Spatial Pattern Matching: A New Direction for Finding Spatial Objects

Yun Li[1], Yixiang Fang[2], Reynold Cheng[3], Wenjie Zhang[2]
[1]Department of Computer Science and Technology, Nanjing University, China
[2]School of Computer Science and Engineering, The University of New South Wales, Australia
[3]Department of Computer Science, University of Hong Kong, Hong Kong
Email: liycser@gmail.com, yixiang.fang@unsw.edu.au,
ckcheng@cs.hku.hk, wenjie.zhang@unsw.edu.au

Abstract

In this paper, we study the spatial pattern matching *(SPM) query. Given a set D of spatial objects (e.g., houses and shops), each with a textual description, we aim at finding all combinations of objects from D that match a user-defined spatial pattern P. A pattern P is a graph whose vertices represent spatial objects, and edges denote distance relationships between them. The SPM query returns the instances that satisfy P. An example of P can be "a house within 10-minute walk from a school, which is at least $2km$ away from a hospital". The SPM query can benefit users such as house buyers, urban planners, and archaeologists. We first formally formulate the SPM problem, and then propose efficient query algorithms. We also develop an online system, called SpaceKey, which is based on the SPM query, to support some real applications such as property searching. Finally, we point out a list of possible research directions for future work.*

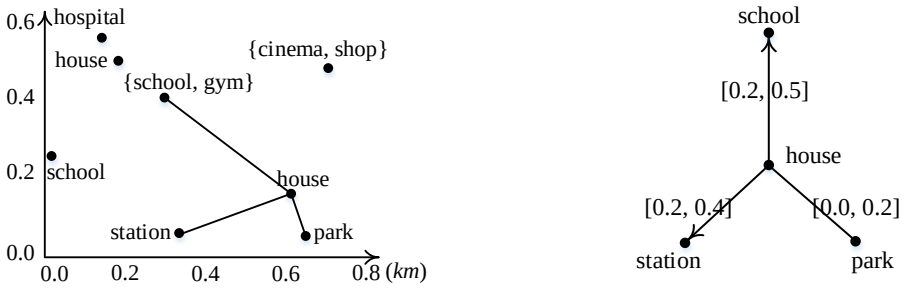
1 Introduction

With the rapid development of location-based services, spatial databases, which contain objects carrying locations and other information [2, 8, 7, 11], are prevalent in emerging platforms (e.g., Google Maps[1] and Foursquare[2]). In this paper, we study *spatial pattern matching*, or SPM in short. In this problem, we wish to find out the instances in a spatial database D that satisfy a given spatial pattern P. Figure 1(a) illustrates D, which contains a variety of spatial objects (e.g., park, station, and house). An example of P, which specifies keywords of objects and their distance relationships (e.g., a house is less than $0.2km$ away from a park; the distance between a bus station and a house is between $0.2km$ and $0.4km$), is shown in Figure 1(b). An instance that satisfies P (called a match), containing objects connected in solid lines, is shown in Figure 1(a).

The SPM can be useful for accommodation-search applications (e.g., AirBNB[3], and trivago[4]), where users can look for apartments or hotels based on their preferences. By using SPM, a user can indicate more complex

Yixiang Fang is the corresponding author.

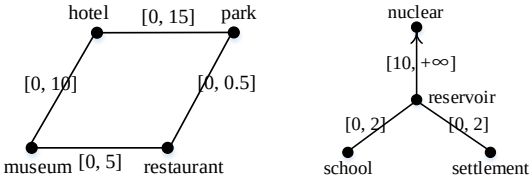
[1]https://maps.google.com.hk
[2]https://foursquare.com
[3]http://www.airbnb.com.hk
[4]http://www.trivago.hk



(a) Spatial object set D (b) Spatial pattern P

Figure 1: Illustrating the SPM query [6].

constraints that are currently not supported in these platforms. For example, a user may want to look for a house which is not too far from a shop, and the shop is close to a canteen. She may also want the house to be at least some distance from a subway station (to avoid noise and crowd). These requirements, which have not been provided by existing applications, can be captured succinctly in terms of a *spatial pattern* (e.g., Figure 1(b)). As another example, a travel agent may wish to enumerate all the possible itineraries, and recommend them to customers [2]. This kind of requests can be formulated as a spatial pattern (e.g., Figure 2(a)), with their spatial relationship indicated (e.g., a museum should be 10 minutes of driving from the restaurant). The SPM can also be used in urban planning. As discussed in [13], urban analysts often need to survey the layout of a city. They need to identify facilities with specific distance relationships (e.g., find out reservoirs that are $10km$ or more from nuclear power stations), in order to gain some insights on the the design of city infrastructures (Figure 2(b)).



