images which exhibit the similar layout as the reference image to enable the recovery of the complete astral images [2]. Other examples may include searching objects of similar distributions from images after computer vision techniques are applied to identify objects from images.

In the aforementioned examples, there is one running theme throughout the problems encountered: we have an example-based spatial pattern that defines the relative locations of a set of spatial objects together with their categories. We aim to retrieve the sets of objects that exhibit a similar pattern. Note that in these examples, a user often needs to repeat the following: issue a query and explore the returned results, and modify the query. The process may be repeated many times before the user gets satisfactory results. This problem cannot be well addressed by existing techniques designed for pattern-based queries, including graph-based spatial pattern matching [11] and spatial exemplary query [21], as to be discussed in Section 2.

Inspired by these example applications, in this paper, we propose a novel type of query termed example-based spatial pattern matching (EPM) query. It takes as input a set of example (or target) spatial objects that depict the spatial pattern that users want to retrieve, and returns all sets of spatial objects that match the spatial pattern. Each example (or target) object is associated with positional information and at least one category. For example, Figure 1 shows an EPM query consisting of 7 example objects, each of which contains one category and one geolocation. However, given such an EPM query, it may not always be possible to find 7 objects from the database such that they contain the specified categories and exhibit exactly the same geometrical shape as Figure 1. To support fault tolerance and make EPM query more realistic, we adopt a new criterion for matching from the problems of *points matching under noise regions* [2, 3, 15] by the community of computational geometry, in which *matching under tolerance* ε is defined as follow: *Given two sets A, B of n points each, if there exists a congruence that maps in a 1-to-1 fashion the points of B into the* ε*-neighbourhood of points of A, then we say A matches B under tolerance* ε.

We follow this definition to decide if a set of spatial objects in the database matches the target objects in an EPM query geographically. In Figure 2, 7 objects in red dots constitute a match under tolerance $\varepsilon = 0.5$km for the EPM query in Figure 1. EPM aims to return all sets of objects that match the query under tolerance ε.

[1] http://outsideoftheboot.com/

[2] https://www.mathworks.com/matlabcentral/fileexchange/65328-astrotnstack-align-and-stack-astro-photography-pictures

To answer the EPM query, an intuitive idea is as follow: we start from one target object of the EPM query, then find the candidate objects with the same category from the database. For each candidate object, we check its neighbourhood to see if there exist objects with same categories as the remaining target objects. If so, we determine whether they match the EPM query under tolerance ε. We propose an efficient algorithm to implement this process. To further improve the performance, we propose three advanced techniques to prune the unpromising candidate objects, reduce the search space, and avoid the same match being found multiple times. Moreover, the intermediate results of our algorithm can be considered as approximate results of the EPM query and we establish an approximation guarantee for the intermediate results of our algorithm.

In summary, the key contributions of this paper are as follows.

- We propose a novel type of query termed example-based spatial pattern matching (EPM) query, and illustrate its applications. To the best of our knowledge, it is the first work to adopt the *matching under tolerance* ε in POI retrieval.
- We propose an efficient algorithm for the EPM query and 3 advanced pruning techniques to improve the algorithm. In addition, we establish an approximation guarantee for the intermediate results of our algorithm, which can serve as approximate results when higher efficiency is needed.
- We conduct extensive experiments on 4 real-world datasets to evaluate the effectiveness and efficiency of our proposed algorithm. Furthermore, we demonstrate that our approximate solution is two orders of magnitude faster than the exact solution and returns matches with comparable quality.

In the rest of this paper, Section 2 reviews the related work and Section 3 gives problem definition. We present the proposed algorithm in Section 4 and three pruning techniques in Section 5. In Section 6, approximation guarantee is provided. Section 7 reports the experimental evaluations, and Section 8 offers conclusions.

2 RELATED WORK

Spatial Keyword Query. A spatio-textual object o consists of geographical location $o.loc$ and a set of keywords $o.text$. Spatial keyword queries aim to retrieve such spatio-textual objects of interest. Most types of spatial keyword queries [6, 8–10, 16, 22, 23, 25, 27] aim to return a single object as a result. For example, boolean range keyword query [6, 8, 9, 16, 22, 23, 25] takes as input a region and a set of keywords, and returns all the objects each of which is located inside the region and contains all the query keywords. As another example, location-aware top-k text retrieval query [10] takes as input a query location and a set of query keywords, and returns the top-k ranked objects, where the ranking score of each object is computed by a linear combination of the spatial proximity and the text relevancy between the object and the query. Some spatial keyword queries consider the nearby objects when retrieving objects to better meet users' need. For example, top-k prestige-based relevant spatial web object retrival [4] considers the nearby relevant objects in ranking objects. As another example, clue-based spatio-textual query [17] takes as input a querying POI along with its nearby POIs (called *clue* POIs), and returns top-k POIs which have the same category as querying POI and have the highest spatio-textual

context similarities against the clue POIs. Several other proposals [5, 13, 28, 29] aim to return a set of objects as a result to the user query. For example, m-closest keywords query [13, 28] takes as input a set of query keywords, and returns a set of objects such that they collectively cover all the query keywords and the maximum distance between any pair of them is minimized.

Our EPM query differs from these spatial keyword queries. They rank the objects with smaller distance to the query point higher, or the group of objects close to each other higher. However, our EPM query cares about the relative locations between the returned spatial objects. Therefore, the query processing algorithms for these queries cannot be extended to answer the EPM query.

Spatial Pattern Matching Query. This type of query aims to retrieve a set of objects satisfying the query pattern. First, SPM (graph-based spatial pattern matching query) [11] takes as input a graph which consists of a set of target objects along with distance constraints between them. For example, a *house* is 2–5km away from a *school*, and is at least 3km from a *hospital*. SPM aims to return all instances $(house, school, hospital)$ satisfying these distance constraints. SPM does not care about the relative locations of target objects, which makes it difficult for users to compose such SPM queries to reflect their intentions. If we adapt SPM to solve the aforementioned three motivating examples, the drawbacks are three folds. (i) If the user requires n target objects in the spatial pattern, then he needs to specify $\frac{n(n-1)}{2}$ distance constraints for all pairs of target objects. It would be troublesome for users to compose so many distance constraints. (ii) Worse still, these distance constraints cannot control the geometrical shape of the target objects and do not preserve the relative locations of them. It would bring matches of low quality in the returned objects. We illustrate this observation in the case study in Section 7.2. (iii) SPM is proven to be NP-hard and is rather inefficient when pattern size is large. We adapt SPM to serve as a baseline in our experiments. Compared with SPM, our EPM query is user-friendly to compose, generates matches of high quality, and is efficient even if pattern size is large.

Second, Luo et al. introduce SEQ (spatial exemplary query) in a poster paper [21], which takes as input a set of objects together with the Euclidean distance between each pair of them. Similar to previous spatial keyword query, SEQ uses a linear combination of spatial proximity and textual relevancy to measure the similarity between a set of objects and the query, whose semantics is totally different from the *matching under tolerance* ε in EPM. For example, an SEQ query could be {("*house*", "*school*", "*hospital*"); "*house*" is 2km away from "*school*" and 2.5km away from "*hospital*", "*school*" is 3km away from "*hospital*"}. Under the definition of SEQ, a set of objects {("*restaurant*", "*school*", "*hospital*"); "*restaurant*" is 2.5km away from "*school*" and 2km away from "*hospital*", "*school*" is 3.5km away from "*hospital*"} could be returned in the top-k results since its scores of spatial similarity and textual similarity are moderate. However, it will never happen in our EPM query because their categories do not match in a one-to-one manner and SEQ does not take into account the tolerance ε either. Clearly, SEQ cannot solve our motivating examples precisely. Furthermore, it is unclear how to specify the parameters in SEQ (e.g., relative weights in scoring functions and the number of results to be returned), which makes it difficult to adapt SEQ to solve EPM.

Spatial Points Matching. Alt et al. [2] study the problem of finding the geometric transformations (rotation, translation and reflection) that map a point set A approximately into the neighbourhood of a point set B. Arkin et al. [3] consider the points matching problem that aims to find a transformation of a set of n points such that each transformed point lies in one of n given pairwise-disjoint noise regions. Heffernan et al. [14] propose an efficient approximate algorithm to test whether two equal cardinality point sets A and B in the plane are ε-congruent.

The idea of "matching under tolerance" is inspired by these proposals. The difference is that, they focus on determining if two sets of points match under tolerance, whereas EPM aims to find out all sets of POIs, each of which matches the given pattern. Their solutions inspire us to design the baseline algorithm for the EPM.

3 PROBLEM DEFINITION

Let D be a database of spatial objects. Each object $o \in D$ is represented by $(o.\rho, o.\phi)$ where $o.\rho$ is the location of o (i.e., latitude and longitude), and $o.\phi$ is a keyword to indicate the category of o or describe o. For example, Figure 3(a) shows 5 objects in the database, where o_1 is described by its keyword "house". An example-based spatial pattern P is represented by a set $\{p_1, p_2, \cdots, p_n\}$, where each point $p = (p.\rho, p.\Phi) \in P$ has a location $p.\rho$ and a set of keywords $p.\Phi$. Figure 3(b) gives an example $P = \{p_1, p_2, p_3, p_4\}$. The keywords of p_1 are {"house", "apartment"}. The notations used in this paper are summarized in Table 1.

Definition 1. ***Rigid motion***. *Rigid motion denotes a set of transformations, each of which is one of the following operations:*

- *Translation. If a point $p(x, y)$ is translated along vector $\vec{v}(a, b)$, we get its new position $p(x + a, y + b)$.*
- *Rotation. If a point $p(x, y)$ is rotated counterclockwise θ radians based on the center (a, b), we get its new position $p((x-a)\cos\theta - (y-b)\sin\theta + a, (x-a)\sin\theta + (y-b)\cos\theta + b)$.*

Definition 2. ***Match***. *Given an example-based spatial pattern $P = \{p_1, \cdots, p_n\}$, a set O of n objects, and a tolerance ε, we call O a match of P if there exists a rigid motion (i.e., a set of translation and rotation) on P, and a bijection $f : P \to O$, such that after rigid motion, for each $i \in \{1, \cdots, n\}$, we have (1) $f(p_i).\phi \in p_i.\Phi$; and (2) p_i is located inside $C^{\varepsilon}_{f(p_i)}$, which denotes the circle with center $f(p_i)$ and radius ε.*

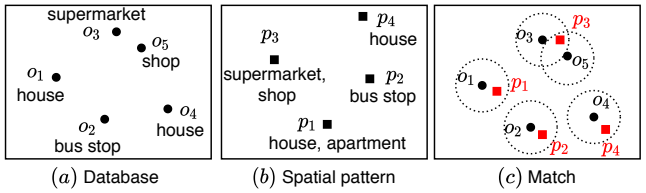
For example, in Figure 3, if we rotate and translate the spatial pattern P in Figure 3(b) to the position in Figure 3(c), we can find a bijection: $P \to \{o_1, o_2, o_3, o_4\}$, i.e., $p_1 \to o_1$, $p_2 \to o_2$, $p_3 \to o_3$, $p_4 \to o_4$, since p_1 (p_2, p_3, p_4, resp.) covers the keyword of o_1 (o_2, o_3, o_4, resp.) and is located inside the circle $C^{\varepsilon}_{o_1}$ ($C^{\varepsilon}_{o_2}, C^{\varepsilon}_{o_3}, C^{\varepsilon}_{o_4}$ resp.). Therefore $\{o_1, o_2, o_3, o_4\}$ is a match of P. Similarly, $\{o_1, o_2, o_5, o_4\}$ is also a match of P.

