Distributed and Parallel Databases (2023) 41:511–548
https://doi.org/10.1007/s10619-021-07333-2

A spatial co-location pattern mining framework insensitive to prevalence thresholds based on overlapping cliques

Vanha Tran[1] · Lizhen Wang[1] · Lihua Zhou[1]

Accepted: 11 December 2020 / Published online: 18 March 2021

Abstract
Discovering spatial co-location patterns is a process of finding groups of distinct spatial features whose instances are frequently located together in spatial proximity. A co-location pattern is prevalent if its participation index is no less than a minimum prevalence threshold given by users. Most of the existing algorithms are very sensitive to the prevalence threshold, when users change the prevalence threshold, these algorithms have to re-collect table instances and re-calculate participation indexes of all patterns for mining the prevalent patterns that users expect to acquire. To tackle this issue, we propose an overlapping clique-based spatial co-location pattern mining framework (OCSCP). In our framework, we design a two-level filter mechanism with the first level is a feature type filter and the second level is a neighboring instance filter. By employing the mechanism, under a certain neighbor relationship, spatial instances are divided into a set of overlapping cliques and each clique is also a co-location instance of a pattern. And then, a co-location pattern hash map structure is designed to store table instances of patterns based on these overlapping cliques. The participation index of each pattern can be fast and directly calculated from the hash map structure. Thus, when the prevalence threshold is changed, the proposed framework does not need to re-gather table instances, and the mining result can be adaptively and quickly given to users. The proposed algorithms are performed on both synthetic and real-world data sets to demonstrate that our algorithms can rapidly respond to user requirements compared to the previous algorithms.

Keywords Spatial co-location pattern · Overlapping clique · Hash map · Participation index · Prevalence threshold

✉ Lizhen Wang
lzhwang@ynu.edu.cn

Extended author information available on the last page of the article

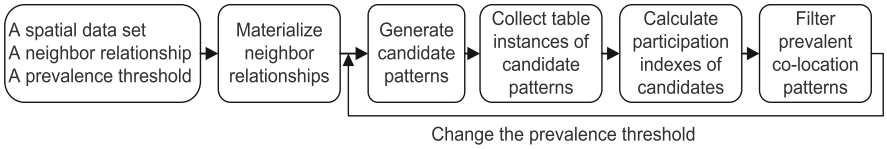


Fig. 1 A common framework of spatial co-location pattern mining

1 Introduction

Nowadays, many different domains have generated large and rich spatial data sets. For example, weather and climate monitoring or satellite observations can produce terabytes of spatial data each day [5]. This brings a big challenge to spatial data mining. As an important branch of spatial data mining, spatial co-location pattern mining technology has been proven valid to identify interesting and previously unknown spatial knowledge hidden in the spatial data sets.

The objective of spatial co-location pattern mining is to find a set of groups of distinct spatial features whose instances are frequently located together in spatial proximity. Spatial co-location patterns provide distribution rules of spatial features in geographic space. The rules help users in many fields with planning and deciding. They can make intelligent decisions and arrange smart plans based on the rules. For example, in city facilities, a group of {Hotel, Shopping center, Restaurant} is a prevalent co-location pattern because they frequently appear together in nearby geographic space. Users can use the information about the pattern to plan their trip, e.g., if they go shopping at a shopping center, they can easily find out restaurants and hotels in the near space of the shopping center. This co-location pattern also can be provided to businessmen who want to arrange a new store, they would like to know the different types of businesses that frequently appeared together. According to the information obtained from pattern {Hotel, Shopping center, Restaurant}, they can determine the profitability of similar stores and their neighborhood stores to make a better decision relating to the new store.

Another example is in the transportation domain. Based on traffic data sets we can find out a co-location pattern as {Frequent point of traffic accidents, Traffic jam, Police car}. The information about the pattern is provided to the traffic manager to properly arrange police presence and rapid handle traffic accidents to avoid traffic jams. The same benefit of spatial co-location patterns is also embodied in the mobile communication domain when the mobile company wants to provide attractive location-sensitive advertisements and recommendations.

Overall, the spatial co-location pattern mining technique is a powerful tool for discovering knowledge from massive spatial data sets, it can be applied to many fields such as disease control and public health [12], urban construction [3], business [11], transportation and location-based services [32], mobile communication [6], public security [10], environmental management [1], social science [17], geology [19], astronomy [2] and so on.

Summarizing the existing spatial co-location pattern mining algorithms, there is a common mining framework which is drawn in Fig. 1. This spatial co-location

pattern mining framework uses an expensive generate-test candidate model [33]. This framework has six phases. In the first phase, users give a spatial data set, a neighbor relationship R, and a minimum prevalence threshold *min_prev*. The neighbor relationships between instances are materialized in the second phase. An then, in the third phase, a set of candidate patterns (a candidate is symbolized by c) is generated based on the spatial feature types of the input data set. The number k of distinct feature types in a candidate is called the size of the candidate (size k candidate, c_k). Next, the table instance (symbolized by $T(c_k)$) of each candidate, which is a set of co-location instances (also called row instances, a co-location instance is a set of instances in which all the instances have the neighbor relationship with each other), is collected in the fourth phase. This is the most expensive job in the whole mining process [5]. After that, the participation index (which measures the prevalence of a pattern) of each candidate is calculated in the fifth phase. Finally, in the sixth phase, candidates whose participation indexes are larger than or equal to the minimum prevalence threshold *min_prev* are filtered and they are called prevalent spatial co-location patterns.

Under a certain neighbor relationship, if mining results do not meet the requirement, users must adjust the minimum prevalence threshold *min_prev* to obtain expected results. This mining framework has to re-execute from the third phase to the sixth phase since it is an incremental mining process. Assume that c_k and $c_k^{'}$ are two size k patterns, $T(c_k)$ and $T(c_k^{'})$ are the table instances of the two patterns, respectively. First, the size $(k+1)$ candidate, c_{k+1}, is generated by joining features in c_k and $c_k^{'}$. Second, the table instance of c_{k+1}, $T(c_{k+1})$, is discovered based the table instances of c_k and $c_k^{'}$ (the neighboring instances in $T(c_{k+1})$ is obtained by joining and validating based on neighboring instances in $T(c_k)$ and $T(c_k^{'})$). Given a prevalence threshold *min_prev*, if c_k and/or $c_k^{'}$ are not prevalent with this value of the prevalence threshold, candidate c_{k+1} which is constructed from c_k and $c_k^{'}$, must not be prevalent (according to Lemma 1 that will be given later in Sect. 3), and it does not need to gather table instance $T(c_{k+1})$ of c_{k+1}. However, when users change *min_prev* to make c_k and $c_k^{'}$ prevalent, it needs to re-collect table instances $T(c_k)$ of c_k and $T(c_k^{'})$ of $c_k^{'}$ besides collecting table instance $T(c_{k+1})$ of candidate c_{k+1} and then the participation index of c_{k+1} is calculated to judge the prevalence of it. It means that Steps 3 to 6 have to be re-performed.

Many mining algorithms belonging to this framework have been published so far, e.g., join-based [16], partial-join [31], join-less [30], iCPI-tree [21], order-clique-based [22], SGCT [27], MGPUCPM [4], GPU grid-based [15]. However, this mining framework has some deficiencies as follows:

(1) The generate-test candidate model is very time-consuming. The set of candidates and their table instances will become very huge if the spatial features and instances are large and/or data sets are dense (the number neighbors of instances are huge). Table instances corresponding to candidate patterns must be collected and tested to obtain real prevalent spatial co-location patterns. It takes a lot of time to process such a huge candidate set and its table instances. To demonstrate this point, Fig. 2 shows the average execution time in each phase

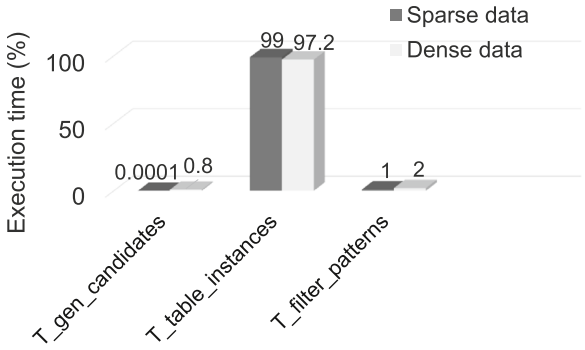


Fig. 2 Percentage of the execution time for each phase based on the mining framework shown in Fig. 1

of the joinless algorithm [30] when the number of features and instances are set to 20 and 15,000, respectively. In Fig. 2, *T_gen_candidates*, *T_table_instances*, and *T_filter_patterns* are the execution time of generating candidates (correspond to the third phase in Fig. 1), collecting table instances of candidates (the fourth phase), and calculating and filtering prevalent patterns (the fifth and sixth phases), respectively. It is clear to see that the largest fraction of the execution time is devoted to collecting table instances of candidate patterns.

(2) This mining framework is particularly sensitive to the minimum prevalence threshold. The participation index is an important parameter for measuring the prevalence of a co-location pattern and it is calculated based on the table instance of each candidate. It means that only after finishing collecting table instances of patterns, we can get the participation indexes of patterns. When users change the minimum prevalence threshold, the mining framework has to re-perform from the third phase to the sixth phase, which needs to re-collect all table instances of the candidate patterns. Therefore, it cannot quickly respond to the change of users in the prevalence threshold. The flexibility of it is poor.