(a) Trip planning (b) Urban planning

Figure 2: Example spatial patterns.

To our best knowledge, there does not exist any solution that solves SPM effectively and efficiently. A class of recent works related to SPM query is spatial-keyword query (SKQ) [2, 4, 19, 14, 1, 5, 3]. Typically, these solutions require users to provide a number of query keywords, and return spatial objects that are geographically near to each other; the objects should also be relevant to the keywords. However, this may fail to capture the user's intention, because they do not allow users to specify explicit distance constraints between objects. Another topic related to SPM is graph pattern matching (or GPM) [12], which aims to find subgraphs matching a given pattern in a graph. However, using GPM solutions to solve the SPM problem is not straightforward. Particularly, we have to transform the set of spatial objects involved (e.g., Figure 1(a)) into a graph, and then run a GPM solution on it.

Our contributions. We first present a formal definition of spatial pattern [6, 10]. We propose several distance constraints for a spatial pattern, which specify (1) minimum and maximum distances between two object types; and (2) *exclusion* and *inclusion*. Figure 1(b) illustrates the exclusion relationship ($\rightarrow$), which expresses that (1) a bus station should be at least $0.2km$ from a house but not more than $0.4km$; and (2) no station should be in the vicinity of $0.2km$ of a house. Based on this spatial pattern, we define the SPM problem,

and we prove that it is NP-hard. To tackle this issue, we propose two efficient and effective algorithms based on the IR-tree index [4, 16]. The first solution, which we called the multi-pair-join (or MPJ), is based on performing multi-way join operations [17, 20, 18] on each edge of the spatial pattern. We also develop a sampling-based estimation method to guide the execution order of the joins. Our second solution, namely the multi-star-join (or MSJ), also follows the multi-way join framework, but includes more sophisticated techniques, such as bounded pattern and optimized join order.

In addition, we have implemented our solution in a system, called SpaceKey [9]. SpaceKey is a system for retrieving and visualizing spatial objects returned by SPM queries and some other well-known SKQs. Thus, SpaceKey allows users to perform comparison analysis among queries.

We formulate the SPM problem in Section 2. Section 3 presents our SPM solutions. We introduce SpaceKey in Section 4. We discuss future research directions and conclude in Section 5.

2 Problem Definition

Let D be a database of spatial objects (or *objects* for brevity). Each object $o_i \in D$ ($1 \leq i \leq |D|$) has 2D coordinates (x_i, y_i), and is associated with a set of keywords, denoted by $doc(o_i)$. We say that o_i *matches* with a keyword w, if $w \in doc(o_i)$. Given two objects o_i and o_j, we use $|o_i, o_j|$ to denote their Euclidean distance. We denote a spatial circle with center o and radius r by $O(o, r)$. Next, we formally define spatial patterns.

Definition 1 (spatial pattern): A spatial pattern P is a graph $P(V, E)$ of n vertices $\{v_1, v_2, \cdots, v_n\}$ and m edges, such that the following constraints hold:

- Each vertex $v_i \in V$ has a keyword w_i;
- Each edge $(v_i, v_j) \in E$ has a distance interval $[l_{i,j}, u_{i,j}]$, where $l_{i,j}$ ($u_{i,j}$) is the lower (respectively upper) bound of distances between two matching objects in D;
- Each edge $(v_i, v_j) \in E$ is associated with one of the signs: (1) $v_i \rightarrow v_j$; (2) $v_i \leftarrow v_j$; (3) $v_i \leftrightarrow v_j$; and (4) v_i–v_j.