Definition 3. ***EPM problem***. *Given a database D, an example-based spatial pattern P and a tolerance ε, the EPM problem (denoted by $EPM(P, D, \varepsilon)$) returns all matches of P in D.*

Remark. In this work, we assume each object is associated with only *one* keyword, and each point in the spatial pattern may have *multiple* keywords. However, our work can be easily extended to the case where each object contains *multiple* keywords. We only need

Table 1: Table of notations

Notation	Definition
D	database of spatial objects
o, O	an object, a set of objects
p, P	a point in spatial pattern, a spatial pattern
$o.\rho, p.\rho$	location of o, location of point p
$o.\phi, p.\Phi$	keyword of o, the set of keywords of p
ε-circle	a circle with radius ε
C_o^ε	a circle with center o and radius ε
$D_{p.\Phi}$	a set of objects whose keywords are covered by $p.\Phi$, i.e., $\{o \in D \mid o.\phi \in p.\Phi\}$
n, m, d	$n = \lvert P\rvert$, $m = \lvert D\rvert$, $d = \max_{1\le i\le n} \lvert D_{p_i.\Phi}\rvert$
$\lvert\overline{p_1p_2}\rvert$	length of line segment $\overline{p_1p_2}$ (i.e., Euclidean distance between p_1, p_2)



(a) Database (b) Spatial pattern (c) Match

Figure 3: EPM query

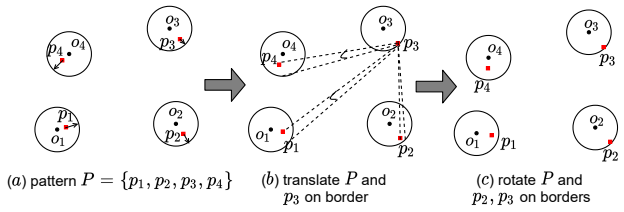


Figure 4: Observation: 2 points onto borders

to change the first condition in Definition 2 as: $f(p_i).\phi \cap p_i.\Phi \neq \emptyset$. Our proposed algorithm (in Section 4) and techniques (in Section 5) are equally applicable to solve this generalized query. Furthermore, each point in the pattern may have different importance/weights, we can allow different tolerances for each point in the EPM. Our proposed algorithm and techniques are also applicable for this case.

4 BASIC FRAMEWORK

To solve $EPM(P, D, \varepsilon)$, intuitively, we need to consider two aspects.

• **A1**. How to efficiently find all sets of objects, such that each set has $|P|$ objects and contains all required keywords in P.

• **A2**. For each set of objects O in A1, how to decide if it matches P geographically.

We may not get too many insights from A1 directly. But when considering A2, we observe that *O matches P geographically, if and only if there exists a rigid motion which maps at least two points p_i, p_j of P onto the borders of $C_{o_i}^\varepsilon$, $C_{o_j}^\varepsilon$, and other points p_k $(k \neq i, j)$ inside $C_{o_k}^\varepsilon$* [2]. We use Figure 4 to illustrate this observation. In Figure 4(a), O matches P and each p_i is located inside $C_{o_i}^\varepsilon$. For each $p_i \in P$, we compute $r_i = \varepsilon - |\overline{o_ip_i}|$, which is the minimum distance that moves p_i onto the border of $C_{o_i}^\varepsilon$. We find $r^* = \min_i\{r_i\}$. In this example, r_3 is the smallest. Then we translate P by r_3 units in direction $\overrightarrow{o_3p_3}$. As a result, we make p_3 onto the border as shown in Figure 4(b). Next, for each $i = \{1, 2, 4\}$, based on rotation center p_3, we rotate $\overline{p_3p_i}$ to make p_i onto the border of $C_{o_i}^\varepsilon$ and record the rotation angle α_i. We find $\alpha^* = \min_i\{\alpha_i\}$. Here, α_2 is the smallest. Then we rotate P by angle α_2 with rotation center p_3. As a result, we make p_2, p_3 onto the borders and other points inside the circles as shown in Figure 4(c). Based on this example, we give a proof sketch of the above observation.

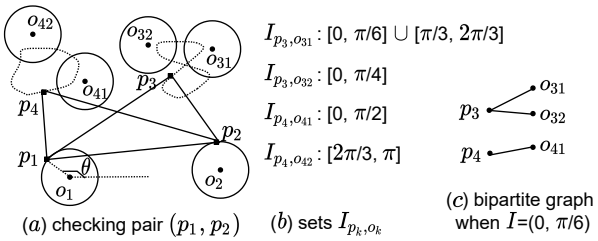


Figure 5: Find matches when checking (p_1, p_2)

- Sufficiency. Given P, O, if at least two points p_i, p_j reside on the borders of $C_{o_i}^\varepsilon$, $C_{o_j}^\varepsilon$, and other points inside their ε-circles, then O matches P by definition.
- Necessity. If O matches P, then we can apply the method in Figure 4 to make at least two points onto the borders of their ε-circles, and others inside the corresponding ε-circles.

Based on this observation, we can find all the matches if we check each pair $(p_i, p_j) \in P$ and let them move on the borders of $C_{o_i}^\varepsilon$, $C_{o_j}^\varepsilon$, where $o_i \in D_{p_i.\Phi}, o_j \in D_{p_j.\Phi}$. Here $D_{p_i.\Phi}$ is the candidate objects of p_i, i.e., the set of objects whose keywords are covered by $p_i.\Phi$, and $D_{p_j.\Phi}$ is defined similarly.

Now $EPM(P, D, \varepsilon)$ becomes: **How to find matches when p_i, p_j move on the borders of $C_{o_i}^\varepsilon, C_{o_j}^\varepsilon$, where $o_i \in D_{p_i.\Phi}, o_j \in D_{p_j.\Phi}$.** We use Figure 5 to illustrate the solution of this problem. In Figure 5(a), suppose $|P| = 4$, we are checking pair (p_1, p_2). We let p_1 run on the border of $C_{o_1}^\varepsilon$, as the distance $|\overline{p_1p_2}|$ is fixed, we can determine p_2's position on the border of $C_{o_2}^\varepsilon$. After p_1, p_2 has been determined, we can determine the positions of other points in P. In other words, the trajectories of p_3, p_4 (dash curves in Figure 5(a)) can be completely determined when p_1 moves on $C_{o_1}^\varepsilon$. We can see that the trajectory of p_3 (p_4 resp.) intersects with a few circles centered at its candidate objects, i.e., $C_{o_{31}}^\varepsilon$, $C_{o_{32}}^\varepsilon$ ($C_{o_{41}}^\varepsilon$, $C_{o_{42}}^\varepsilon$ resp.). We store in the set $I_{p_3,o_{31}}$ ($I_{p_3,o_{32}}$ resp.) the angle θ such that when p_1 has polar coordinates θ, p_3 is located inside $C_{o_{31}}^\varepsilon$ ($C_{o_{32}}^\varepsilon$ resp.). We also compute $I_{p_4,o_{41}}$, $I_{p_4,o_{42}}$ for p_4 as shown in Figure 5(b). For example, $I_{p_4,o_{41}} = [0, \pi/2]$, it indicates p_4 is located inside $C_{o_{41}}^\varepsilon$ when p_1 has coordinate $\theta \in [0, \pi/2]$. These sets I_{p_k,o_k} help to identify the circles where each p_k is located, and thus can be used to find out the matches. Now, it remains to find all ways to assign one of $\{o_{31}, o_{32}\}$ to p_3, and one of $\{o_{41}, o_{42}\}$ to p_4. This can be solved by modelling as a *bipartite graph matching* problem. Specifically, all I_{p_k,o_k}'s partition $[0, 2\pi]$ into a set of disjoint subintervals. For each subinterval, we construct a bipartite graph, and each *maximum cardinality matching* corresponds to a match in $EPM(P, D, \varepsilon)$. For example, in Figure 5(b), 4 sets $I_{p_3,o_{31}}$, $I_{p_3,o_{32}}$, $I_{p_4,o_{41}}$, $I_{p_4,o_{42}}$ partition $[0, 2\pi]$ into 13 subintervals, i.e., $\{0\}$, $(0, \pi/6)$, $\{\pi/6\}$, $(\pi/6, \pi/4)$, $\{\pi/4\}$, $(\pi/4, \pi/3)$, $\{\pi/3\}$, $(\pi/3, \pi/2)$, $\{\pi/2\}$, $(\pi/2, 2\pi/3)$, $\{2\pi/3\}$, $(2\pi/3, \pi)$, $\{\pi\}$. For subinterval $I = (0, \pi/6)$, as I is contained in $I_{p_3,o_{31}}$, $I_{p_3,o_{32}}$ and $I_{p_4,o_{41}}$, we construct a bipartite graph G with 3 edges, namely, (p_3, o_{31}), (p_3, o_{32}) and (p_4, o_{41}), as shown in Figure 5(c). Apparently, two maximum cardinality matchings $\{(p_3, o_{31}), (p_4, o_{41})\}$,

$\{(p_3, o_{32}), (p_4, o_{41})\}$, combined with $\{(p_1, o_1), (p_2, o_2)\}$, form two matches in $EPM(P, D, \varepsilon)$.