(3) It is hard to quickly mine high-size co-location patterns. The framework shown in Fig. 1 is an incremental mining process. The size $(k + 1)$ candidates and their table instances are generated based on the size k prevalent patterns and their table patterns. That means only when size k patterns are finished, size $(k + 1)$ patterns will be performed. It cannot start mining at any size k $(k > 2)$ patterns. Users sometimes do not want to know all of the sizes of co-location patterns, they only care about a certain size k of co-location patterns, and the framework is hard to do it. The flexibility of the framework is not sufficient. Moreover, although the candidate patterns can be pruned by using the monotonically non-increasing property of the participation index [16], it fails when the minimum prevalence threshold is small. However, the prevalence threshold cannot be set to a big value, because it will miss some significant co-locations [14].

To address these deficiencies, this paper proposes a new method named overlapping cliques-based spatial co-location pattern mining algorithm (OCSCP for short) to discover prevalent co-location patterns. In terms of graphs, a clique is a set of vertices in which all the vertices are adjacent to each other [7]. However, different from

the general definition of cliques, in the spatial co-location pattern domain, instances belong to a clique must be having different feature types. If a clique satisfies this condition, a clique can be viewed as a row instance for mining co-location patterns. Since an instance can belong to more than one clique at the same time, just as an instance participates in multiple row instances. That is why we call them overlapping cliques.

The original idea in this paper is in our previous work [18], which is that if all row instances are obtained first, we do not need to generate and test candidate patterns, table instances of any patterns can be directly collected. In a given neighbor relationship, table instances are fixed because they are based on the distribution of the input data set and the distance threshold. Hence, if users change the minimum prevalence threshold, prevalent patterns can be directly and quickly filtered instead of re-collecting table instances of candidates. Therefore, mining efficiency can be improved significantly.

However, in our previous work, a mining algorithm based on overlapping **maximal clique** partitioning (OMCP for short) was presented. First, the OMCP algorithm utilizes a grid method to divide instances of an input data set into different cells and group nine cells as a block. Then, neighboring instance pairs (two instances have a neighbor relationship) are listed in each block. These neighboring instance pairs are treated as initialization of overlapping maximal cliques. Next, OMCP discovers all overlapping maximal cliques in each block by trying to put instances into the initialized maximal cliques to build maximal cliques. After that, row instances are extracted from the maximal cliques. And table instances of patterns are directly collected by gathering these row instances with the same type of features. Finally, participation indexes of patterns are calculated and prevalent patterns are filtered. The OMCP algorithm only needs to perform once to gain participation indexes of patterns and it does not need to re-collect table instances when the minimum prevalence threshold is adjusted.

In this paper, we design a new method to yield all overlapping **cliques** instead of discovering overlapping **maximal cliques** as the previous work. In the new method, neighbor relationships of instances are materialized into a set of overlapping cliques. First, we devise a two-layer clique structure that is made up by clique headers and clique bodies. Then, a two-level filtering mechanism to find all cliques is designed. The first level is feature type filtering. According to the co-location pattern notion, only belonging to different feature type instances may form row instances. Next, the second level is neighboring instance filtering. One instance which wants to put into the clique bodies to combine bigger cliques if only the instance has a neighbor relationship with the clique header. After obtaining all overlapping cliques, a two-layer hash map structure is utilized to store the cliques. The participation indexes of patterns can directly and quickly calculate from the hash map structure. In summary, the new algorithm designed in this paper has two advantages over the OMCP algorithm:

(1) An instance can be quickly positioned into the cliques that it can be put into to build larger cliques. This method effectively reduces unnecessary verification of neighboring instances when enumerating cliques.

(2) All row instances are generated directly from the overlapping cliques.

The remainder of this paper is organized as follows: Sect. 2 describes the related work. The fundamental concept of spatial co-location pattern mining and the notion of overlapping cliques are represented in Sect. 3. Two overlapping cliques-based spatial co-location patterns mining algorithms are proposed in Sect. 4. The experimental results and analysis are discussed in Sect. 5. The work of this paper is concluded in Sect. 6.

2 Related work

As an important branch of data mining, a large number of spatial co-location pattern mining algorithms have been proposed. These algorithms can roughly fall into three categories, they are traditional mining algorithms, mining algorithms for different spatial data types, and compression co-location pattern mining algorithms.

Some spatial co-location pattern mining algorithms such as join-based [16], partial-join [31], join-less [30], CPI-tree [20], iCPI-tree [21] are divided into the **traditional co-location pattern mining algorithm** category. The join-based, partial-join algorithms use a time-consuming join operation to obtain row instances of candidates. While CPI-tree and iCPI-tree adopt a prefix tree structure to describe neighbor relationships between instances. The two algorithms are no longer using join operations, thus their performances are improved. This type of co-location pattern mining algorithms is on the premise of mining all correct and complete patterns, it focuses on improving the efficiency and storage space of mining algorithms.

The **different spatial data type mining algorithms** effectively deal with some more practical spatial data sets such as fuzzy, interval, uncertain, spatiotemporal data sets, and so on. These data sets cannot be processed by the traditional mining algorithms. Ouyang et al. [13] developed the single co-location pattern (SCP) and range co-location pattern (RCP) algorithms for mining co-location patterns on fuzzy data sets by using a membership threshold and a membership range threshold, respectively. An efficient co-location pattern mining method which deals with interval data sets was proposed by Wang et al. [23]. In this method, semantic proximity is used to measure the proximity between two instances on interval data sets. An algorithm that finds probabilistic prevalent co-locations in spatially uncertain data sets is also developed by Wang et al. [24]. By defining a concept called probabilistic prevalent co-locations, this algorithm tries to find all the co-locations that are likely to be prevalent in a randomly generated possible world. Leibovici et al. [10] developed an algorithm that considers both the location and existence time of instances. This algorithm effectively mines patterns from spatio-temporal data sets. Huang et al. [8] proposed a new measure called the maximal participation ratio to tackle the problem of mining co-location patterns with rare spatial features.

The results of spatial co-location pattern mining often contain numerous patterns, which makes it hard for users to understand or apply. Thus, **compression co-location pattern mining algorithms** have attracted many researchers. An order-clique-based (OCB) and sparse-graph and condensed tree (SGCT) algorithms were

developed for mining maximal co-location patterns (a prevalent co-location pattern c is a maximal pattern if it has no super prevalent patterns) by Wang et al. [22] and Yao et al. [27], respectively. Yoo et al. [28] proposed a mining top-k closed co-location pattern algorithm (a prevalent co-location pattern c is called closed pattern if there exists no proper super prevalent pattern $c' \supset c$ such that the participation indexes of c' and c are equal). A framework for generating condensed co-location sets from spatial databases was developed by Yoo et al. [29]. To address redundant co-location patterns, two algorithms RRclosed and RRnull to perform the redundancy reduction for prevalent co-location patterns were developed by Wang et al. [25]. This researcher group also developed a super participation index-closed (SPI-closed) algorithm which considers the intrinsic characteristics of spatial co-locations and effectively captures the nature of spatial co-location patterns [26]. This type of mining algorithm concentrates on provides rich and practically instructive patterns to users.

To avoid generating candidate patterns, some mining algorithms, which directly mine co-location patterns without generating candidates by utilizing maximal cliques, had been developed. A maximal clique can extract to a set of sub cliques that are row instances of co-location patterns. Kim et al. [9] developed a maximal clique generating algorithm for spatial co-location pattern mining called AGSMC. AGSMC builds a tree-type data structure and generates maximal cliques by scanning the tree. And then, the maximal cliques are used as transaction-type data for mining spatial co-location patterns. AI-Naymat [2] proposed an algorithm named GridClique to generate maximal cliques from large spatial data sets that can be converted to transaction data and then using the association rule mining algorithms for generating spatial co-location patterns. More recently, a spatial co-location pattern mining algorithm based on overlap maximal clique partidesignedtioning called OMCP was proposed in [18]. However, extracting maximal cliques from a graph is known as an NP-hard problem, hence, the performance of these algorithms is limited.

In this paper, we propose an overlapping clique-based spatial co-location pattern mining framework (OCSCP) and we design two mining algorithms based on OCSCP. The OCSCP algorithm employs **cliques** instead of **maximal cliques** without converting the cliques to transaction data. To efficiently discover all cliques, designing a two-layer clique structure with clique headers and clique bodies, the instances in the clique bodies are gradually expanded by a two-level filtering strategy. And then, a co-location pattern hash map structure is developed to gather table instances of all patterns. The participation indexes of co-location patterns can be calculated quickly by scanning the co-location pattern hash map structure. The principal contributions of this work are summarized as follows:

(1) Non-generate-test candidate mining algorithms are developed. The generate-test candidate model is no longer used in the proposed algorithms. Hence, the mining performance is improved significantly.
(2) The proposed algorithms are insensitive to the prevalence threshold. When users change the minimum prevalence threshold for their different mining purposes, our algorithms can adaptively and quickly give new results.

(3) The proposed algorithms can efficiently and directly mine high-size patterns.

3 Preliminaries

In this section, we first briefly review some basic notions of spatial co-location pattern mining. And then, some definitions of overlapping cliques are described in detail.