Let (v_i, v_j) be an edge in E, with distance interval $[l_{i,j}, u_{i,j}]$. Also, let o_k and o_l be the two objects returned in a match of E, where $w_i \in doc(o_k)$ and $w_j \in doc(o_l)$. We now discuss the four possible *signs* of an edge in Definition 1:

- $v_i \rightarrow v_j$ [v_i excludes v_j]: No object with keyword w_j in D should have a distance less than $l_{i,j}$ from o_k.
- $v_i \leftarrow v_j$ [v_j excludes v_i]: No object with keyword w_i in D should have a distance less than $l_{i,j}$ from o_l.
- $v_i \leftrightarrow v_j$ [mutual exclusion]: No object with keyword w_j in D should have a distance less than $l_{i,j}$ from o_k, and the distance of any object with keyword w_i in D should be at least $l_{i,j}$ away from o_l.
- v_i–v_j [mutual inclusion]: The occurrence of any object (other than o_k and o_l) with keywords w_i and w_j in D with distance shorter than $l_{i,j}$ is allowed.

For example, consider the edge *house*$\rightarrow$*school* with distance interval $[0.2, 0.5]$ (*km*) in the pattern of Fig. 1(b). Intuitively, the user wishes to retrieve two objects (say, o_s and o_t) such that: (1) o_s and o_t have keywords *house* and *school* respectively; (2) the distance of o_s from o_t is between $0.2km$ and $0.5km$; and (3) there does not exist any object with keyword *school*, which is less than $0.2km$ from o_s. The arrow in *house*$\rightarrow$*school* is expressed as *house excludes school*, and captures the user's intention of not getting any match for which the *house* has a *school* object less than $0.2km$ from it. We define an *e-match* of an edge (v_i, v_j), as follows:

Definition 2 (e-match): Two objects o_k and o_l constitute an e-match of (v_i, v_j), if $doc(o_k)$ and $doc(o_l)$ include w_i and w_j, respectively, and the objects satisfy the distance constraints of (v_i, v_j).

Definition 3 (match): Given a spatial pattern $P(V, E)$ and a set S of objects, S is a match of P if there exists a surjection $\phi : V \rightarrow S$, such that for all $v, v' \in V$, if $(v, v') \in E$, then the object pair $(\phi(v), \phi(v'))$ forms an e-match of (v, v').

Problem definition. Given a database D of spatial objects and a spatial pattern P, spatial pattern matching (SPM) aims to find all the matches of P in D.

For example, in Fig. 1(a), the four objects connected in solid lines are a match of the pattern in Fig. 1(b) and they form an answer to this SPM query.

3 Solutions

Although SPM can be used for a wide range of applications, it is computational intractable since it is NP-hard as proved in [10]. To solve the SPM problem, we develop two efficient algorithms (i.e., `MPJ` and `MSJ`) based on the IR-tree index [4, 16]. Due to the space limitation, we focus on introducing the key ideas of the second solution. In the following, we first introduce the three key techniques in `MSJ` and then discuss the overall algorithm.

3.1 The Bounded Pattern

Given a spatial pattern P, we define its ***bounded pattern*** $\widehat{P}$ as a clique graph satisfying properties:

- There are n vertices $\{\widehat{v_1}, \widehat{v_2}, \cdots, \widehat{v_n}\}$;
- Each vertex is linked with each other vertex;
- $\forall(\widehat{v_i}, \widehat{v_j})$ of $\widehat{P}$, its distance interval $[\widehat{l_{i,j}}, \widehat{u_{i,j}}]$ is initialized as $[l_{i,j}, u_{i,j}]$ if $(v_i, v_j) \in P$, or $[0, +\infty]$ if $(v_i, v_j) \notin P$.
- The distance intervals on all the edges are computed by dynamic programming using Eqs (1) and (2).

$$\widehat{u_{i,j}} = \min_{1 \le k \le n} \{\widehat{u_{i,j}}, \widehat{u_{i,k}} + \widehat{u_{k,j}}\}. \tag{1}$$

$$\widehat{l_{i,j}} = \max_{1 \le k \le n} \begin{cases} 0 & [\widehat{l_{i,k}}, \widehat{u_{i,k}}] \cap [\widehat{l_{k,j}}, \widehat{u_{k,j}}] \neq \emptyset \\ \widehat{l_{k,j}} - \widehat{u_{i,k}} & \widehat{u_{i,k}} < \widehat{l_{k,j}} \\ \widehat{l_{i,k}} - \widehat{u_{k,j}} & \widehat{l_{i,k}} > \widehat{u_{k,j}} \end{cases}. \tag{2}$$

We can observe that, when computing the lower and upper bound distances between any two vertices using Eqs (1) and (2), we have considered all the paths between them, and so they are *globally* tight. This implies, we can use them to refine P, which may reduce the query computational cost.