Algorithm 1: EPM-Alg(P, D, ε)

Input : Database D, spatial pattern P, tolerance ε
Output: All matches of P

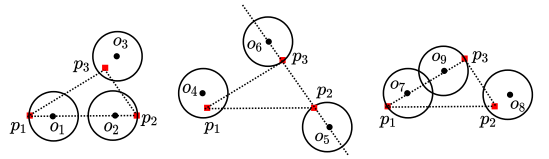
1 initialize the result set R;
2 **foreach** *pair* (p_i, p_j) *of* P **do**
3 $D_{p_i.\Phi} \longleftarrow$ candidate objects of p_i, $\{o | o.\phi \in p_i.\Phi\}$;
4 $D_{p_j.\Phi} \longleftarrow$ candidate objects of p_j, $\{o | o.\phi \in p_j.\Phi\}$;
5 **foreach** *pair* $(o_i, o_j) \in D_{p_i.\Phi} \times D_{p_j.\Phi}$ **do**
6 **foreach** $p_k \in P$ $(k \neq i, j)$ **do**
7 $D_{p_k.\Phi} \longleftarrow$ candidates of p_k, $\{o | o.\phi \in p_k.\Phi\}$;
8 **foreach** $o_k \in D_{p_k.\Phi}$ **do**
9 $I_{p_k,o_k} \longleftarrow$ set of angles θ s.t. when p_i has polar coordinate θ, p_k is located in $C^{\varepsilon}_{o_k}$;
10 sort all I_{p_k,o_k}, partition $[0, 2\pi]$ into subintervals $\mathcal{I}$;
11 **foreach** *subinterval* $I \in \mathcal{I}$ **do**
12 construct bipartite graph $G = (V_1 \cup V_2, E)$, $V_1 = P \setminus \{p_i, p_j\}$, $V_2 = \{o \mid \exists p \in V_1, s.t.\ I \subseteq I_{p,o}\}$, $E = \{(p, o) | p \in V_1, o \in V_2, s.t.\ I \subseteq I_{p,o}\}$;
13 $R \longleftarrow R \cup$ max. matches of size $(n-2)$ in G;
14 **return** R

Algorithm 1 gives the basic framework to solve $EPM(P, D, \varepsilon)$. (The idea of Lines 5–10 is inspired by [2]). First, for each pair (p_i, p_j) of P (Line 2), we fetch their candidate objects $D_{p_i.\Phi}$, $D_{p_j.\Phi}$ (Line 3–4). For each candidate object pair (o_i, o_j), we let p_i, p_j run on the borders of $C^{\varepsilon}_{o_i}$, $C^{\varepsilon}_{o_j}$, respectively (Line 5). For each $p_k \in P (k \neq i, j)$, we get its candidate objects $D_{p_k.\Phi}$, observe its trajectory and compute the sets I_{p_k,o_k} (Line 6–9). These sets $\bigcup_{p_k \neq p_i,p_j} \{I_{p_k,o_k}\}$ partition $[0, 2\pi]$ into a set of disjoint subintervals $\mathcal{I}$, each of which is contained in some source sets I_{p_k,o_k}'s. For each subinterval $I \in \mathcal{I}$, we construct a bipartite graph $G = (V_1 \cup V_2, E)$, where V_1 comprises points of P excluding p_i, p_j, i.e., $V_1 = P \setminus \{p_i, p_j\}$, V_2 comprises the candidate objects associated with V_1 during subinterval I, i.e., $V_2 = \{o \mid \exists p \in V_1, s.t.\ I \subseteq I_{p,o}\}$, and E consists of such pair (p, o) that p is located inside C^{ε}_o during I, i.e., $E = \{(p, o) | p \in V_1, o \in V_2, s.t.\ I \subseteq I_{p,o}\}$ (Line 11–12). Each maximum cardinality matching of size $(|P| - 2)$ in G (i.e., each p_k $(k \neq i, j)$ is assigned an object) corresponds to a match of the EPM problem, and is included into R (Line 13) as a result (Line 14).

Index and Complexity. We use inverted lists to index the objects based on the keywords, to facilitate the retrieval of $D_{p_i.\Phi}$. Let $n = |P|$, $d = \max_{1 \leq i \leq n} |D_{p_i.\Phi}|$. At first, we need to check all pairs of P which require $O(n^2)$ iterations. At each iteration when we check (p_i, p_j), $O(d)$ time is required to retrieve $D_{p_i.\Phi}, D_{p_j.\Phi}$ and $O(d^2)$ object pairs (o_i, o_j) are generated. For each (o_i, o_j), it takes $O(nd)$ time to compute all I_{p_k,o_k}'s [2] and $O(nd \log(nd))$ time to sort them, and $O(nd)$ subintervals are generated. For each subinterval, it takes $O(c \cdot (|V| + |E|) + |V|^{0.5}|E|)$ time to find all maximum cardinality matchings on graph $G(V, E)$ [12]. In our case, $|V| = O(n)$, $|E| = O(nd)$, thus $O(c \cdot nd + n^{1.5}d)$ time is required for one graph G, where c is the number of maximum cardinality matchings in G. As we have $O(nd)$ subintervals, which means we need $O(c \cdot n^2d^2 + n^{2.5}d^2)$ time in this step. Overall, the total complexity is

$$n^2(d + d^2(nd + nd \log nd + c \cdot n^2d^2 + n^{2.5}d^2)) = n^4d^4(c + n^{0.5}).$$

Usually, pattern size n is very small in our real-world applications. Typically, n is not exceeding 10. c is also a small number, it denotes the number of maximum cardinality matchings in one bipartite graph, which is much smaller than the number of matches in $EPM(P, D, \varepsilon)$.



(a) based on (p_1, p_2) (b) based on (p_2, p_3) (c) based on (p_1, p_3)

Figure 6: Checking all pairs (p_i, p_j) of P

Lemma 1. *Checking all pairs (p_i, p_j) of P in Algorithm 1 is compulsory to find all matches of $EPM(P, D, \varepsilon)$.*

Proof. We prove this lemma with 3 examples. In Figure 6(*a*), p_1 and p_2 are diametrically symmetric (i.e., $|\overline{p_1p_2}| = |\overline{o_1o_2}| + 2\varepsilon$). To determine $\{o_1, o_2, o_3\}$ is indeed a match, we have to check (p_1, p_2). Only when p_1, p_2 run on the border of $C^{\varepsilon}_{o_1}, C^{\varepsilon}_{o_2}$, will p_3 be located inside $C^{\varepsilon}_{p_3}$. If we use other pairs, say (p_1, p_3), i.e., let p_1, p_3 run on the borders of $C^{\varepsilon}_{o_1}, C^{\varepsilon}_{o_3}$, then p_2 will never enter $C^{\varepsilon}_{p_2}$. Similarly, in Figure 6(*b*), $\{o_4, o_5, o_6\}$ can be detected as a match only when checking (p_2, p_3); in Figure 6(*c*), we can find the match $\{o_7, o_8, o_9\}$ only when checking (p_1, p_3). Therefore, to find all matches, we need to check all 3 pairs (p_i, p_j) of P. In general case, given a pattern $P(p_1, p_2, \cdots, p_n)$, for each (p_i, p_j), there might exist a set $O(o_1, \cdots, o_i, \cdots, o_j, \cdots, o_n)$ such that p_i, p_j are diametrically symmetric with regard to o_i, o_j (i.e., $|\overline{p_ip_j}| = |\overline{o_io_j}| + 2\varepsilon$ or $|\overline{p_ip_j}| = |\overline{o_io_j}| - 2\varepsilon$), then we must check (p_i, p_j) to determine O as a match. Overall, to find out all matches of P, we have to check all pairs (p_i, p_j). □

Based on Lemma 1, we cannot reduce the complexity of Algorithm 1. But we can prune object pairs (o_i, o_j) to reduce search space. Next, we present 3 pruning techniques to achieve this goal.

5 PRUNING TECHNIQUES

We present three pruning techniques to reduce the number of candidate objects for each p of P, and thus reduce the number of candidate object pairs (o_i, o_j) that need to be checked.

In Section 5.1, we propose *feasibility test* to determine the candidate objects for each $p \in P$. In Section 5.2, we propose *feasible set* and derive an upper bound of number of matches that each object belongs to, which enables us to prune unpromising objects. In Section 5.3, we propose heuristic rules for *ordering object pairs* to be checked for early termination of Algorithm 1.

5.1 Feasibility Test

Consider an object pair $(o_i, o_j) \in D_{p_i.\Phi} \times D_{p_j.\Phi}$. To ensure p_i, p_j can move on the borders of $C^{\varepsilon}_{o_i}$, $C^{\varepsilon}_{o_j}$ respectively, (o_i, o_j) should satisfy the following distance constraint,

$$|\overline{o_io_j}| \in [|\overline{p_ip_j}| - 2\varepsilon, |\overline{p_ip_j}| + 2\varepsilon]. \quad (1)$$

Definition 4. ***$F(o_i, p_j)$***. *Let o_i be a candidate object of p_i, i.e., $o_i \in D_{p_i.\Phi}$. We denote by $F(o_i, p_j)$ the set of candidate objects of p_j that can form a candidate object pair with o_i, i.e., $F(o_i, p_j) = \{o_j \in D_{p_j.\Phi} | \, |\overline{o_i o_j}| \in [|\overline{p_i p_j}| - 2\varepsilon, |\overline{p_i p_j}| + 2\varepsilon]\}$.*

Lemma 2. *Given $o \in D_{p_i.\Phi}$, if there exists $p_j \in P$ $(j \neq i)$ such that $F(o, p_j) = \emptyset$, then o cannot constitute a match and can be pruned.*

Lemma 2 holds obviously. We can apply this lemma to reduce the number of candidate objects for each $p \in P$. To efficiently compute $F(o_i, p_j)$, we can build an IR-tree [10] on database D beforehand. Algorithm 2 gives the pseudo codes to generate complete $F(o, p)$. First we find the candidate objects for each p_i (Line 2–3), and sort them in ascending order of candidate size (Line 4), which aims to decrease the number of times of computing $F(o_i, p_j)$. In Line 5–14, we visit each $o \in D_{p_i.\Phi}$ and compute $F(o, p_j)$ using IR-tree, $1 \leq i < j \leq n$. If o is pruned, we add (o, p_i) into S_1 (p_i is used to identify where o comes from); otherwise, we update $F(o, p_j)$, as $F(o, p)$ is symmetric, we update $F(o_j, p_i)$ as well. The removal of S_1 may invalidate some other objects. In Line 15–24, we repeatedly check if there are new infeasible objects. For each object $o \in S_1$, we find those affected objects o_j such that $o \in F(o_j, p_i)$, and update the $F(o_j, p_i)$. If o_j becomes infeasible, we put (o_j, p_j) into S_2. After checking all objects in S_1, if no new infeasible object is found, then we terminate; otherwise, we send S_2 to S_1 and repeatedly check S_1.