Definition 1 (Co-location pattern): Given a set of spatial instances, $S = \{S_1, ..., S_n\}$, where $S_i = \{f_{i_1}, ..., f_{i_m}\}$ is corresponding instances of feature $f_i \subset F = \{f_1, ..., f_n\}$, each instance in S is formed by ⟨feature type, instance identification, location⟩, and a neighbor relationship R over on S. A spatial co-location pattern is a subset of F, $c = \{f_1, ..., f_k\}$ $(1 < k < n)$, whose instances are frequent neighbors under the neighbor relationship R. The number of the distinct features in c is k and it is called the size of c.

Definition 2 (Row instance and table instance): A row instance $I = \{f_{1_j}, ..., f_{k_q}\}$ $(1 \leq j, q \leq m)$ is a subset of S which includes all feature types in c and each instance in I has a neighbor relationship with the others. A group of all row instances of c is called the table instance of c and denoted as $T(c)$.

Definition 3 (Participation ratio): The participation ratio of a feature f_i in a co-location pattern c is denoted by

$$PR(c, f_i) = \frac{N(f_i, T(c))}{N(f_i, S)} \tag{1}$$

where $N(f_i, T(c))$ and $N(f_i, S)$ are the number of distinct instances of f_i in $T(c)$ and the number of instances of f_i in S, respectively.

Definition 4 (Participation index): The participation index of pattern c is denoted by

$$PI(c) = min\{PR(c, f_i)\} \tag{2}$$

Definition 5 (Prevalent co-location pattern): Given a minimum prevalence threshold, *min_prev*, if the participation index of pattern c is no less than the minimum prevalence threshold, $PI(c) \geq min\_prev$, pattern c is called a prevalent co-location pattern.

Lemma 1 (The anti-monotonicity property of the participation ratio and the participation index): *The participation ratio and the participation index are anti-monotonicity with the increase of the size of patterns.*

$$PR(c', f_i) \leq (PR(c, f_i)) \tag{3}$$

Table 1 The definition of the spatial co-location pattern mining problem

Given	(1) A set of spatial instances S, each instance in S has an information vector as ⟨feature type, instance identification, location (x, y)⟩.
	(2) A spatial neighbor relationship R, normally a Euclidean distance metric is employed with a neighbor distance threshold d.
	(3) A minimum prevalence threshold min_prev.
Find	All prevalent co-location patterns.

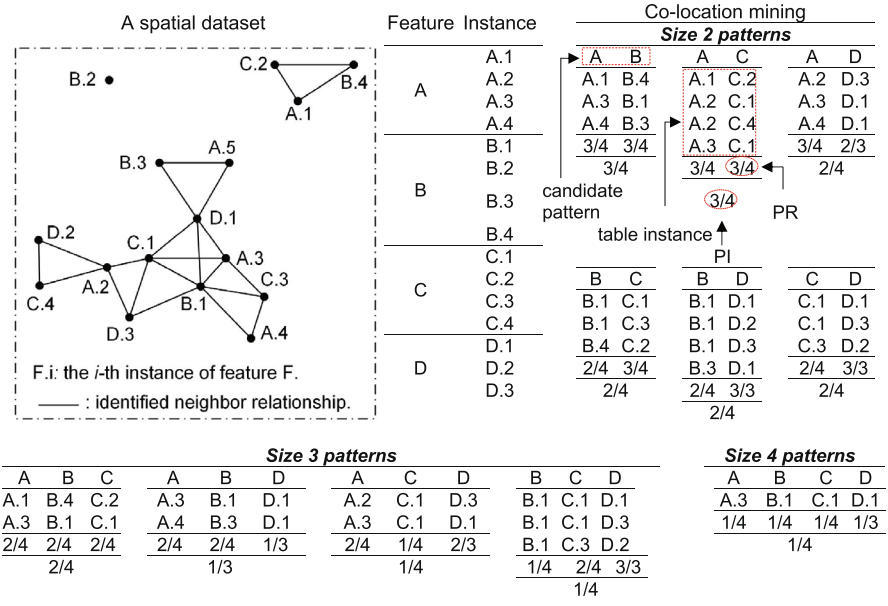


Fig. 3 An illustration of spatial co-location pattern mining

$$PI(c') \leq PI(c) \tag{4}$$

where c' is a superset of c.

Proof Please reference [9] in detail. □

The problem of spatial co-location pattern mining is defined in Table 1.

For example, Fig. 3 illustrates the mining process of spatial co-location patterns in the traditional framework shown in Fig. 1. The input spatial data set has four spatial features, A, B, C, D and they have 4, 4, 4, 3 instances, respectively. Under a neighbor relationship given by users, if two instances have the neighbor relationship, they are connected by solid lines. First, size 2 candidates are generated, {A, B}, {A, C}, {A, D}, {B, C}, {B, D}, and {C, D}. Then, the table instance of each candidate

is collected. For example, the table instance of {A, B} is $T(\{A, B\})$ = {{A.1, B.4}, {A.3, B.1}, {A.4, B.3}}. After that, the participation ratios and participation indexes of these candidates are calculated, e.g., $PR(\{A, B\}, A) = N(T(\{A, B\}), A)/N(A, S) = 3/4$; $PR(\{A, B\}, B) = N(T(\{A, B\}), B)/N(B, S) = 3/4$. Thus, the participation index of candidate {A, B} is $PI(\{A, B\}) = \min\{3/4, 3/4\} = 0.75$. Assuming that users set $min\_prev = 0.6$, since $PI(\{A, B\}) = 0.75 > 0.6$, thus {A, B} is a prevalent pattern.

It is can be seen that when the value of the prevalence threshold is $min\_prev = 0.6$, the size 2 prevalent patterns are {A, B} and {A, C}. However, if users change the prevalence threshold $min\_prev = 0.3$, all the six size 2 patterns are prevalent. As a result, we have to generate size 3 candidates which are constructed based on the size 2 prevalent patterns, they are {A, B, C}, {A, B, D}, {A, C, D}, and {B, C, D}. Collecting their table instances, calculating participation ratios and participation indexes, and filtering size 3 prevalent patterns have to be re-performed to determine whether these candidates are prevalent. The mining processing will continue to perform generating candidates based on lower size prevalent patterns and testing the prevalence of the candidates until there are no higher candidates generated.

It is in clear view that if users change the prevalence threshold, the mining process has to re-perform from size 2 candidates to the highest size candidates. It makes the generate-test mining framework less flexible.

Moreover, it is easy to see that the anti-monotonicity of the participation ratio and participation index are satisfied, e.g., $PR(\{A, B\}, A) = 3/4 > PR(\{A, B, C\}, A) = 2/4$ and $PI(\{A, B, D\}) = 1/3 > PI(\{A, B, C, D\}) = 1/4$. Since {A, B, D} is not prevalent, {A, B, C, D} is not a prevalent pattern too.

Definition 6 (Clique): Give a set of spatial instances S and a neighbor relationship R. A clique is a subset of S which satisfies the following conditions:

- All instances in the subset belong to different feature types.
- All instances in the subset have the neighbor relationship R with each other.

Comparing with the definition of row instances in the term of spatial co-location pattern mining, it is easy to recognize that a clique is a row instance of a co-location pattern. It is known that cliques are a notion in graph analysis. However, different from the graph analysis, in the spatial co-location pattern mining domain, all instances in cliques have to belong to different feature types. For example, in Fig. 3, {A.1, B.4, C.2} is a clique and it is a row instance of pattern {A, B, C}.

Definition 7 (Overlapping cliques): Cliques that hold one or more same instances are called overlapping cliques.

If a clique has no common instances with other cliques, it is an independent clique. For the convenience of description, we uniformly name overlapping cliques. From the illustration of spatial co-location pattern mining shown in Fig. 3, we discover that an instance can belong to multiple cliques. For example, instance C.1 belongs to three cliques, they are {A.2, C.1, D.3} which is a row instance of pattern {A, C, D}, {B.1, C.1, D.3} which is a row instance of pattern {B, C, D}, and {A.3, B.1, C.1, D.1} which is a row instance of pattern {A, B, C, D}. Cliques {A.3, C.1, D.3}, {B.1, C.1, D.3} and {A.3, B.1, C.1, D.1} are overlapping cliques.

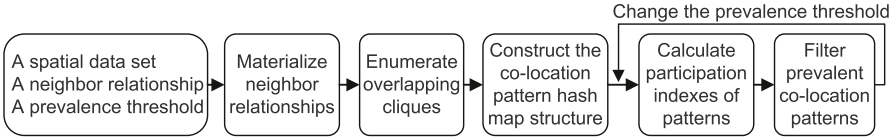


Fig. 4 The proposed mining framework

From the graph analysis point of view, give a spatial data set and a neighbor relationship R, all neighbor relationships of instances can be partitioned into a set of overlapping cliques. If we obtain all cliques of the input data set, all table instances are also collected. Instead of according to the minimum prevalence threshold to collect table instances of candidate patterns from small-size to large-size, we enumerate all overlapping cliques on S. Then a co-location pattern hash map structure is constructed to store these cliques. Participation indexes of patterns can be easy to calculate from the hash map structure. Finally, comparing to the minimum prevalence threshold given by users, prevalent patterns are filtered. Hence, we propose a mining framework that is drawn in Fig. 4. This framework has also six phases. The first, second, fifth, and sixth phases are the same as in the framework shown in Fig. 1. There are two differences, the third phase enumerates all overlapping cliques and the fourth phase constructs a co-location hash map structure to store these cliques and calculate participation indexes conveniently. The two phases are described in the next section in detail.