Let e=v_i–v_j be an edge with mutual inclusion. We have the following refining criteria:

❶ If $[l_{i,j}, u_{i,j}] \cap [\widehat{l_{i,j}}, \widehat{u_{i,j}}] = \emptyset$, then P is a wrong pattern, since no pair of objects can satisfy the distance constraint.

❷ If $[\widehat{l_{i,j}}, \widehat{u_{i,j}}] \subset [l_{i,j}, u_{i,j}]$, we delete (v_i, v_j), as any set of objects matched with $P \backslash e$ is also a match of P.

❸ If neither criterion ❶ nor criterion ❷ can be applied, then we refine $[l_{i,j}, u_{i,j}]$ as $[l_{i,j}, u_{i,j}] \cap [\widehat{l_{i,j}}, \widehat{u_{i,j}}]$, since any set of objects matched with P is also a match of $\widehat{P}$.

We illustrate above refining criteria by Example 1 as follows.

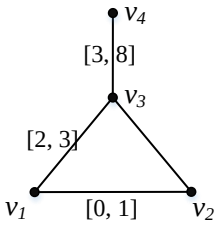


Figure 3: Illustrating pattern refining.

Example 1: Consider a pattern in Figure 3, and the edge e= (v_2, v_3) has three different cases. Note that $[\widehat{l_{2,3}},\widehat{u_{2,3}}]$ is always a subinterval of [1, 4]. If [$l_{2,3}$, $u_{2,3}$]=[5, 6], then it is a wrong pattern by criterion ❶; if [$l_{2,3}$, $u_{2,3}$]=[0, 5], then we delete e by criterion ❷; and if [$l_{2,3}$, $u_{2,3}$] =[2, 5], we update it as [2, 4] by criterion ❸.

If the relationship between v_i and v_j is not mutual inclusion, we replace criteria ❷ and ❸ by criterion ❹:
❹ If $\widehat{u_{i,j}}<u_{i,j}$, we simply refine $[l_{i,j}, u_{i,j}]$ as $[l_{i,j}, \widehat{u_{i,j}}]$.

3.2 The PJ Algorithm

To compute the e-matches of an edge, we assume that there is an IR-tree built for D. Given an IR-tree and an edge with keywords w_i and w_j, PJ finds all the matched pairs of IR-tree nodes level by level in a top-down manner. Specifically, for the root level, the root node and itself form a matched pair (assume that the IR-tree has both keywords w_i and w_j). Then, we find the child node pairs that match and follow them, repeating the same process until all the matched objects at the leaf level are found.

We now explain when a pair of nodes match. Let p and q be two non-leaf nodes, whose inverted files contain w_i and w_j respectively, at the same level of the IR-tree. Intuitively, if p and q's MBRs are far from or too close to each other, we cannot find any pair of objects under them that form an e-match.Then q does not match with p if the minimum distance of their MBRs is not larger than $u_{i,j}$. Similarly, the maximum distance of their MBRs must be less than $l_{i,j}$.

To prune unmatched node pairs, we exploit a key property of the IR-tree. That is, for each node in the IR-tree, its MBR must be contained by the MBR of its parent node. As a result, after finding all matched node pairs at a specific tree level, for the next (lower) level, we can directly find the matched node pairs from their child node pairs, and ignore all the other node pairs. By repeating this process level by level, we can safely prune a large number of unmatched pairs of nodes and obtain all the e-matches.

3.3 The Join Order for MSJ

We propose a simple yet effective and efficient method to determine the join order, denoted by MSJOrder, which relies on a key observation that, in an IR-tree (or other tree-based indexes), with a typical node capacity in the hundreds and a fill-factor of approximately 0.7, the leaf level makes up well beyond 99% of the index [16]. This implies that, the number of non-leaf nodes is much smaller than that of leaf nodes. Meanwhile, the non-leaf nodes, especially those at the lowest level, generally well summarize the objects' locations, which inspires the design of PJ. Thus, we propose to use the number of matched non-leaf node pairs to approximate the join order.