Complexity. Let $n = |P|$, $d = \max_{1 \leq i \leq n} |D_{p_i.\Phi}|$, $m = |D|$. It takes $O(nd)$ time to get all $D_{p_i.\Phi}$ and $O(n \log n)$ time for sorting. In Line 5–14, for each $o \in D_{p_1.\Phi}$, we invoke subroutine $F(o, p_j)$ $(n-1)$ times; for each $o \in D_{p_2.\Phi}$, we invoke $F(o, p_j)$ $(n-2)$ times, $\cdots$. Overall, we invoke $F(o, p_j)$

$$d(n-1) + d(n-2) + \cdots + d = O(n^2 d)$$

times. Since we invoke $F(o_i, p_j)$ on IR-tree which takes $O(aM \log m)$ time, where a is the maximum number of nodes being accessed in each layer of the IR-tree and M is the fanout. Therefore, it takes $O(n^2 d \cdot aM \log m)$ time in Line 5–14. In Line 15–24, if $o \in D_{p_i.\Phi}$, $\bigcup_{j \neq i} F(o, p_j)$ contains $O(nd)$ objects, which means o will be accessed $O(nd)$ times. And we have at most $|P| \cdot d = O(nd)$ objects. Therefore, it requires $O(nd \cdot nd) = O(n^2 d^2)$ time. Overall, Algorithm 2 takes

$$O(nd + n \log n + n^2 daM \log m + n^2 d^2) = O(n^2 daM \log m + n^2 d^2).$$

Usually, fanout M is very small. a is also very small as the maximum distance between any (p_i, p_j) is much smaller than the Euclidean space of D, resulting in that only a small fraction of nodes are visited in each layer of the IR-tree.

Table 2 gives $F(o, p)$ for objects in Figure 5. Since o_1 is the candidate object of p_1, we do not need to compute $F(o_1, p_1)$, the corresponding column is filled with 'x'.

Improvement. We first execute Algorithm 2, then execute Algorithm 1. (1) We can prune infeasible objects based on Lemma 2; (2) we can generate all candidate object pairs (o_i, o_j) to be fed into Algorithm 1. Specifically, Line 5 of Algorithm 1 is changed to: foreach pair (o_i, o_j), $o_j \in F(o_i, p_j)$, and Line 7 is changed to: $D_{p_k.\Phi} \longleftarrow F(o_i, p_k) \cap F(o_j, p_k)$.

Algorithm 2: FeasibilityTest(P, D, ε)

```
Input : Database D, spatial pattern P, tolerance ε
Output: F(o, p) for each promising object o
1  initialize a filtered set S1;
2  foreach p_i ∈ P do
3      D_{p_i.Φ} ⟵ candidate objects of p_i
4  sort {p_1, ··· , p_n} in ascending order of |D_{p_i.Φ}|;
5  for i = 1 to n − 1 do
6      foreach o ∈ D_{p_i.Φ} do
7          for j = i + 1 to n do
8              L ⟵ F(o, p_j) using IR-tree;
9              if L is empty then
10                 S1.add(o, p_i); break
11             else
12                 update F(o, p_j) using L;
13                 foreach o_j ∈ L do
14                     update F(o_j, p_i) with o
15 repeat
16     initialize a filtered set S2;
17     foreach (o, p_i) ∈ S1 do
18         J ⟵ {o_j | o ∈ F(o_j, p_i)};
19         foreach o_j ∈ J do
20             remove o from F(o_j, p_i);
21             if o_j pruned by Lemma 2 then
22                 S2.add(o_j, p_j)
23     S1 ⟵ S2;
24 until S1 is empty;
```

Table 2: Table $F(o, p)$

objects		$F(*, p_1)$	$F(*, p_2)$	$F(*, p_3)$	$F(*, p_4)$
p_1	o_1	×	$\{o_2\}$	$\{o_{31}, o_{32}\}$	$\{o_{41}, o_{42}\}$
p_2	o_2	$\{o_1\}$	×	$\{o_{31}, o_{32}\}$	$\{o_{41}, o_{42}\}$
p_3	o_{31}	$\{o_1\}$	$\{o_2\}$	×	$\{o_{41}\}$
	o_{32}	$\{o_1\}$	$\{o_2\}$	×	$\{o_{41}, o_{42}\}$
p_4	o_{41}	$\{o_1\}$	$\{o_2\}$	$\{o_{31}, o_{32}\}$	×
	o_{42}	$\{o_1\}$	$\{o_2\}$	$\{o_{32}\}$	×

5.2 Feasible Set

Let $UB(o)$ denote the upper bound of the number of matches containing o. If current matches found so far containing o has reached $UB(o)$, it means o will no longer constitute a new match, and thus can be pruned. Based on this observation, we propose feasible set and derive an $UB(o)$ for each object o. This enables us to prune unpromising objects and reduce duplicate matches in Algorithm 1.

Definition 5. ***Feasible set***. *Given pattern $P = \{p_1, \cdots, p_n\}$ and objects $O = \{o_1, \cdots, o_n\}$, if (1) $\forall$ i, $o_i.\phi \in p_i.\Phi$; and (2) $\forall$ i, j, $|\overline{p_i p_j}| \in [|\overline{o_i o_j}| - 2\varepsilon, |\overline{o_i o_j}| + 2\varepsilon]$, then we call O a feasible set of P.*

Lemma 3. *If O is a match, then O must be a feasible set. However, if O is a feasible set, O is not necessarily a match.*

Proof. A match is a feasible set by definition. But a feasible set is not necessarily a match. Figure 7 provides a counterexample. p_1, p_2 are diametrically symmetric, i.e., $|\overline{p_1p_2}| = |\overline{o_1o_2}| + 2\varepsilon$. Let C_1, C_2 be circles with radius r, centered at p_1, p_2 respectively. p_3, g, h are intersection points of C_1 and C_2, C_1 and $C^{\varepsilon}_{o_3}$, C_2 and $C^{\varepsilon}_{o_3}$ respectively. Then we have $|\overline{p_1p_3}| = |\overline{p_1g}| = |\overline{p_2h}| = |\overline{p_2p_3}| = r$. Since $|\overline{o_1o_3}| - 2\varepsilon \le |\overline{p_1g}| \le |\overline{o_1o_3}| + 2\varepsilon$, then $|\overline{o_1o_3}| - 2\varepsilon \le |\overline{p_1p_3}| \le |\overline{o_1o_3}| + 2\varepsilon$, which means pair (p_1, p_3) meets Eq. 1. Similarly, we can also verify that pair (p_2, p_3) meets Eq. 1. Hence, $\{o_1, o_2, o_3\}$ is a feasible set. However, it is not a match. Due to diametrical symmetry, we have to put p_1, p_2 onto the borders of $C^{\varepsilon}_{o_1}, C^{\varepsilon}_{o_2}$, in such situation, p_3 cannot enter $C^{\varepsilon}_{o_3}$. Hence, $\{o_1, o_2, o_3\}$ is not a match. □

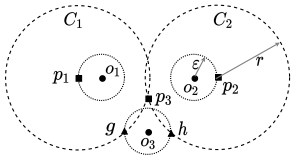


Figure 7: Counterexample

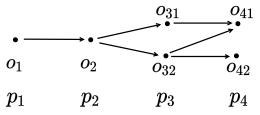


Figure 8: Feasible set

Based on Lemma 3, feasible sets contain all matches, we can set $UB(o)$ to be the number of feasible sets containing o. Fortunately, we can apply Depth-First-Search (DFS) on $F(o, q)$ to obtain all feasible sets. We take Table 2 and Figure 8 for explanation. In Figure 8, we start from o_1 ($\in D_{p_1.\Phi}$). As $o_2 \in F(o_1, p2)$, we can move to o_2. Since $o_{31} \in F(o_1, p_3) \cap F(o_2, p_3)$, we can reach o_{31} in the next step. Finally we arrive at o_{41} as $o_{41} \in F(o_1, p_4) \cap F(o_2, p_4) \cap F(o_{31}, p_4)$, and we get a feasible set $H = \{o_1, o_2, o_{31}, o_{41}\}$. Once we find a feasible set H, we increase $UB(o)$ by 1 for each $o \in H$.

Algorithm 3: FeasibleSet(T, H, k)

Input : Table $F(o, p)$, current set H, current position k
Output: Feasible sets $\mathcal{H}$, upper bound of each object

1 **if** $k = n + 1$ **then** // a feasible set is found
2 $\mathcal{H}.add(H)$;
3 **foreach** $o \in H$ **do**
4 $UB(o) \longleftarrow UB(o) + 1$;
5 **else**
6 **foreach** $o \in D_{p_k.\Phi}$ **do**
7 $H[k] = o$; // put o into current set
8 **for** $l = k + 1$ **to** n **do**
 // update candidate objects for $p_l \in P$
9 $D_{p_l.\Phi} = D_{p_l.\Phi} \cap F(o, p_l)$;
10 FeasibleSet(T, H, $k + 1$);

Algorithm 3 presents the pseudo codes of searching all feasible sets. We use H to represent a feasible set, and k denotes the current position in H. If k equals $n + 1$, which means a feasible set has been found, we put H into set $\mathcal{H}$ and update the upper bound for each object in the set H (Line 1–4). Otherwise, we check each candidate object in current point p_k (Line 6–10). When exploring $o \in D_{p_k.\Phi}$, we put o into H (Line 7), and narrow down the size of candidate objects for subsequent points of P (Line 8–9), and then continue exploring next position in set H (Line 10).

Table 3: Candidate Inverted Lists (CIL)

object pair	$D_{p_1.\Phi}$	$D_{p_2.\Phi}$	$D_{p_3.\Phi}$	$D_{p_4.\Phi}$
(o_1, o_2)	$\emptyset$	$\emptyset$	$\{o_{31}, o_{32}\}$	$\{o_{41}\}$
(o_1, o_{31})	$\emptyset$	$\{o_2\}$	$\emptyset$	$\{o_{41}\}$
(o_1, o_{32})	$\emptyset$	$\{o_2\}$	$\emptyset$	$\{o_{41}\}$
(o_1, o_{41})	$\emptyset$	$\{o_2\}$	$\{o_{31}, o_{32}\}$	$\emptyset$
(o_2, o_{31})	$\{o_1\}$	$\emptyset$	$\emptyset$	$\{o_{41}\}$
(o_2, o_{32})	$\{o_1\}$	$\emptyset$	$\emptyset$	$\{o_{41}\}$
(o_2, o_{41})	$\{o_1\}$	$\emptyset$	$\{o_{31}, o_{32}\}$	$\emptyset$
(o_{31}, o_{41})	$\{o_1\}$	$\{o_2\}$	$\emptyset$	$\emptyset$
(o_{32}, o_{41})	$\{o_1\}$	$\{o_2\}$	$\emptyset$	$\emptyset$

Complexity. Note that each row of table $F(o, p)$ is accessed at most once during DFS. Since we have at most $O(nd)$ rows in table $F(o, p)$, where each row contains at most $O(nd)$ objects. Therefore, Algorithm 3 takes $O(nd \cdot nd) = O(n^2d^2)$ time, where $n = |P|$, $d = \max_{1 \le i \le n} |D_{p_i.\Phi}|$.