4 Overlapping clique-based spatial co-location pattern mining algorithms

In this section, we first describe two finding overlapping clique methods in detail. And then, a co-location pattern hash map structure is proposed to store these cliques. After that, calculating participation indexes of patterns from the hash map structure is given. Finally, filtering prevalent pattern method is represented.

4.1 A basic algorithm

4.1.1 Enumerating overlapping cliques

Definition 8 (Ordered features and instances): Given f_{i_j} and f_{k_q} are two instances. We call feature f_i is smaller than feature f_k if $f_i < f_k$, $\forall i, k$ in the lexicographical order. And instance f_{i_j} is smaller than instance f_{k_q}, $f_{i_j} < f_{k_q}$, if they meet one of the following conditions:

- $f_i < f_k$ for all i and k
- $f_i = f_k$ and $j < q$

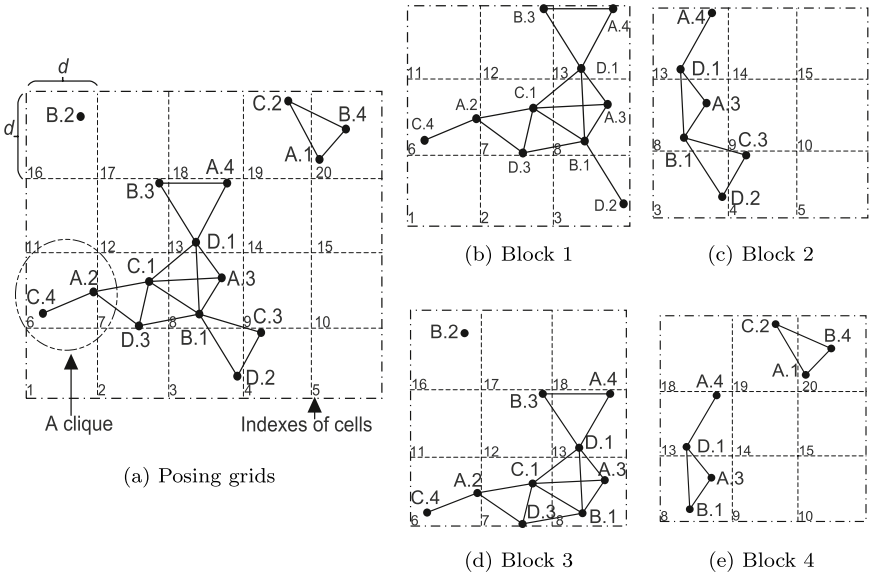


Fig. 5 Posing rectangular grids with a size $d \times d$ and dividing into blocks on the input data set

Definition 9 (Star neighborhood): The star neighborhood of a spatial instance f_{i_j} is a set of spatial instances which have a neighbor relationship R with it and denoted as

$$SN(f_{i_j}) = \{f_{k_q} \subset S | f_{i_j} < f_{k_q} \wedge R(f_{i_j}, f_{k_q})\} \tag{5}$$

where $R(f_{i_j}, f_{k_q})$ represents f_{i_j} and f_{k_q} having the neighbor relationship R.

In this paper, we adopt a Euclidean distance metric with a distance threshold d to determine the neighbor relationship R, i.e., $R(f_{i_j}, f_{k_q}) \Leftrightarrow dist(f_{i_j}, f_{k_q}) \leq d$ with $dist(f_{i_j}, f_{k_q})$ is the distance between f_{i_j} and f_{k_q}. The star neighborhood of each instance is converted to a hash map (named neighboring hash map) with the key is a string constructed by instance f_{i_j} with each its neighboring instance and the value is 1 that represents that the two instances have a neighbor relationship. Since instances are ordered, each neighboring instance pair is only stored once, there are no repeating elements in the neighboring hash map.

To improve the efficiency of finding neighboring instance pairs, a grid with size $d \times d$ of cells is posed on the input data set. Besides, to handle dense data when running algorithms in a limited memory environment, we group every nine cells as a block and each time we only deal with one block. In order to ensure that no neighboring instance pairs are cut off between blocks, we employ an overlapping block partitioning scheme (as shown in Fig. 5). When finding the neighboring instances of an instance f_{i_j}, instead of calculating the distance of f_{i_j} with all other instances in the input data set and comparing with the distance threshold d, we only need to validate

Block ID	Ordered instances	Star neighborhoods	Neighboring hash map	Initial cliques
1	A.2,A.3, A.4, B.1, B.3, C.1, C.4, D.1, D.2, D.3	A.2 : C.1, C.4, D.3 A.3 : B.1, C.1, D.1 A.4 : B.3, D.1 B.1 : C.1, D.1, D.2, D.3 B.3 : D.1 C.1 : D.1, D.3	A2C1: 1 A2C4: 1 A2D3: 1 A3B1: 1 A3C1: 1 A3D1: 1 A4B3: 1 A4D1: 1 B1C1: 1 B1D1: 1 B1D2: 1 B1D3: 1 B3D1: 1 C1D1: 1 C1D3: 1	A : {A.2 : [[C.1], [C.4], [D.3]], A.3 : [[B.1], [C.1], [D.1]], A.4 : [[B.3], [D.1]]} B : {B.1 : [[C.1], [D.1], [D.2], [D.3]], B.3 : [[D.1]]}, C : {C.1 : [[D.1], [D.3]]}
2	A.3, A.4, B.1, C.3, D.1, D.2	A.3 : B.1, D.1 A.4 : D.1 B.1 : C.3, D.1, D.2 C.3 : D.2	A3B1: 1 A3D1: 1 A4D1: 1 B1C3: 1 B1D1: 1 B1D2: 1 C3D2: 1	A : {A.3 : [[B.1], [D.1]], A.4 : [[D.1]]} B : {B.1 : [[C.3], [D.1], [D.2]]} C : {C.3 : [[D.2]]}
3	A.2,A.3, A.4, B.1, B.3, C.1, C.4, D.1, D.3	-	-	-
4	A.1, A.3, A.4, B.1, B.4, C.2, D.1	A.1 : B.4, C.2 A.3 : B.1, D.1 A.4 : D.1 B.1 : D.1 B.4 : C.2	A1B4: 1 A1C2: 1 A3B1: 1 A3D1: 1 A4D1: 1 B1D1: 1 B4C2: 1	A : {A.1 : [[B.4], [C.2]] A.3 : [[B.1], [D.1]] A.4 : [[D.1]]} B : {B.1 :[[D.1]] B.4 : [[C.2]]}

- : no values

Fig. 6 An illustration of star neighborhoods, neighboring hash map, and initial cliques

the neighbor relationship of f_{i_j} with the instances that fall into the same block with f_{i_j}. In addition, since these instances are ordered, only the instances that are smaller than f_{i_j} are needed to validate. Thus, computing the distance between instance pairs and comparing the distance with the distance threshold are reduced efficiently.

For example, Fig. 5b–e plot the result of the partitioning of the input data set shown in Fig. 5a into a set of blocks with each block contains nice cells. If the star neighborhood of A.4 is required, instead of calculating the distance between A.4 with the other 13 instances, it only calculates the distance between A.4 with B.1, C.3, D.1, and D.2, and compares the distance with the distance threshold. As shown in Fig. 6, the star neighborhood of instance A.3 is {B.1, C.1, D.1} and {C.1, C.3, D.1, D.3} is the neighboring instance set of B.1. A.3 and its neighborhoods are combined to A3B1:1, A3C1:1, and A3D1:1 that are elements in the neighboring hash map.

However, there is a situation where instances within one block may be totally contained in other blocks, the block is redundant and it does not need to process. As an illustration in Fig. 5, all instances of block 3 are in block 1, thus neighbor relationships of instances in block 3 can be obtained from block 1, and we skip block 3. Since instances within a block are ordered, detecting redundant blocks can be accelerated. Assume that bl_1 and bl_2 are two blocks. For each instance f_{i_j} in bl_1, it only needs to validate the appearance of instance identification j in the set of instances of feature f_i in block bl_2. For example, in Fig. 5a and b, the instances in block 1 and block 2 are $bl_1 = \{A.2, A.3, A.4, B.1, B.3, C.1, C.4, D.1, D.2, D.3\}$ and $bl_2 = \{A.3, A.4, B.1, C.3, D.1, D.2\}$, respectively. To detect whether bl_2 is a redundant block, for A.3 in bl_2, it only needs to check instance identification 3 is in the set of instances of A in bl_1, $S_A = \{A.2, A.3, A.4\}$. If A.3 is in S_A, the validating process continues checking the next instances of bl_2 (e.g., A.4, B.1), else it is terminated immediately, the remaining instances of bl_2 do not need to perform

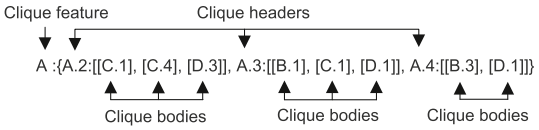


Fig. 7 The structure used to store cliques in our algorithm

validating. For C.3, it is not in {C.1, C.4} of bl_1, and the validating process is broken instantly, it directly returns false that represents bl_1 and bl_2 are two independent blocks. Obviously, to detect whether a block is redundant, one instance in a block only needs to validate with a small number of instances in other blocks.