Specifically, we perform three steps in MSJOrder. First, for each edge, we apply the PJ algorithm until the handling of leaf nodes, to find all the matched pairs of non-leaf nodes at the lowest level. Second, we count the number of matched pairs of non-leaf nodes for each edge. Third, we perform the same greedy algorithm as that of MPJOrder, where the estimated numbers of e-matches of edges are replaced by the corresponding numbers of matched non-leaf node pairs, and obtain a join order Γ. Note that all the sets of matched pairs of non-leaf nodes are kept after running MSJOrder, as they will be reused later in the join process.

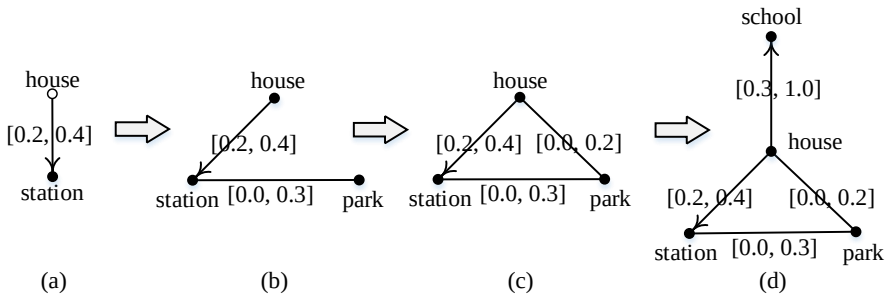


Figure 4: Illustrating the process of MSJ.

3.4 The Overall MSJ Algorithm

Based on the bounded pattern computation, PJ algorithm, and join order above, we develop the MSJ algorithm. Specifically, we first compute the bounded pattern $\widehat{P}$ of P using dynamic programming and refine P. Then, we find the matched non-leaf node pairs for all the edges of P in a collective manner, through which the join order Γ is computed. After that, for each edge of Γ, we compute its e-matches by using PJ. Finally, we follow the order Γ and compute all the matches by linking these e-matches. In Figure 4, we illustrate the process of joining e-matches in MSJ, which follows a particular join order. For detailed pseudocodes, please refer to [6, 10].

4 Application: SpaceKey

In this section, we present *SpaceKey*, a system for retrieving and visualizing spatial objects returned by various SKQs, such as SPM query, mCK query, etc. Moreover, it allows users to perform comparison analysis between different SKQs. We further customize it for a real application of property searching, by resolving practical issues, as well as realizing some additional functionalities related to property searching. We first introduce SpaceKey in Section 4.1 and then present the customized system for property searching in Section 4.2.

4.1 SpaceKey: Exploring Patterns in Spatial Databases

In SpaceKey, to issue an SPM query, the user can easily draw a spatial pattern and view the query results. Figure 5 shows the user interface of SpaceKey configured to run on a dataset of UK Points of Interest (PoIs). To draw a pattern, a user can drag icons (representing keywords) from the panel (bottom-left) to create vertices (top-left), and then create edges by linking pairs of icons. Their distance intervals and relationship can be edited using the pop-up panel, which overlaps with the map in Figure 5. Once the user clicks the "Query" button, all the matches will be returned and the user can view them one by one through the "Previous" and "Next" buttons. Additionally, SpaceKey allows a query user to edit a previous spatial pattern, which would help her to interactively compose a new query pattern and explore the matched objects.

Besides, SpaceKey can seamlessly supports other SKQ algorithms. Currently, we have incorporated three additional SKQ query types: mCK [14], CoSKQ [1, 15], and minSK [3]. Thus, a user can choose which query model to use to fit her needs. In Figure 5, a query user can issue a specific query after clicking its type on top of the left panel. Moreover, SpaceKey provides an Application Programmer Interface (API), which consists of a list of functions. To plug a new SKQ query type or algorithm into SpaceKey, the user only needs to follow the API and slightly modifies the HTML codes in the panel under the logo, and then she can easily view the query results and compare them with other SKQ query algorithms.