Improvement. We sequentially execute Algorithm 2, Algorithm 3 and Algorithm 1. In Algorithm 3, we obtain all feasible sets $\mathcal{H}$ which can be used to save the redundant computations in searching $D_{p_k.\Phi}$ in Line 7 of Algorithm 1. Specifically, in Section 5.1, we have $D_{p_k.\Phi} = F(o_i, p_k) \cap F(o_j, p_k)$, and this requires a lot of intersection operations of sets during the whole algorithm. To reduce the overhead, we can use $\mathcal{H}$ to generate all $D_{p_k.\Phi}$'s directly through *candidate inverted lists (CIL)*. For each feasible set $H = \{o_1, o_2, \cdots, o_n\}$ in $\mathcal{H}$, for each (o_i, o_j) of H, we create a CIL as follows,

$$(o_i, o_j) : D_{p_1.\Phi} = \{o_1\}, \cdots, D_{p_i.\Phi} = \emptyset, D_{p_j.\Phi} = \emptyset, \cdots, D_{p_n.\Phi} = \{o_n\},$$

After visiting all feasible sets in $\mathcal{H}$, we combine those CILs with same object pair (o_i, o_j).

For example, suppose $\mathcal{H}$ contains $H_1 = \{o_1, o_2, o_{31}, o_{41}\}$ and $H_2 = \{o_1, o_2, o_{32}, o_{41}\}$. In H_1, we create a CIL for (o_1, o_2),

$$(o_1, o_2) : D_{p_1.\Phi} = \emptyset, D_{p_2.\Phi} = \emptyset, D_{p_3.\Phi} = \{o_{31}\}, D_{p_4.\Phi} = \{o_{41}\},$$

while in H_2, we create another CIL for (o_1, o_2),

$$(o_1, o_2) : D_{p_1.\Phi} = \emptyset, D_{p_2.\Phi} = \emptyset, D_{p_3.\Phi} = \{o_{32}\}, D_{p_4.\Phi} = \{o_{41}\},$$

after combining them, we get complete CIL for (o_1, o_2), i.e.,

$$(o_1, o_2) : D_{p_1.\Phi} = \emptyset, D_{p_2.\Phi} = \emptyset, D_{p_3.\Phi} = \{o_{31}, o_{32}\}, D_{p_4.\Phi} = \{o_{41}\}.$$

We can get complete CILs for $\mathcal{H}$ as shown in Table 3. Line 5 of Algorithm 1 can be replaced by the object pairs in CILs. And in Line 7, $D_{p_k.\Phi}$ can also be obtained directly from CILs.

Furthermore, in Algorithm 3, we also get $UB(o)$ for each object o. We derive Lemma 4 to further improve Algorithm 1. When checking (o_i, o_j) in the CILs, we first apply Lemma 4 to see if it can be pruned. And in Line 8 of Algorithm 1, we will also apply Lemma 4 to see if o_k can be pruned.

Lemma 4. *o can be pruned if the number of matches found so far which contain o, has reached $UB(o)$. Object pair (o_i, o_j) in the CILs can be pruned if at least one of $\{o_i, o_j\}$ is pruned.*

5.3 Order of Object Pairs

To prune as many objects as possible based on Lemma 4, we propose to reorder object pairs in the CILs to achieve that we can find as

many matches as possible in the first a few rounds, so that objects can reach their upper bounds earlier.

As there is no metric that can best characterise the number of matches, here we propose a few heuristic metrics to define the score of object pairs in the CILs, and sort them accordingly.

Number of remaining matches. Let $L(o)$ be the number of found matches that contain o. We define

$$S_1(o_i, o_j) = |UB(o_i) - L(o_i)| \times |UB(o_j) - L(o_j)|.$$

If $S_1(o_i, o_j)$ is large, there remains plenty of matches containing o_i or o_j to be found, and (o_i, o_j) should be explored as early as possible.

Candidate size. For (o_i, o_j), suppose o_i, o_j are candidates of p_i, p_j respectively, we define

$$S_2(o_i, o_j) = \Pi_{k \neq i,j} |D_{p_k.\Phi}|.$$

If $S_2(o_i, o_j)$ is large, it means (o_i, o_j) may constitute a lot of matches, and (o_i, o_j) should be prioritized for processing.

Freedom of rigid motion. Suppose o_i, o_j are candidate objects of p_i, p_j respectively. When p_i, p_j run on the borders of $C^{\varepsilon}_{o_i}, C^{\varepsilon}_{o_j}$, if their moves are restricted (see Figure 6), it is less likely to find matches. We define

$$S_3(o_i, o_j) = 1 - \frac{\left||\overline{o_i o_j}| - |\overline{p_i p_j}|\right|}{2\varepsilon}.$$

If $S_3(o_i, o_j)$ is large, p_i, p_j can move freely on the borders, then p_k ($k \neq i, j$) will scan larger area as p_i moves, which makes it easier for p_k to enter $C^{\varepsilon}_{o_k}$ ($o_k \in D_{p_k.\Phi}$), and thus more matches could be found. Therefore, (o_i, o_j) should be processed first.

Mixed metric. We combine the aforementioned factors to measure the significance of (o_i, o_j). We propose the following score function using linear interpolation,

$$S_4(o_i, o_j) = S_1(o_i, o_j) + \alpha \cdot S_2(o_i, o_j) + \beta \cdot S_3(o_i, o_j),$$

where α and β can be set empirically. The higher score, the better.

Improvement. We first execute Algorithm 2, Algorithm 3, then build and reorder CILs, we sort object pairs based on a score function ($S_1(o_i, o_j)$ by default). After that, we execute Algorithm 1 (Line 6–13) on each (o_i, o_j) in the CILs.

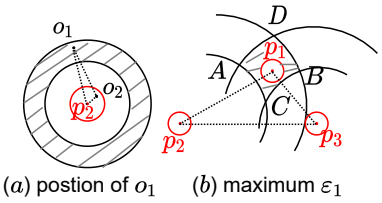
6 APPROXIMATE MATCHES

In some applications that have high demands in query latency, we can provide approximate results to trade off the efficiency and accuracy. Recall that feasible sets are intermediate results of our improved algorithms (i.e., only executing Algorithm 2 and Algorithm 3). They cover all the matches and take less running time than the complete algorithm. Thus, it is a natural idea to return the feasible sets to users as approximate results.

In the remaining of this section, we prove that we can guarantee the quality of the approximate results by parameter ε', i.e., the feasible sets in $EPM(P, D, \varepsilon)$ are true matches in $EPM(P, D, \varepsilon')$. To complete the proof, we consider the number of points in P in three cases: (1) $|P| < 3$, (2) $|P| = 3$, and (3) $|P| > 3$.

Case 1: $|P| < 3$. It is trivial to show that each feasible set is indeed a match. In this case, $\varepsilon' = \varepsilon$.

Case 2: $|P| = 3$. Given pattern $P = \{p_1, p_2, p_3\}$, tolerance ε, and feasible set $O = \{o_1, o_2, o_3\}$, consider the following 3 cases:



(a) postion of o_1 (b) maximum ε_1

Figure 9: Estimate ε_1

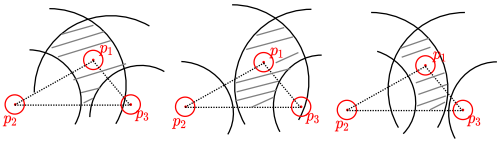


Figure 10: Other situations of ε_1

(1) o_2, o_3 reside in the circular disks of $C^{\varepsilon}_{p_2}, C^{\varepsilon}_{p_3}$ respectively. Denote the maximum distance between p_1 and o_1 as ε_1;
(2) o_1, o_3 reside in the circular disks of $C^{\varepsilon}_{p_1}, C^{\varepsilon}_{p_3}$ respectively. Denote the maximum distance between p_2 and o_2 as ε_2;
(3) o_1, o_2 reside in the circular disks of $C^{\varepsilon}_{p_1}, C^{\varepsilon}_{p_2}$ respectively. Denote the maximum distance between p_3 and o_3 as ε_3.

Now we proceed to find out ε_1 in case (1). In Figure 9(a), as o_2 is inside $C^{\varepsilon}_{p_2}$, we have $|\overline{p_2 o_2}| \leq \varepsilon$. Based on triangle inequality, we have

$$|\overline{o_1 o_2}| - |\overline{p_2 o_2}| \leq |\overline{o_1 p_2}| \leq |\overline{o_1 o_2}| + |\overline{p_2 o_2}|,$$

as $|\overline{o_1 o_2}|$ satisfies Eq. 1, then we have

$$|\overline{o_1 p_2}| \leq |\overline{p_1 p_2}| + 2\varepsilon + |\overline{p_2 o_2}| \leq |\overline{p_1 p_2}| + 3\varepsilon, \tag{2}$$

$$|\overline{o_1 p_2}| \geq |\overline{p_1 p_2}| - 2\varepsilon - |\overline{p_2 o_2}| \geq |\overline{p_1 p_2}| - 3\varepsilon. \tag{3}$$

That means o_1 must be located inside a ring-shaped area with center p_2, inner radius $r_1 = |\overline{p_1 p_2}| - 3\varepsilon$ and outer radius $r_2 = |\overline{p_1 p_2}| + 3\varepsilon$, denoted by $R^{r_1, r_2}_{p_2}$, as illustrated in Figure 9(a). Similarly, as o_3 is inside $C^{\varepsilon}_{p_3}$, o_1 must be located inside another ring-shaped area $R^{r_3, r_4}_{p_3}$, where $r_3 = |\overline{p_1 p_3}| - 3\varepsilon$, $r_4 = |\overline{p_1 p_3}| + 3\varepsilon$. Obviously, the circle $C^{\varepsilon}_{p_1}$ is contained in the intersection area of $R^{r_1, r_2}_{p_2}$ and $R^{r_3, r_4}_{p_3}$ as shown in Figure 9(b). We only consider the intersection area above the line $\overline{p_2 p_3}$, it is because that if o_1 is located below line $\overline{p_2 p_3}$, then the orientation of O is opposite to that of P, i.e., ($\angle\overrightarrow{p_2 p_1}$ - $\angle\overrightarrow{p_2 p_3}$) and ($\angle\overrightarrow{o_2 o_1}$ - $\angle\overrightarrow{o_2 o_3}$) have opposite signs, here $\angle\overrightarrow{p_2 p_1}$ is the angle of $\overrightarrow{p_2 p_3}$ in polar coordinates, we will check the orientation of O before putting it into the feasible sets.