Definition 10 (Two-layer clique structure): A two-layer clique structure is a hash map structure with key-value pairs where

- The key is a feature type and it is called a clique feature.
- The value is a hash map structure with its key that is an instance (called a clique header) and its value that is a set of neighborhoods of the instance (called a clique body).

For example, Fig. 7 illustrates an example of a two-layer clique structure of feature A. In this structure, the clique feature is feature A, clique headers are instances belonging to A (A.2, A.3, A.4), and clique bodies are the instances that have a neighbor relationship with the clique headers, e.g., C.1, C.4, D.3. We initialize the clique structure based on the star neighborhood. Figure 6 (the fifth column) also lists all initial cliques in each block which is constructed by the star neighborhood of each instance.

It can be seen that a clique header leads to a set of clique bodies, and a clique is combined by a clique header and a clique body. As an example, in block 1 of Fig. 6, clique header A.2 and its clique body set [[C.1], [C.4], [D.3]] form cliques {A.2, C.1}, {A.2, C4} and {A.2, D.3}.

To obtain larger clique bodies, we iterate each instance in a block and try to put the instance to clique bodies. A two-level filtering mechanism, which includes feature type level filtering and neighboring instance-level filtering, is designed to yield all cliques.

Definition 11 (Feature type filter): Given a spatial instance f_{i_j} and a clique with clique feature f_k. f_{i_j} can be put into clique bodies led by f_k if and only if $f_i < f_k, \forall i, k$.

According to Definition 6, a clique is a set of neighboring instances and these instances have to belong to different feature types. When a new instance attempts to insert into a clique body to get a larger clique, the first requirement is that the feature type of the new instance is different from the all features of the current clique. Based on the two-layer clique structure given in Definition 10, a clique feature f_k can draw out a set of cliques and these cliques have to include an instance that its feature type is f_k. In other words, it has already existed an instance of f_k in the clique bodies. Thus, it only needs to compare the feature type of the new instance f_{i_j} with clique

feature f_k to determine f_{i_j} can be or not be added to these cliques led by clique feature f_k.

By ordering the features, it makes generating cliques more efficient. Assume that $f_i < f_k < f_t$ are three ordered features and f_{k_q} is an instance that belongs to feature type f_k. According to the definition of the star neighborhood, if f_{k_q} and an instance of feature type f_i have a neighbor relationship, f_{k_q} has to appear in the star neighbor of the instance, it means that f_{k_q} has already existed in the cliques that are led by f_i. Besides, according to the definition of cliques and row instances, f_{k_q} can not be placed into the cliques led by feature type f_k (since they have the same feature type). Therefore, we only need to consider adding f_{k_q} to the cliques derived from the feature types that are larger than f_k.

For example, listed in block 1 in Fig. 6, instances A.2, A.3, and A.4 can only be put into cliques with clique features are B and C since A < B and A < C; instances B.1 and B.3 have already existed in the cliques that are led by feature A since A.3 and A.4 have a neighbor relationship with B.1 and B.3, respectively. Thus, B.1 can only be considered to put into the cliques that are led by clique feature C.

Definition 12 (Neighboring instance filter): Given a spatial instance f_{i_j} and a clique with clique header f_{k_q}, $f_{i_j} < f_{k_q}$. Instance f_{i_j} can be put into clique bodies led by clique header f_{k_q} if and only if $R(f_{i_j}, f_{k_q})$.

As the designed mechanism of the two-layer clique structure, a clique is combined by a clique header and a clique body. The clique header has a neighbor relationship with all instances in the clique body. When a new instance f_{i_j} that tries to add into the clique body to build a larger clique, f_{i_j} must have the neighbor relationship with the clique header. Thus, it only validates the neighbor relationship of f_{i_j} with the clique header. Besides, since the instances in a block are sorted, we only need to verify the neighbor relationship between f_{i_j} and instances larger than f_{i_j} in a block.

As an example, instance A.2 recorded in block 1 in Fig. 6 cannot be placed into clique bodies led by B.1 and B.3 because A.2 has no neighbor relationships with B.1 and B.3. But A.2 can be put into clique bodies of C.1 since A.2 and C.1 have a neighbor relationship.

When a new instance passes the two levels of filtering, it can be placed in clique bodies to form larger cliques. However, it is still not guaranteed that the new instance can be put into the clique bodies because it may have no neighbor relationships with the all instances in clique bodies. Therefore, we need to further validate the neighbor relationship of the new coming instance with the instances in the clique bodies. The neighbor relationship can be quickly queried by the neighboring hash map. If they are neighborhoods, the new coming instance can be put into the clique bodies to make larger cliques.

For example, after filtering instance A.2 in block 1, A.2 can be positioned into cliques with clique header C.1, {C.1 : [[D.1], [D.3]]}. Then, the neighbor

relationship between A.2 and the clique bodies, [D.1] and [D.3], is validated. A2D1 is not in the neighboring hash map in block 1, A.2 cannot be put into [D.1]. And A2D3 is in the neighboring hash map, thus A.2 is placed into [D.3] to form [A.2, D.3].

Algorithm 1: Enumerating overlapping cliques

```
   Input: a spatial data set S, a neighbor distance threshold d
   Output: a set of overlapping cliques OCs
 1 function EnumerateOverlappingCliques(S, d);
 2 blockSet = PartitionGrid(S, d);
 3 blockSet = DeleteRedundantBlocks(blockSet);
 4 for block ∈ blockSet do
 5     starNBRs = FindNeighbors(block, d);
 6     nbrHashmap = ConstructNeighboringHashmap(starNBRs);
 7     initCliques = InitializeCliques(starNBRs);
 8     for inst ∈ block do
 9         initCliques = UpdateCliques(inst, nbrHashmap, initCliques);
10     OCs.add(initCliques);
11 return OCs;
```

To sum up the above description, Algorithm 1 gives the pseudocode of the processing of enumerating overlapping cliques in detail. Algorithm 1 has three phases. In the first phase, we impose a rectangular grid on the input data set and group all instances of each nine cells as a block by the PartitionGrid function (Step 2) and detect redundant blocks (Step 3). The second phase makes initialization, include finding star neighborhoods (Step 5), neighboring hash map (Step 6), and initial overlapping cliques (Step 7). The third phase iterates each instance in the current block and tries to place the instance into clique bodies if they meet the neighbor relationship by the UpdateCliques function which is described by Algorithm 2 in detail (Step 9). This function returns a set of overlapping cliques that is feed to construct a co-location pattern hash map structure to mining spatial co-location pattern mining in the next section.

Algorithm 2: Updating overlapping cliques

```
   Input: inst, nbrHashmap, initCliques
   Output: initCliques with updated clique bodies
 1 function UpdateCliques(inst, nbrHashmap, initCliques);
 2 for clique ∈ initCliques do
 3     if inst.Feature < clique.Feature then
 4         isNBR = CheckNeighbor(inst, clique.Header, nbrHashmap);
 5         if isNBR == True then
 6             for body ∈ clique.Bodies do
 7                 flag =True;
 8                 for instInBody ∈ body do
 9                     isNBR = CheckNeighbor(inst, instInBody, nbrHashmap);
10                     if isNBR == False then
11                         flag =False;
12                         break;
13                 if flag =True then
14                     clique.Bodies.Add(inst, body);
15 return initCliques;
```

By using the two-level filtering mechanism described in Definitions 11 and 12, an instance in a block can be quickly located in clique bodies where it can be placed into to combine larger bodies. The verification space is effectively reduced, and therefore enumerating cliques is accelerated. The pseudocode of the process of updating cliques in a block is described by Algorithm 2 in detail. Feature-level filtering is executed first. The feature type of the new coming instance is compared with each clique in initial cliques (Steps 2 - 3). If the new coming instance passes the feature level filter, it will be entered instance-level filtering by validating if the new coming instance has a neighbor relationship with each clique header (Step 4). If they have a neighbor relationship (Step 5), it means the new coming instance may be placed to clique bodies to form larger clique bodies. Then, Algorithm 2 iterates each instance in the satisfied two-level filter clique bodies to validated if the new coming instance and instances in bodies have a neighbor relationship (Steps 6 - 12). If the new coming instance has a neighbor relationship with all instances in a body, it is placed to the body to build a larger clique body (Steps 13 - 14).