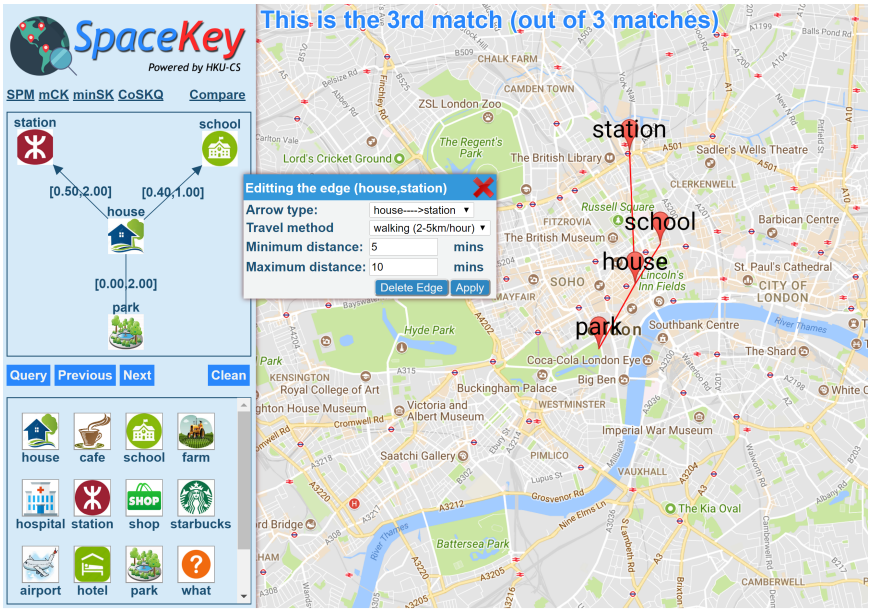


Figure 5: User interface of SpaceKey [9].

Furthermore, SpaceKey can report statistics about the results returned by different algorithms, such as the number of object sets returned, the average pair-wise distance between objects in each set, and the diameter of objects. These features allow users to perform a detailed comparison between the results output from different algorithms. Additionally, users can plug in their own analysis functions through the API provided. For more details of SpaceKey, please refer to [9].