Now that o_1 is located in the shaded area in Figure 9(b), we can deduce that $\varepsilon_1 = \max\{|\overline{Ap_1}|, |\overline{Bp_1}|, |\overline{Cp_1}|, |\overline{Dp_1}|\}$, where A, B, C, D are the intersection points of $R^{r_1, r_2}_{p_2}$ and $R^{r_3, r_4}_{p_3}$. It is easy to find out ε_1 using cosine formula. Actually there are other situations concerning the intersection area of $R^{r_1, r_2}_{p_2}$ and $R^{r_3, r_4}_{p_3}$, as illustrated in Figure 10, which depends on each $|\overline{p_i p_j}|$ of P and ε. Whichever situation it is, the ε_1 is determined by maximum distance between p_1 and the intersection points of $R^{r_1, r_2}_{p_2}$, $R^{r_3, r_4}_{p_3}$ and triangle $\triangle p_1 p_2 p_3$.

Likewise, we can find out ε_2, ε_3 in the remaining two cases. After that, we set $\varepsilon' = \min\{\varepsilon_1, \varepsilon_2, \varepsilon_3\}$. For example, if $\varepsilon' = \varepsilon_1$, then following the case (1), we can put O into the ε_1-circles of P on one-to-one basis. Equivalently, we can put P into the ε_1-circles of O on one-to-one basis, which suggests O is a match in $EPM(P, D, \varepsilon_1)$.

Table 4: Table of datasets

Datasets	Object	Total keywords	Size
Gowalla	2,705,595	630	1.20GB
4SQ1	3,410,868	429	2.05GB
4SQ2	10,150,064	519	5.68GB
Weeplaces	894,582	762	829MB

Case 3: $|P| > 3$. Given pattern $P = \{p_1, \cdots, p_n\}$ and tolerance ε, suppose $O = \{o_1, \cdots, o_n\}$ is feasible set of P, we can derive an ε' as follows:

(1) We choose (p_i, p_j) from P and factorize P into $n-2$ sub-patterns of size 3: $\{p_i, p_j, p_1\}, \{p_i, p_j, p_2\}, \cdots, \{p_i, p_j, p_n\}$.
(2) For each subpattern $\{p_i, p_j, p_k\}$, we let (o_i, o_j) reside in the circular disks of $C^{\varepsilon}_{p_i}, C^{\varepsilon}_{p_j}$ respectively, and denote the maximum distance between p_k and o_k as ε_k. After finding all ε_k's, we set $\varepsilon_{i,j} = \max_{k \neq i,j}\{\varepsilon_k\}$ and derive Lemma 5.
(3) After enumerating all (p_i, p_j) of P, we set $\varepsilon' = \min_{i,j}\{\varepsilon_{i,j}\}$ and derive Lemma 6.

Lemma 5. *Consider a pattern $P = \{p_1, \cdots, p_n\}$ and tolerance ε. If O is feasible set of $EPM(P, D, \varepsilon)$, then O must be a match in $EPM(P, D, \varepsilon_{i,j})$.*

Proof. In each subpattern $\{p_i, p_j, p_k\}$, we know that if we locate o_i, o_j inside the circular disks of $C^{\varepsilon}_{p_i}, C^{\varepsilon}_{p_j}$ respectively, the maximum distance $|\overline{o_k p_k}| \leq \varepsilon_k \leq \varepsilon_{i,j}$. That means, O can be located inside the $\varepsilon_{i,j}$-circles of P on 1-to-1 basis. Therefore, O is a match in $EPM(P, D, \varepsilon_{i,j})$. □

Lemma 6. *Consider pattern P and tolerance ε, if O is feasible set of $EPM(P, D, \varepsilon)$, then O must be a match in $EPM(P, D, \varepsilon')$.*

Proof. Suppose $\varepsilon' = \varepsilon_{i',j'}$, from Lemma 5, we know that O is a match in $EPM(P, D, \varepsilon_{i',j'})$, then O is also a match in $EPM(P, D, \varepsilon')$. □

Remark. When choosing (p_i, p_j), it takes $O(1)$ to obtain an ε_k. Hence, $O(n)$ time is required to get $\varepsilon_{i,j}$. As there are $O(n^2)$ pairs of (p_i, p_j), it takes $O(n^3)$ time in total to get final ε'.

7 EXPERIMENTS

We evaluate the effectiveness, efficiency and scalability of our algorithms [1]. All methods are implemented in Java 8 and run on a PC with Intel Xeon W-2133 CPU @ 3.60GHz and 32GB RAM.

7.1 Experimental Setup

Datasets. We use 4 real-world datasets from location-based social networks (LBSN), namely, Gowalla, Weeplaces, 4SQ1 and 4SQ2. The last two datasets are crawled from Foursquare. They contain location and category information of venues. They contain 2705595, 894582, 3410868 and 10150064 venues respectively. And they cover 630, 762, 429 and 519 categories respectively. Table 4 lists the details.
EPM Queries. We generate pattern P as follows: 1) the size $|P|$ varies from 3 to 11, the default value is 7 (in addition, we add two extreme cases $|P| = 50, 100$); 2) we randomly choose an object $o \in D$ and assign it to $p_1 \in P$. Then we collect the objects O inside the circular region with center o and radius r. Next we randomly choose objects from O and assign them to the remaining points of P. By default, $r = 20$km, we will study the effect of r in Section 7.6. For each $p \in P$, we set $p.\rho = o.\rho$, $p.\Phi = \{o.\phi\}$. To enrich $p.\Phi$, we retrieve the nearby objects of o and add their keywords into $p.\Phi$. $|p.\Phi|$ varies from 1 to 5, the default value is 3; 3) tolerance ε varies from 0.5km to 2.5km, the default value is 1.5km.

[1]Code will be released.

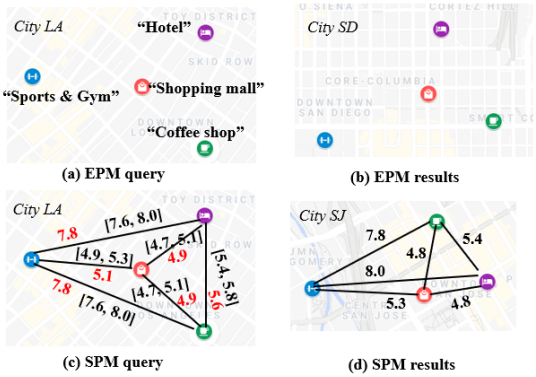


Figure 11: Case study

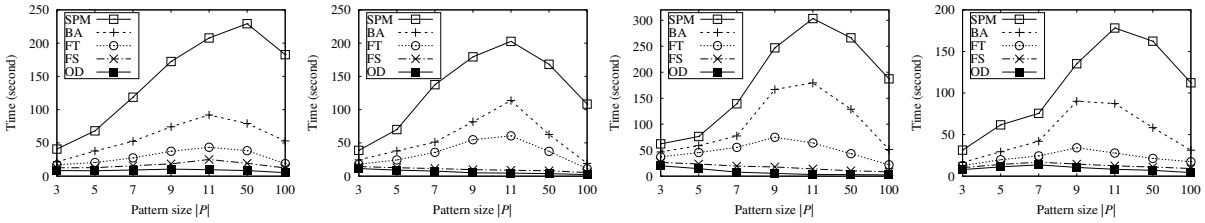
Index and methods. We use IR-tree [7] to index the objects in database and set the fanout $M = 32$. Note that our algorithms can adopt any other spatial-textual indexes. To evaluate our proposed algorithm and techniques, we compare following methods: 1) BA: Algorithm 1 in Section 4. 2) FT: Algorithm 1 combined with feasibility test. 3) FS: Algorithm 1 combined with feasibility test and feasible set . 4) OD: Algorithm 1 combined with all pruning techniques . As there exists no baseline, we adapt SPM [11] to serve as a baseline. We convert an EPM instance to an SPM instance as follows: for each pair (p_i, p_j) in EPM, we create an edge (p_i, p_j) with mutual inclusion relationship and distance interval $[|\overline{p_i p_j}| - 2\varepsilon, |\overline{p_i p_j}| + 2\varepsilon]$ in SPM. Since SPM does not consider the positional information of target objects, for each returned result, we adopt our algorithm to verify whether it is a match in EPM. We use the codes of SPM from its inventors, which also uses the IR-tree for indexing. For each test case, we randomly choose 20 queries and report the average running time.

7.2 Case Study

We give a case study on *urban planning* to show the superiority of EPM over SPM [11]. We choose 4 objects (in Los Angeles) from Gowalla as an EPM query and set $\varepsilon = 0.1$km, as shown in Figure 11(a). Figure 11(b) shows one of the matches (in San Diego) returned in EPM, which preserves the relative positions between the target objects in Figure 11(a) and can be useful for further analysis. In order to achieve the same goal using SPM, we need to describe the desired layout with a complete graph G as shown in Figure 11(c), then we feed G into an SPM solver. As a result, SPM returns a match (in San Jose) as shown in Figure 11(d), which deviates far from users' desired layout and lower down users' satisfaction. It demonstrates that SPM is not suitable for applications where users want to preserve the relative locations of target objects.

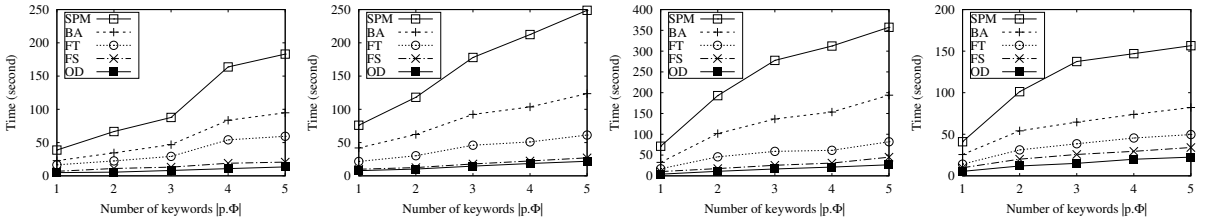
7.3 Efficiency

In this section, we study the effect of pattern size, the number of keywords and the tolerance to evaluate the efficiency of our proposed methods on 4 datasets.