For example, Fig. 8 illustrates the process of Algorithm 2 to obtain overlapping cliques on Block 2. First instance A.3 (Column 1) is considered and after executing the feature level filter, A.3 can be put into cliques B.1 : [[C.3], [D.1], [D.2]] and C.3 : [[D.2]] (Column 2) since the feature type of A.3 is A and it is smaller than clique features B and C. Then, the mining process enters the instance level filter, after checking the neighbor relationship of A.3 with B.1 and C.3, only A.3 and B.1 have a neighbor relationship (Column 3). Next, each instance in bodies of clique header B.1, [[C.3], [D.1], [D.2]], is checked whether having a neighbor relationship with A.3 (Column 4). Only A.3 with D.1 meets the neighbor

Instances	Feature type filter	Neighboring instance filter	Neighboring with bodies	Updated cliques
A.3	B.1 : [[C.3], [D.1], [D.2]] C.3 : [[D.2]]	B.1 : [[C.3], [D.1], [D.2]]	A3C3: False A3D1: True A3D2: False	A : {A.3 : [[B.1], [D.1]], A.4 : [[D.1]]} B : {B.1 : [[C.3], [D.1], [**A.3, D.1**], [D.2]]} C : {C.3 : [[D.2]]}
A.4	B :{B.1:[[C.3], [D.1], [**A.3, D.1**], [D.2]]} C : {C.3 : [[D.2]]}	-	-	A : {A.3 : [[B.1], [D.1]], A.4 : [[D.1]]} B : {B.1 : [[C.3], [D.1], [**A.3, D.1**], [D.2]]} C : {C.3 : [[D.2]]}
B.1	C : {C.3 : [[D.2]]}	C.3 : [[D.2]]	B1D2: True	A : {A.3 : [[B.1], [D.1]], A.4 : [[D.1]]} B : {B.1 : [[C.3], [D.1], [**A.3, D.1**], [D.2]]} C : {C.3 : [[D.2], [**B.1, D.2**]]}
C.3	-	-	-	-
D.1	-	-	-	-
D.2	-	-	-	-

- : no values

Fig. 8 The process of the UpdateClique function on Block 2

Table 2 All overlapping cliques of the data set in Fig. 3

Block No.	Overlapping cliques
Block 1	A:{ A.2:[[C.1], [C.4], [D.3]], A.3:[[B.1], [C.1], [D.1]], A.4:[[B.3], [D.1]]}, B:{ B.1:[[C.1], [A.3, C.1], [D.1], [A.3, D.1], [D.2], [D.3]], B.3:[[D.1], [A.4, D.1]]}, C:{ C.1:[[D.1], [A.3, D.1], [A.3, B.1, D.1], [D.3], [A.2, D.3], [B.1, D.3]]}
Block 2	A:{ A.3:[[B.1], [D.1]], A.4:[[D.1]]}, B:{ B.1:[[C.3], [D.1], [A.3, D.1], [D.2]]}, C:{ C.3:[[D.2], [B.1, D.2]]}
Block 4	A:{ A.1:[[B.4], [C.2]], A.3:[[B.1], [D.1]], A.4:[[D.1]]}, B:{ B.1:[[D.1], [A.3, D.1]], B.4:[[C.2], [A.1, C.2]]}

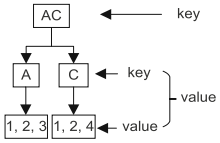
condition, thus they are combined to form clique body [A.3, D.1] (Column 5). Finally, these updated cliques are used as the next iteration step.

As can be seen, by using a feature-level filter and an instance-level filter, it is fast to determine whether an instance can be placed into clique bodies to compose larger overlapping cliques. Table 2 lists all overlapping cliques are discovered on the data set shown in Fig. 3 by our algorithm. After obtaining a set of overlapping cliques, we employ a co-location pattern hash map structure to compress neighbor relationships of instances. Participation indexes of patterns can be quickly calculated based on the structure.

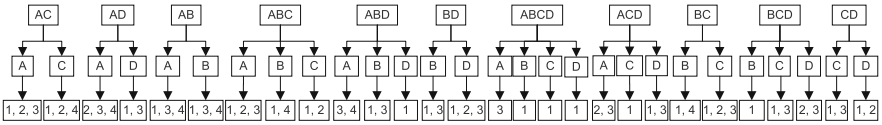
4.1.2 Constructing the co-location pattern hash map structure

In this section, a co-location pattern hash map structure is constructed to gather these overlapping cliques and therefore participation indexes (PIs) of patterns can be directly and quickly calculated based on the hash map structure.

Definition 13 (Co-location pattern hash map structure): The co-location pattern hash map structure is a two-layer hash map structure with a key and a value, which is denoted by



(a) A structure of an element of the co-location pattern hash map structure



(b) The co-location pattern hash map structure of the overlapping cliques of the data set in Fig. 3

Fig. 9 An example of the co-location pattern hash map structure

- The key is a set of feature types of instances in the cliques.
- The value is a hash map structure with its key is each feature type of each instance in overlapping cliques and its value is the instance identification of these instances in the cliques.

For example, Fig. 9a shows a co-location pattern hash map structure that is constructed by overlapping cliques {A.2, C.1}, {A.2, C.4}, {A.3, C.1} and {A.1, C.2} which are discovered from Table 2. All overlapping cliques that are enumerated by Algorithm 1 are plotted in Fig. 9b.

It can be seen that neighboring instances are converted into the co-location pattern hash map structure without missing any instances that have a neighbor relationship. The key of the hash map structure is a co-location pattern and all co-location patterns can be extracted from the keys of the hash map structure. Participation indexes of patterns can be directly calculated from the values of this hash map structure. Algorithm 3 describes the pseudocode of the construction of the co-location pattern hash map structure. This algorithm iterates through cliques (Step 2) and loops each instance in a clique (Step 3) to compose a key (Step 4) and a value (Step 5). After that, the key and the value are put into the co-location pattern hash map structure (Step 6).

Algorithm 3: Constructing the co-location pattern hash map structure

Input: OCs
Output: a co-location pattern hash map structure $CPHashmap$
1 **function** ConstructeCPHashmap(OCs);
2 **for** $clique \in OCs$ **do**
3 **for** $inst \in clique$ **do**
4 $key =$ ComposeKey($clique$);
5 $value =$ ComposeValue($clique$);
6 $CPHashmap$.Update($key, value$);
7 return $CPHashmap$;

4.1.3 Filtering prevalent co-location patterns

Participation indexes of patterns are easily and conveniently computed based on the co-location hash map structure, so when users give/adjust the minimum prevalence threshold, prevalent co-location patterns can be determined quickly. Since the table instances of patterns have already been stored in the co-location hash map structure, our algorithm only needs to re-perform Phases 5 and 6 in the proposed mining framework shown in Fig. 4.

Algorithm 4 shows the pseudocode of calculating participation indexes and filtering prevalent patterns. The algorithm travels each item in the co-location hash map structure (Step 2) to get a pattern which is the key of the item (Step 3) and calculates the participation index of the pattern by passing the value of the item into the CalculatePI function (Step 4). If the PI of the pattern is larger than the minimum prevalence threshold (Step 5), it becomes a prevalent pattern and is added to the mining result (Step 6).

Algorithm 4: Filtering prevalent co-location patterns algorithm

Input: $CPHashmap$, min_prev
Output: A set of all prevalent co-location patterns$PCPs$
1 **function** FilterPrevalentCoLocationPatterns($CPHashmap$,min_{prev});
2 **for** $element \in CPHashmap$ **do**
3 $pattern = CPHashmap$.Key;
4 $PI =$ CalculatePI($CPHashmap$.Value);
5 **if** $PI > min_{prev}$ **then**
6 $PCPs$.Add($pattern, PI$);
7 return $PCPs$;

It can be seen that the proposed algorithm arranges all neighbor relationships of instances into a hash map structure. The hash map makes participation indexes of patterns can be quickly and conveniently computed. When users change the minimum prevalence threshold, instead of re-collecting table instances, our

algorithm only need to implement Algorithm 4 (corresponding to Phases 5 and 6 in the framework described in Fig. 4) for calculating participation indexes and filtering prevalent patterns. Hence, the proposed mining algorithm is robust to minimum prevalence thresholds. In addition, the proposed mining framework has no candidate and it can rapidly compute participation indexes of arbitrary size patterns and measure the prevalence of them.

4.1.4 The time complexity analysis

The computational complexity of the proposed mining framework includes four main parts: posing rectangular grids and detecting redundant blocks, enumerating overlapping cliques (Algorithms 1 and 2), constructing the co-location hash map (Algorithm 3), calculating participation indexes and filtering prevalent patterns (Algorithm 4).

Assume that n is the total number of instances and AR is the area of the spatial data set. The whole area is divided into a grid with a set of cells, and the area of each cell is $d \times d$, where d is the neighbor distance threshold. Therefore, the average number of instances falls into a cell is $n\frac{d^2}{AR}$. To pose the grid, it needs to iterate each instance of the input data set, thus the computational complexity of posing rectangular grids is about $O(n)$. In Algorithm 1, nine cells are group as a block, the average number of instances in a block is about $9n\frac{d^2}{AR}$. Suppose the number of feature types is m. The average number of instances of each feature type is $\frac{n}{m}$ and these instances are divided into the whole area of the spatial data set. Hence, the average number of instances of each feature type that fall in each block is $9\frac{n}{m}\frac{d^2}{AR}$. To detect redundant blocks, instances of a feature type f_i only need to validate with the instances that belong to f_i in the remaining blocks, hence the computational complexity of detecting redundant blocks is about $O(9\frac{n}{m}\frac{d^2}{AR} \times m \times \frac{AR}{9d^2}) = O(n)$, with $\frac{AR}{9d^2}$ is the number of blocks.

To enumerate overlapping cliques in Algorithm 2, each instance in a block is implemented in the two-level filter. In the worst case, all feature types appear in the current block, it means that the number of clique features is the largest. And every instance in the current block has its own neighboring instances, thus each instance is a clique header. Assume t_{avg} and k_{avg} are the average number of bodies led by one instance and the average number of instances in a body, respectively. The upper bound of the computational cost of enumerating overlapping cliques is about $O(9n\frac{d^2}{AR} \times t_{avg} \times k_{avg} \times 9n\frac{d^2}{AR} \times \frac{AR}{9d^2d}) = O(n^2 t_{avg} k_{avg})$.