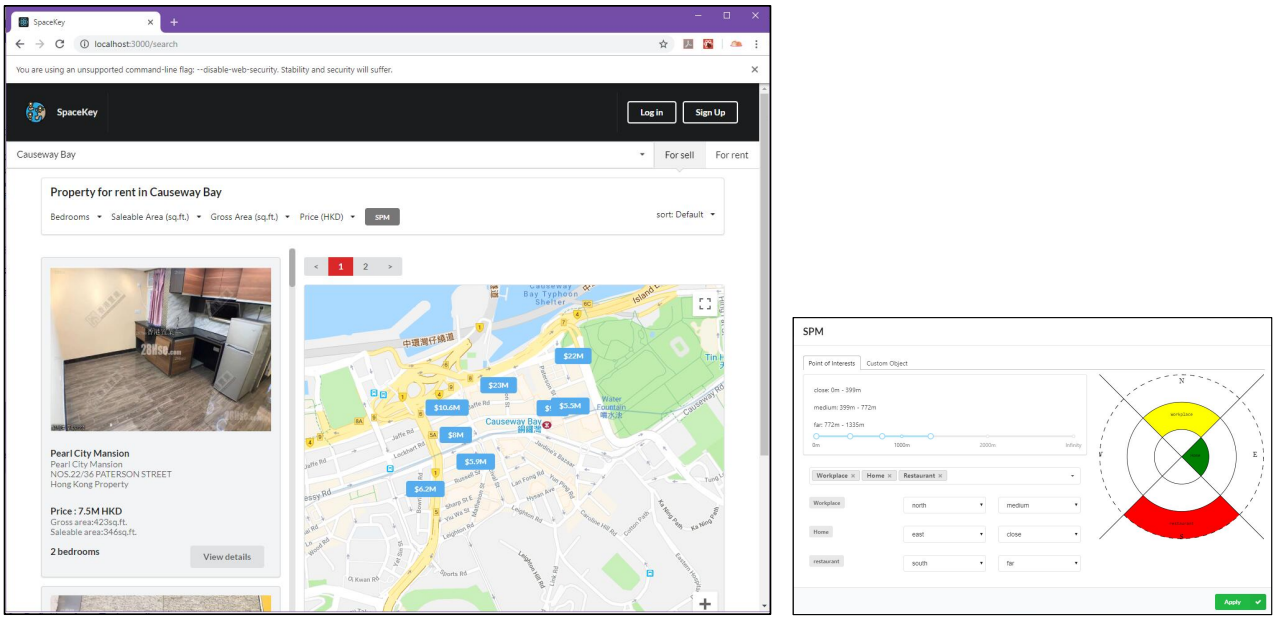
4.2 SpaceKey for Property Searching

The demand for property is steadily rising with the significant market size of real-estate across the world. Existing property searching applications only provide simple filtering functionalities about the properties, such as price, size, etc. However, in reality, users might have some complex requirements about the surroundings of their desired properties. To deal with this unsatisfied demand, we customize SpaceKey so that it enables users to perform property searching. Next, we first introduce the interfaces and then discuss additional features.

Figure 6(a) shows the main interface of the customized system. After specifying the query requirements (e.g., room types, areas, and price), the query results will be shown on the map, including the points of interests (POIs) and the properties that satisfies the requirements. On the left, the detailed information of the properties is displayed and the "View details" button will redirect the user to the original website. The user can also get the routing info between locations through simple maneuvers.

Figure 6(b) is the panel for specifying user's desired surroundings. SpaceKey allow users to specify three distance intervals, which stands for close, medium and far. Users could modify these three intervals by dragging points along the axis. In the input bar the user can search and add existing keywords to specify the pattern. For directions, the user can choose north, west, east, west or any; For distance, the user can choose close, medium, far, any or unwanted. On the right is a comprehensive pattern display: As the user add more keywords to the pattern, their respective area will be highlighted. In this panel, the user can also specify custom objects.

Besides SPM patterns, it includes additional features related to the housing location, which might be helpful for the user to find the desired property. The additional features include:



(a) Search page overview

(b) SPM filter panel

Figure 6: SpaceKey overview.

- **Unwanted objects**: the users can set a distance constraint to "unwanted" when they are specifying their desired surroundings.
- **Custom objects**: for users who want to find a property to have strong ties to some specific locations, for example, his workplace or friends' house, it allows them to specify custom objects in the spatial pattern. In other words, the user can add temporary objects to the POI database with its unique keyword, which enables the user with the ability to specify the distance/directional constraints between his/her desired property and the custom object.
- **Directional relations**: users can specify the direction of their desired PoIs. For example, a Hong Kong user may want a temple in the north of her property for the sake of fortune.

5 Conclusions and Future Work

In this paper, we propose the SPM problem and devise efficient solutions to address the SPM problem. Based on the SPM query, we develop a system called SpaceKey. We also develop two applications, namely spatial pattern visualisation and advanced hotel search. There are interesting avenues for future work, for example:

• **Diversified top-k SPM:** As shown by Spacekey, sometimes the number of matches returned by an SPM query is enormous, if the query pattern is not well specified. As a result, the user may have difficulty to rank the numerous results and choose the best ones. To address this issue, it is desirable to enable a new kind of top-k query that maximized the "diversity" of the query results. For example, we can define the diversity as the number of distinct objects in the returned matches, and then study how to find the top-k matches that maximize the diversity.

• **Approximate SPM**: An SPM query may not always return one or more matches. This can be because the query pattern is rare, and consequently there are no instances that precisely match it. In such cases, it is better to find sets of objects that are approximately matched with the query pattern. For instance, if we cannot find an exact match for the query pattern, the query algorithm can automatically relax the distance constraints such that

there is at least one match for the pattern in the database. Note that a key challenge in such approximate SPM is how to minimize relaxation on the constraints. Another choice is to ask the user to specify a priority value of each edge in the pattern, and then edges with higher priority should be satisfied when finding the matched objects.

• **Distributed solutions of SPM:** Real-world spatial datasets, such as Google Maps, are often kept in a cluster of machines distributedly. In this case, it is necessary to develop distributed solutions to support SPM query. Besides, to accelerate the SPM queries, it is natural to develop distributed solutions that support SPM queries to improve the throughput of query processing.