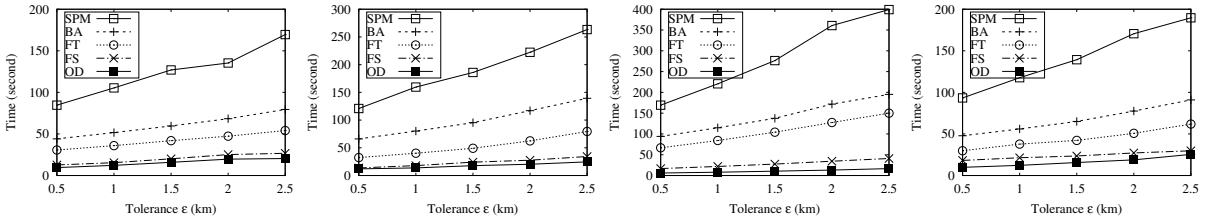
(a) Gowalla (b) 4SQ1 (c) 4SQ2 (d) Weeplaces

Figure 12: Effect of pattern size



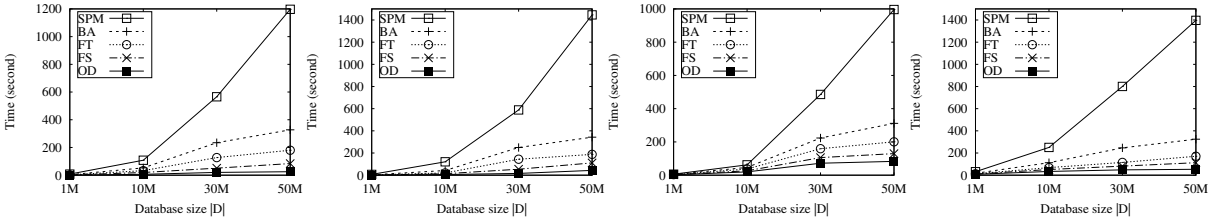
(a) Gowalla (b) 4SQ1 (c) 4SQ2 (d) Weeplaces

Figure 13: Effect of keywords



(a) Gowalla (b) 4SQ1 (c) 4SQ2 (d) Weeplaces

Figure 14: Effect of tolerance



(a) Gowalla (b) 4SQ1 (c) 4SQ2 (d) Weeplaces

Figure 15: Scalability

Effect of pattern size. Figure 12 gives the running time w.r.t. the pattern size $|P|$. We observe that (1) equipped with all pruning techniques, OD is very efficient and can find all matches in less than 20 seconds even on the dataset with over 10 million objects. (2) The running time of FS and OD grows very slowly or even decreases when $|P|$ increases. On the one hand, when $|P|$ increases, we need to enumerate more pairs (p_i, p_j), which increase the running time. On the other hand, if there are more points in the pattern, the distance constraints between each other are stronger, which helps to prune more objects and decrease the running time. The total running time depends on which of the two factors is dominant. The decreasing trend of FS and OD with $|P|$ indicates that our proposed techniques can efficiently prune a large number of unpromising objects and avoid unnecessary searches. In contrast, BA and FT are dominated by the first factor when $|P|$ is small or moderate (3 to 11), and dominated by the second factor when $|P|$ is large (over 50). (3) When $|P|$ is small or moderate, the gap between the running time of BA, FT, FS and OD increases with $|P|$. In Figure 12(a), when $n = 3$, FT, FS and OD improve BA by 20.1%, 41.3% and 60.6%. When $n = 11$, they improve BA by 43.0%, 72.7% and 89.4%. Similar observations can be made in Figure 12(b)-(d). The reason for this phenomenon is that, when there are more points in the pattern, there are more distance constraints between the points and leaves us more space for pruning. This explains why FT, FS and OD outperform BA more notably when $|P|$ is larger. Interestingly, when $|P|$ is sufficiently large (over 50), the gap becomes small. This is because in such case, there are very few matches in the database, strong distance constraints will prune most of the objects and make all methods

have comparable performance. (4) OD achieves over 10x speedup compared with SPM. This is because SPM treats each $(p_i, p_j) \in P$ independently and generates matches by incrementally joining the candidate objects pairs for each (p_i, p_j), which is much slower than our methods.

Effect of keywords. Figure 13 gives the running time w.r.t. the number of keywords on each point in the pattern. When there are more keywords on each point $p \in P$, the size of candidate objects $|D_{p.\Phi}|$ grows, and we need to explore more object pairs (o_i, o_j) in Algorithm 1, which increases the running time. However, we observe that the running time of FT, FS and OD grows more slowly than that of BA, which indicates the effectiveness and efficiency of our pruning techniques. FT, FS and OD provide much better performance than BA when the number of keywords is larger. For example, in Figure 13(b), when $|p.\Phi| = 1$, FT, FS and OD improve BA by 48.5%, 76.4% and 81.5%. When $|p.\Phi| = 5$, they improve BA by 50.4%, 78.1% and 82.3%. Similar observations can also be made in other figures in Figure 13. This is because, when the number of keywords is larger, there are more candidate objects for each point in the pattern. However, in fact, most candidate objects will not constitute the true matches and thus can be pruned based on Lemma 2, which indicates that FT, FS and OD have more significant improvements over BA when $|p.\Phi|$ is larger. Furthermore, we observe that OD is over 10x faster than SPM in all situations.

Effect of tolerance. Figure 14 gives the running time w.r.t. the tolerance. When the tolerance is larger, the distance constraints in the pattern are looser and more candidate objects survive from the pruning, which implies more time is required to find all the matches. The running time of all methods grows nearly linearly w.r.t. the tolerance. Our proposed techniques scale well w.r.t. the tolerance and consistently outperform SPM (over 10x speedup).

7.4 Scalability

We study scalability w.r.t. the database sizes. We apply random sampling to generate smaller dataset and random duplicating to generate larger dataset, thus generating datasets from 1 million to 50 million. Figure 15 shows the running time under different database sizes. We observe that the running time of BA grows faster than FT, FS and OD. This is because the complexity of BA is $O(n^4 d^4 (c+n^{0.5}))$, when $|D|$ increases t times, then d ($d << |D|$) will increase nearly t times as well, and thus the total complexity will increase t^4 times. Compared with BA, the running time of FS and OD grows more slowly, which indicates our pruning techniques can effectively prune a huge number of unpromising object pairs. For example, in Figure 15b, when $|D| = 50$M, FT, FS and OD improve BA by 45.3%, 67.8% and 87.4%. From Figure 15, we observe that our methods have good scalability w.r.t. the database size. As for SPM, its running time increases exponentially as $|D|$ increases. Our methods are more than an order of magnitude faster than SPM.

7.5 Approximate Matches

To further reduce query latency, we can return feasible sets combined with bounded error ε' as approximate matches to users. We report the efficiency and quality of approximate matches on Gowalla.

Efficiency. To evaluate the efficiency, we compare 3 methods. (1) OD. It returns all exact matches. (2) AP. After generating $F(o, p)$ using Algorithm 2, we find all feasible sets via DFS on $F(o, p)$ using Algorithm 3. Then we deduce an ε' based on Lemma 6. Finally, AP returns all feasible sets and ε'. (3) EN. It is a baseline for finding all feasible sets without the help of $F(o, p)$: starting from a candidate object (say o_1) of p_1, we retrieve the candidate objects of p_2 (say o_2) which satisfy the distance constraint (i.e., Eq. 1) with o_1. Next we find candidate objects of p_3 (say o_3) which satisfy Eq. 1 with o_1, o_2 simultaneously. We repeat this procedure until all feasible sets are found. Table 5 lists the running time (ms) of 3 methods. Comparing AP and EN, we can see that AP is 4x–6x faster than EN, which demonstrates the efficiency of our feasibility test. And comparing AP and OD, we find that over 90% running time of OD is used to verify the feasible sets to obtain the exact matches. *We can reduce the query latency by two orders of magnitude if we report approximate matches.*

Table 5: Performance of approximate matches

	Running time (ms)			Quality			
	OD	AP	EN	Prec.	ε(km)	ε'(km)	ε_t(km)
Gowalla	9,638	32	120	37.2%	1.5	2.17	1.93
4SQ1	17,558	128	795	31.3%	1.5	2.76	2.28
4SQ2	14,822	338	1,437	34.5%	1.5	2.98	2.15
Weeplaces	14,315	55	264	36.1%	1.5	2.73	1.87

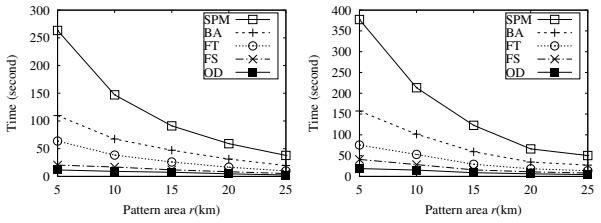
Quality. To measure the quality, we report following metrics in method AP. (1) Precision rate, i.e., #(true matches)/#(feasible sets). (2) ε, ε' and ε_t. ε is the EPM tolerance. ε' is the theoretical tolerance derived from Lemma 6, representing that the returned feasible sets are matches in $EPM(P, D, \varepsilon')$. ε_t is the minimum tolerance that makes all returned feasible sets true matches in $EPM(P, D, \varepsilon_t)$, and is derived as follows: for each false match H in the returned feasible sets of AP, we find out the minimum ε_t such that H is a match in $EPM(P, D, \varepsilon_t)$ [14]. After considering all false matches in the returned feasible sets, we report the largest ε_t. Table 5 shows these metrics of AP. We can see that about 30% of feasible sets are true matches. As ε_t is very close to ε ($\varepsilon_t \approx 1.5\varepsilon$), it implies that the returned feasible sets are approximate matches of high quality. Furthermore, ε' is also close to ε ($\varepsilon' \approx 2\varepsilon$), which demonstrates that we can find approximate matches with good quality efficiently.

7.6 Miscellaneous Experiments

Effect of score functions. We study the effect of different score functions in method OD. As $S_1(o_i, o_j)$, $S_2(o_i, o_j)$, $S_3(o_i, o_j)$ take values in different scales of magnitude, we vary α from 0.1 to 1 and β from 100 to 1000 in $S_4(o_i, o_j)$. Table 6 lists the running time (ms) under 4 different score functions. For S_4, we present the optimum α and β that give the best performance. Though there is no significant difference among these score functions, we can see that S_1 consistently performs the best, followed by S_2 and S_4, and finally S_3. It suggests that it is more beneficial to sort the pair (o_i, o_j) based on the number of remaining matches (i.e., S_1). Freedom of rigid motion (i.e., S_3) is less effective because it does not consider the number of remaining matches to be found. Mixed metric (i.e., S_4) needs to maintain more additional information of (o_i, o_j) compared with others, which makes it slower than S_1. By default we apply score function S_1 in method OD.