Overlapping cliques are led by clique header and clique bodies, and the co-location hash map is constructed in Algorithm 3 based on these cliques, thus the time complexity of this part is about $O(9n\frac{d^2}{AR} \times t_{avg} \times k_{avg} \times \frac{AR}{9d^2}) = O(nt_{avg}k_{avg})$.

For Algorithm 4, in the worst-case, each pattern has its own overlapping cliques (a pattern has at least one row instance), it means all size patterns of m feature types are presented. Hence, the upper bound time complexity of Algorithm 4 is $O(\sum_{k=2}^{m} C_m^k)$

.

Combining the above analysis, the final worst computational complexity of the proposed algorithm is about: $O(n + n^2 t_{avg} k_{avg} + n t_{avg} k_{avg} + \sum_{k=2}^{m} C_m^k)$. The number of feature types is normally small, $m \ll n$. Therefore, the largest execution time is devoted to finding cliques and constructing the co-location hash map structure. This point is also supported by the experimental section.

4.2 An improvement algorithm

Based on the above time complexity analysis, we discover that the heaviest work in the basic algorithm described in Section 4.1 is enumerating overlapping cliques shown in Algorithm 1. And the UpdateClique function in Algorithm 1 is the most expensive task. Although an instance can be quickly located to clique bodies where the instance can be fit into to form larger bodies by using the two-level filter mechanism, to check which bodies can be put into, the instance has to verify the neighbor relationship between it with each instance in a body. When all instances of the body have a neighbor relationship with the instance, it can be placed in to form a larger clique.

For example, in Fig. 8, after implementing the two-level filter process, A.3 is confirmed that it can be put into bodies led by clique header B.1, B.1 : [[C.3], [D.1], [D.2]]. Now the basic algorithm needs to check the neighbor relationship between A.3 and all instances in bodies of [C.3], [D.1], and [D.2].

If the number of instances in a clique body is larger, the neighbor relationship verification process can be expensive. Thus, we design a method to accelerate the neighbor relationship validation process.

Definition 14 (Complete star neighborhood): The complete star neighborhood of a spatial instance f_{i_j} is a set of spatial instances which have the neighbor relationship R with it and denoted as

$$CSN(f_{i_j}) = \{f_{k_q} \subset S | R(f_{i_j}, f_{k_q})\} \tag{6}$$

This definition is different from Definition 9, the complete star neighborhood of an instance includes all instances which have the neighbor relationship with the instance. For example, Table 3 lists the complete star neighborhood of the instances in each block shown in Fig. 5.

It is observed that if an instance f_{i_j} which would like to put into a clique body, the instance has a neighbor relationship with all instances in the clique body. It means that these instances are in the star neighborhood of f_{i_j}. In other words, the clique body is a subset of the star neighborhood of f_{i_j}. Hence, to determine whether an instance f_{i_j} can be put into a clique body, we only need to validate the clique body or not a subset of the star neighborhood of f_{i_j}.

As an example, Table 4 describes the process of the neighbor relationship verification by using the complete star neighborhood of A.3. If A.3 would like to put into clique body [C.3], all instances in body [C.3] must be in the complete star

Table 3 The complete star neighborhood of the instances falling in each block in Fig. 5

Block ID	Instances	*CSN*	Block ID	Instances	*CSN*
1	A.2	C.1, C.4, D.3	3	–	–
	A.3	B.1, C.1, D.1	4	A.1	B.4, C.2
	A.4	B.3, D.1		A.3	B.1, D.1
	B.1	A.3, C.1, D.1, D.2, D.3		A.4	D.1
	B.3	A.4, D.1		B.1	A.3, D.1
	C.1	A.2, D.1, D.3		B.4	A.1, C.2
2	A.3	B.1, D.1		C.2	
	A.4	D.1			
	B.1	A.3, C.3, D.1, D.2			
	C.3	B.1, D.2			

– no values

Table 4 The process of neighbor relationship validation by using the complete star neighborhood

Clique body	Complete star neighborhood of A.3	Is subset	New clique
[C.3]	[B.1, C.1, D.1]	No	–
[D.1]	[B.1, C.1, D.1]	Yes	[A.3, D.1]
[D.2]	[B.1, C.1, D.1]	No	–

– no values

neighborhood of A.3. It means that body [C.3] is a subset of the complete star neighborhood of A.3. From Table 3, the complete star neighborhood of A.3 is *CSN*(A.3) = {B.1, C.1, D.1}. It can be seen that [C.3] is not a subset of the complete star neighborhood of A.3, thus A.3 cannot put into body [C.3]. Now, body [D.1] is considered, instances in the body are D.1 and D.1 is in the complete star neighborhood of A.3, thus A.3 can be placed into body [D.1] to form a larger body, [A.3, D.1].

Algorithm 5: Updating overlapping cliques in a block based on complete star neighborhoods

```
   Input: inst, initCliques, CSN : the complete star neighborhood
   Output: updated clique bodies initCliques
1  function UpdateCliquesBasedStarNeighbors(inst, CSN, initCliques);
2  for clique ∈ initCliques do
3      if inst.Feature < clique.Feature then
4          isNBR = CheckNeighbor(inst, clique.Header, nbrHashmap);
5          if isNBR == True then
6              for body ∈ clique.Bodies do
7                  isSubset = body ⋂ CSN(instance);
8                  if isSubset == True then
9                      clique.Bodies.Add(inst, body);
10 return initCliques;
```

To sum up the above description, an advanced algorithm of the basic algorithm is developed and its pseudocode is shown in Algorithm 5. This algorithm has three phases. The first phase and the second phase are feature-level filter and instance-level filter (Steps 2–5), respectively, which are the same as in Algorithm 2. The third phase verifies whether each body in clique bodies is a subset of the complete star neighborhood of the new coming instance (Step 7). If the result of the third phase is true, the new coming instance is placed to the satisfied body to form a larger clique body (Step 9).

5 Experimental results

In this section, we designed a set of experiments to examine the performance of the proposed algorithms. All algorithms were coded and compiled using C++ and executed on a 3.4GHz Intel CPU PC machine with 16GHz main memory.

5.1 Data sets

Both synthetic and real data sets with different distributions are used in our experiments. The details of these data sets are described as follows.

Real data sets: We employ six real data sets to investigate the performance of our algorithms. A summary of these data sets is presented in Table 5. It can be seen that these data sets have different framework area sizes, numbers of instances, numbers of features, and especially their distribution characteristics are distinctive.

Real-1 data set, whose instances form a zonal distribution as shown in Fig. 10a, is a rare plant data set of the Three Parallel Rivers of Yunnan Protected Areas. Three points of interest (POI) data sets of Beijing, Shenzhen, and Shanghai, China, which contain facilities and their locations such as residential area, fast food, and grocery

Table 5 A summary of the real data sets

Name of data set	Space area (m^2)	Number of instances	Number of features	Distribution characteristics
Real-1	72000×123000	336	25	Sparse, zonal
Real-2	138700×230000	106,315	23	Concentrated dense
Real-3	90000×29000	23,867	13	Dense, cluster
Real-4	846000×1152000	97,337	23	Dense, zonal
Real-5	66400×117190	120,355	15	Dense, uniform
Real-6	1699000×14640000	71,851	10	Dense, Sparse

store, are named Real-2, Real-3, and Real-5[1], respectively. Figures 10b, 10c, and 10e show the distributions of the three real data sets. It can be seen that the distribution of the instances in Real-2 is sparse in the suburbs and very dense in the city center, the instances of Real-3 distribute being various clusters, and the instances of Real-5 are densely distributed throughout the space. The Real-4 data set is a California's points of interest data set[2], including churches, flats, parks, etc. As shown in Fig. 10d, the distribution of the instances of this data set is dense and zonal. The final real data set Real-6, which includes supermarkets, banks, cinemas, etc., is a collection of points of interest in the United Kingdom[3]. Figure 10f shows the distribution of the Real-6 data set, it can be seen that it includes both dense and sparse parts.

In the field of co-location pattern mining, when data sets are dense, the number of neighbors of each instance will become larger, the row instances of each pattern will be also more, and it will take more execution time for gathering table instances of patterns. Thus, we carefully chose these data sets with different densities for surveying the performance of the proposed algorithms to prove that our algorithms are more efficient for processing dense data sets.

Synthetic data sets: A similar synthetic data generator which was developed by Shekhar et al. [16] is used to produce synthetic data sets. Table 6 lists the parameters of the synthetic data sets used in different experimental tables/figures.