• **Spatial pattern discovery:** In this paper, we assume that the query users can specify proper patterns as input. This assumption, however, may be too strong for users who are not familiar with the spatial database. To remedy this issue, it is better to suggest some meaningful patterns. To discover these patterns, a natural idea is to adapt the existing solutions of frequent subgraph mining such that the frequent spatial patterns can be discovered from the spatial database.

• **More features in spatial patterns:** There are many possible ways to enrich the spatial pattern. For example, we can allow each vertex of the pattern to carry multiple keywords with logical operations (e.g., "AND" and "OR"), supporting for instance the case where users want to find a house that has nearby a hospital or a doctor. Also, the distance constraint can be changed, such that multiple distance intervals can be specified between two vertices in the spatial pattern.

Acknowledgments

Reynold Cheng was supported by the Research Grants Council of Hong Kong (RGC Projects HKU 17229116 and 17205115) and the University of Hong Kong (Projects 102009508 and 104004129). We would like to thank Jikun Wang, Budiman, and Yin Yue for their implementation of SpaceKey.

References

[1] X. Cao, G. Cong, C. S. Jensen, and B. C. Ooi. Collective spatial keyword querying. In *SIGMOD*, pages 373–384. ACM, 2011.

[2] L. Chen, G. Cong, C. S. Jensen, and D. Wu. Spatial keyword query processing: an experimental evaluation. *PVLDB*, pages 217–228, 2013.

[3] D. Choi, J. Pei, and X. Lin. Finding the minimum spatial keyword cover. In *ICDE*, pages 685–696. IEEE, 2016.

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

[5] K. Deng, X. Li, J. Lu, and X. Zhou. Best keyword cover search. *TKDE*, 27(1):61–73, 2015.

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

[7] Y. Fang, R. Cheng, X. Li, S. Luo, and J. Hu. Effective community search over large spatial graphs. *PVLDB*, 10(6):709–720, 2017.

[8] Y. Fang, R. Cheng, W. Tang, S. Maniu, and X. Yang. Scalable algorithms for nearest-neighbor joins on big trajectory data. *IEEE Transactions on Knowledge and Data Engineering*, 28(3):785–800, 2015.

[9] Y. Fang, R. Cheng, J. Wang, Budiman, G. Cong, and N. Mamoulis. SpaceKey: exploring patterns in spatial databases. In *ICDE*, pages 1577–1580. IEEE, 2018.

[10] Y. Fang, Y. Li, R. Cheng, G. Cong, and N. Mamoulis. Evaluating pattern matching queries for spatial databases. In *VLDB Journal*. Springer-Verlag New York, Inc., 2019.

[11] 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*, 31(4):783–798, 2018.

[12] B. Gallagher. Matching structure and semantics: A survey on graph-based pattern matching. *AAAI FS*, 6:45–53, 2006.

[13] A. B. Gallion and S. Eisner. The urban pattern; city planning and design,. 1950.

[14] T. Guo, X. Cao, and G. Cong. Efficient algorithms for answering the m-closest keywords query. In *SIGMOD*, pages 405–418. ACM, 2015.

[15] C. Long, R. C. W. Wong, K. Wang, and A. W. C. Fu. Collective spatial keyword queries: A distance owner-driven approach. In *SIGMOD*, 2013.

[16] D. Wu, M. L. Yiu, G. Cong, and C. S. Jensen. Joint top-k spatial keyword query processing. *TKDE*, 2012.

[17] Y. Wu, J. M. Patel, and H. Jagadish. Structural join order selection for xml query optimization. In *ICDE*, pages 443–454. IEEE, 2003.

[18] W. Zhang, R. Cheng, and B. Kao. Evaluating multi-way joins over discounted hitting time. In *ICDE*, pages 724–735. IEEE, 2014.

[19] D. Zhang et al. Keyword search in spatial databases: towards searching by document. In *ICDE*, pages 688–699. IEEE, 2009.

[20] L. Zou, L. Chen, and M. T. Özsu. Distance-join: pattern match query in a large graph database. *PVLDB*, 2(1):886–897, 2009.