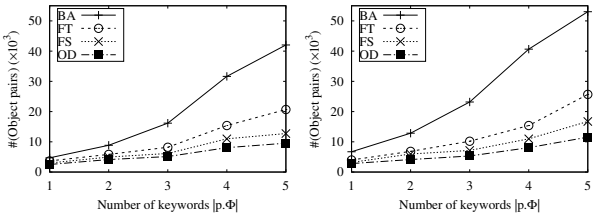
Table 6: Running time (ms) of score functions

	S_1	S_2	S_3	S_4
Gowalla	9,136	9,596	9,960	9,606 ($\alpha = 0.3, \beta = 1000$)
4SQ1	17,346	17,602	17,724	17,568 ($\alpha = 0.1, \beta = 1000$)
4SQ2	6,762	6,810	7,064	6,942 ($\alpha = 0.3, \beta = 800$)
Weeplaces	14,316	14,716	14,756	14,464 ($\alpha = 0.1, \beta = 200$)



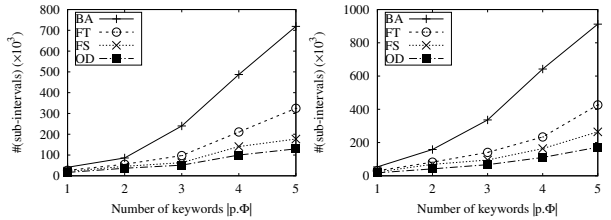
(a) Gowalla **(b) 4SQ1**

Figure 16: Effect of pattern area



(a) Gowalla **(b) 4SQ1**

Figure 17: #(object pairs) to be checked



(a) Gowalla **(b) 4SQ1**

Figure 18: #(sub-intervals) to be checked

Effect of pattern area. To study this factor, we generate pattern P as follows. First we find an object o from D and assign it to $p_1 \in P$. Then we collect the objects O in the circular region with center o and radius r. Next we randomly choose objects from O and assign them to the remaining points of P. We vary r from 5km to 25km and the effect of it is shown in Figure 16. As r increases, the running time decreases. It is because, when r is smaller, the points in P are more semantically and spatially connected. They are more like a real-life community and thus have more potential matches in the database. Therefore, it takes more time for query processing when r is smaller. Furthermore, we can also observe that in all situations, FT, FS and OD improve BA over 50%, 70% and 80% respectively. Our methods consistently outperform SPM. OD achieves over 10x speedup compared with SPM.

Table 7: Breakdown of SPM (t_f, t_v) (seconds)

size $\|P\|$	3	5	7	9	11
Gowalla (t_f)	16.3	30.7	44.8	69.0	95.2
(t_v)	24.5	37.5	73.2	105.9	111.8
4SQ1 (t_f)	19.8	32.9	59.1	73.6	79.0
(t_v)	19.0	37.1	78.4	105.9	123.6

Efficiency of pruning techniques. Technically, our three pruning techniques aim to reduce the object pairs (o_i, o_j) to be checked in the Algorithm 1 (Line 5). Besides the running time, we can also record the number of object pairs to be checked to quantify the improvements of three pruning techniques. We compare BA, FT, FS and OD under different number of keywords, and the results are shown in Figure 17. We can see that the our pruning techniques can effectively reduce the number of object pairs to be checked. And the pruning power is stronger when there are more keywords on each point $p \in P$. For example, in Figure 17(a), when $|p.\Phi| = 1$, FT, FS and OD improve BA by 35.4%, 47.6% and 56.1%. When $|p.\Phi| = 5$, they improve BA by 50.8%, 69.7% and 77.3%. This observation coincides with the conclusion in Figure 13. To further illustrate the efficiency of each algorithm in a finer granularity, we also report the number of sub-intervals being checked in each algorithm (Line 10–13 in Algorithm 1). The results are shown in Figure 18. We can draw similar conclusions as we do from Figure 17.

Breakdown of SPM. In Figure 12, the running time SPM contains (i) t_f: finding SPM matches, and (ii) t_v: verifying as EPM matches. We report the separate running time in Table 7. We observe that almost 40% running time is used for finding SPM matches. It implies that (1) SPM is inefficient in finding candidate sets which preserve the pairwise distances of a given pattern; and (2) verification procedure is indeed time-consuming, which coincides with observations in the comparison of methods OD and AP in Section 7.5.

8 CONCLUSIONS AND FUTURE WORK

In this paper, we propose EPM query. This is the first work to adopt matching under tolerance region in object set retrieval. To efficiently solve EPM query, we propose an efficient algorithm and 3 advanced pruning techniques. Furthermore, we provide theoretical analysis and approximation guarantee for the approximate results returned by our algorithm. Experimental evaluations demonstrate the effectiveness and efficiency of our proposed methods.

This work opens a few directions for future work. First, we plan to investigate the possibility of integration with other example-based queries in relational databases, where data has both spatial attributes and other rich types of attributes. Second, we plan to support more complex transformations (e.g., reflection, scaling) in the EPM query. Third, it is an interesting open problem to define sub-pattern matching problem which returns partially matching results. Finally, it is interesting to develop approximate solutions which allow the users to specify the error bounds.

ACKNOWLEDGMENTS

This research is supported in part by MOE Tier-2 grant MOE2019-T2-2-181 and a grant awarded by AI.SG with award No. AISG2-TC-2021-001.

REFERENCES

[1] Hashem R Al-Masaeid and Ghassan Suleiman. 2004. Relationships between urban planning variables and traffic crashes in Damascus. *Road & Transport Research* 13, 4 (2004), 63–73.

[2] H. Alt, K. Mehlhorn, H. Wagener, and E. Welzl. 1988. Congruence, similarity, and symmetries of geometric objects. *Discrete & Computational Geometry* 3, 3 (1988), 237–256.

[3] E. M Arkin, K. Kedem, J. SB Mitchell, J. Sprinzak, and M. Werman. 1992. Matching points into pairwise-disjoint noise regions: combinatorial bounds and algorithms. *ORSA Journal on Computing* 4, 4 (1992), 375–386.

[4] X. Cao, G. Cong, and C. S. Jensen. 2010. Retrieving Top-k Prestige-Based Relevant Spatial Web Objects. *Proc. VLDB Endow.* 3, 1–2 (2010), 373–384.

[5] X. Cao, G. Cong, C. S. Jensen, and B. C. Ooi. 2011. Collective Spatial Keyword Querying. In *SIGMOD*. 373–384.

[6] L. Chen, G. Cong, and X. Cao. 2013. An efficient query indexing mechanism for filtering geo-textual data. In *SIGMOD*. 749–760.

[7] L. Chen, G. Cong, C. S Jensen, and D. Wu. 2013. Spatial keyword query processing: an experimental evaluation. *Proc. VLDB Endow.* 6, 3 (2013), 217–228.

[8] Y. Chen, Z. Chen, G. Cong, A. R Mahmood, and W. G Aref. 2020. SSTD: a distributed system on streaming spatio-textual data. *Proc. VLDB Endow.* 13, 12 (2020), 2284–2296.

[9] Z. Chen, G. Cong, Z. Zhang, T. Z. J. Fuz, and L. Chen. 2017. Distributed Publish/Subscribe Query Processing on the Spatio-Textual Data Stream. In *ICDE*. 1095–1106.

[10] G. Cong, C. S Jensen, and D. Wu. 2009. Efficient retrieval of the top-k most relevant spatial web objects. *Proc. VLDB Endow.* 2, 1 (2009), 337–348.

[11] Y. Fang, R. Cheng, G. Cong, N. Mamoulis, and Y. Li. 2018. On spatial pattern matching. In *ICDE*. IEEE, 293–304.

[12] K. Fukuda and T. Matsui. 1994. Finding all the perfect matchings in bipartite graphs. *Applied Mathematics Letters* 7, 1 (1994), 15–18.

[13] T. Guo, X. Cao, and G. Cong. 2015. Efficient Algorithms for Answering the M-Closest Keywords Query. In *SIGMOD*. 405–418.

[14] P. J Heffernan and S. Schirra. 1994. Approximate decision algorithms for point set congruence. *Computational Geometry* 4, 3 (1994), 137–156.

[15] A Helmut and JG Leonidas. 1996. Discrete geometric shapes: matching, interpolation, and approximation: a survey. *Institute of Computer Science* (1996).

[16] G. Li, Y. Wang, T. Wang, and J. Feng. 2013. Location-aware publish/subscribe. In *KDD*. 802–810.

[17] J. Liu, K. Deng, H. Sun, Y. Ge, X. Zhou, and C. S. Jensen. 2017. Clue-Based Spatio-Textual Query. *Proc. VLDB Endow.* 10, 5 (2017), 529–540.

[18] Yiding Liu, Tuan-Anh Nguyen Pham, Gao Cong, and Quan Yuan. 2017. An Experimental Evaluation of Point-of-Interest Recommendation in Location-Based Social Networks. *Proc. VLDB Endow.* 10, 10 (2017), 1010—-1021.

[19] Y. Liu, W. Wei, A. Sun, and C. Miao. 2014. Exploiting Geographical Neighborhood Characteristics for Location Recommendation. In *CIKM*. 739–748.

[20] Yiding Liu, Kaiqi Zhao, and Gao Cong. 2018. Efficient Similar Region Search with Deep Metric Learning. KDD, 1850–1859.

[21] S. Luo, J. Hu, R. Cheng, J. Yan, and B. Kao. 2017. SEQ: Example-based Query for Spatial Objects. In *CIKM*. 2179–2182.

[22] A. R Mahmood, A. M Aly, and W. G Aref. 2018. FAST: frequency-aware indexing for spatio-textual data streams. In *ICDE*. IEEE, 305–316.

[23] A. R Mahmood, A. Daghistani, A. M Aly, M. Tang, S. Basalamah, S. Prabhakar, and W. G Aref. 2018. Adaptive processing of spatial-keyword data over a distributed streaming cluster. In *SIGSPATIAL*. 219–228.

[24] Tuan-Anh Nguyen Pham, Xutao Li, and Gao Cong. 2017. A General Model for Out-of-Town Region Recommendation. WWW, 401—-410.

[25] X. Wang, Y. Zhang, W. Zhang, X. Lin, and W. Wang. 2015. Ap-tree: Efficiently support continuous spatial-keyword queries over stream. In *ICDE*. 1107–1118.

[26] D. Yang, B. Qu, J. Yang, and P. Cudre-Mauroux. 2019. Revisiting User Mobility and Social Relationships in LBSNs: A Hypergraph Embedding Approach. In *WWW*. 2147–2157.

[27] C. Zhang, Y. Zhang, W. Zhang, and X. Lin. 2016. Inverted linear quadtree: Efficient top k spatial keyword search. *TKDE* 28, 7 (2016), 1706–1721.

[28] D. Zhang, Y. M. Chee, A. Mondal, A. K. H. Tung, and M. Kitsuregawa. 2009. Keyword Search in Spatial Databases: Towards Searching by Document. In *ICDE*. 688–699.

[29] D. Zhang, B. C. Ooi, and A. K. H. Tung. 2010. Locating mapped resources in Web 2.0. In *ICDE*. 521–532.