5.2 Results and analysis

We chose the joinless algorithm [30] to compare with our proposed algorithms. Since (1) it has been proved is an efficient co-location pattern mining algorithm, (2) this algorithm is based on the mining framework shown in Fig. 1 which employs the generate-test candidate model, (3) the star neighborhood is used in it. Besides,

[1] https://figshare.com/articles/dataset/OCSCP/12941714
[2] https://www.cs.utah.edu/~lifeifei/SpatialDataset.htm
[3] https://www.pocketgpsworld.com/

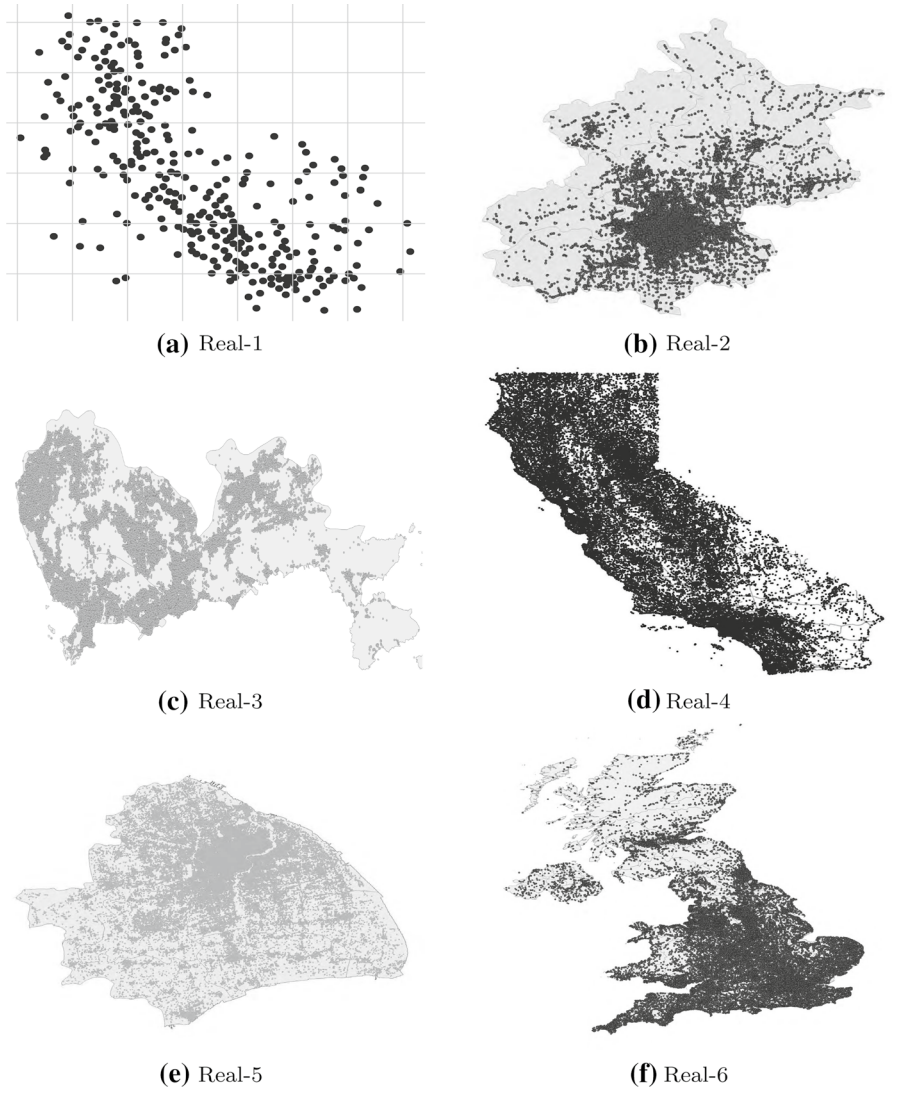


Fig. 10 The distribution of the six real data sets used in our experiments

we also compare with our previous work in [18], in which the OMCP algorithm is proposed.

For the convenience of description, we name the basic algorithm described in Sect. 4.1 as OCSCP-BS and the advanced algorithm represented in Sect. 4.2 as OCSCP-AD. Our experiments are designed including four main parts:

(1) Performance comparison of the algorithms.
(2) Analysis of sensitivity to minimum prevalence thresholds.

Table 6 The summary parameters of synthetic data sets used in different experimental tables/figures

Table / Figure No.	Space area	N. of instances	N. of features	*d*	*min_prev*	Clumpiness
Table 6	1k × 1k	25k, 40k, 55k	17	10	0.2	1
Fig. 11a	1k × 1k	*	17	9	0.2	1
Fig. 11b	10k × 10k	*	17	9	0.2	1
Fig. 11c	10k × 10k	25k	17	10	0.2	*
Fig. 12a	1k × 1k	35k	17	*	0.2	1
Fig. 12b	10k × 10k	35k	17	*	0.2	1
Fig. 13a	1k × 1k	35k	17	10	*	1
Fig. 13a	10k × 10k	35k	17	10	*	1
Fig. 14a	1k × 1k	35k	17	10	0.2	1
Table 8	1k × 1k	30k, 50k, 70k	17	10	0.2	1
Fig. 15a	1k × 1k	*	17	9	0.2	1

* : variables; 1k = 1000

(3) Evaluation of efficiency improvements for discovering high-size patterns.
(4) Assessing the storage space of the proposed algorithms.

5.2.1 Performance comparison of the algorithms

Comparison of computational complexity factors We investigate the computational cost in each phase of the four algorithms as the increase of the density of data sets to prove that our algorithm effectively improves mining performance. Table 7 shows the execution time of each phase in different densities of data sets (the spatial framework size is firm, increase the number of instances). Since the first and second phases in the joinless, OMCP, OCSCP-BS, and OCSCP-AD algorithms are the same, only different phases are listed in Table 7, where *T_gen_candidates*, *T_grid_redundant_blocks*, *T_enum_max_cliques*, *T_table_instances*, *T_enum_over_cliques*, *T_constr_co-loc_hashmap*, *T_call_PI_fi- lter_patterns* are the execution times for generating candidates (the third phase in the framework shown in Fig. 1), posing grids and detecting redundant blocks (a part of the third phase in Fig. 4), enumerating maximal cliques (for OMCP), collecting table instances of candidates (the fourth phase in Fig. 1), enumerating all cliques (the third phase in Fig. 4), constructing the co-location pattern hash map (the fourth phase in Fig. 4), and calculating participation indexes of patterns and filtering prevalent patterns (the fifth and sixth phases in Fig. 1 and Fig. 4), respectively.

As shown in Table 7, in sparse data sets, the performance of these algorithms is not much different. But as the data density increases, OCSCP-BS and OCSCP-AD show better performance. The most execution time in the joinless algorithm is devoted to collect table instances of candidates. This is in line with the shortcomings of the algorithms based on the generate-test candidate model mentioned in the introduction section. Enumerating maximal cliques and overlapping cliques are the heaviest job in the OMCP and the proposed algorithms, respectively. However, finding cliques in OCSCP-BS and OCSCP-AD takes less computation time than

Table 7 The execution time of each phase of the algorithms

Algorithm	Joinless			OMCP			OCSCP-BS			OCSCP-AD		
Factor(sec.)	# instances											
	25000	40000	55000	25000	40000	55000	25000	40000	55000	25000	40000	55000
T_gen_candidates	0.006	0.021	0.046	–	–	–	–	–	–	–	–	–
T_grid_redundant_blocks	–	–	–	0.175	0.201	0.261	0.166	0.217	0.248	0.178	0.217	0.262
T_enum_max_cliques	–	–	–	26.96	180.89	609.897	–	–	–	–	–	–
T_table_instances	31.121	216.262	1398.101	19.623	94.794	279.908	–	–	–	–	–	–
T_enum_over_cliques	–	–	–	–	–	–	14.59	85.8	236.514	10.121	56.645	156.443
T_co–loc_hashmap	–	–	–	–	–	–	8.797	37.42	110.861	8.686	37.669	111.703
T_call_PI_filter_pattern	0.295	1.265	4.101	0.023	0.032	0.033	0.043	0.057	0.062	0.038	0.056	0.057
T_total	31.422	217.548	1402.25	46.781	275.917	890.099	23.596	123.494	347.685	19.023	94.587	268.465
Memory space (MB)	272.207	1470.996	3574.656	522.215	3927.770	4021.660	592.988	3928.891	4018.887	593.781	3933.672	4017.031

– No values

[#]Instances : the number of instances

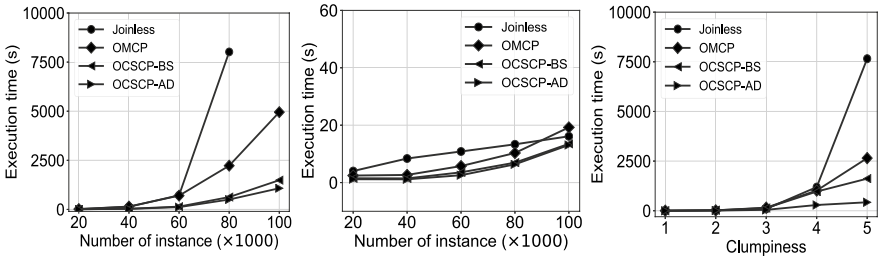


Fig. 11 The execution times of three algorithms in different data set densities

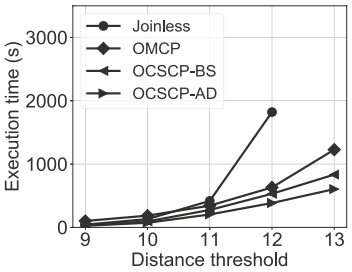
enumerating maximal cliques in OMCP. Since in the newly proposed algorithms, one instance can be quickly located into cliques to form larger cliques without performing unnecessary verification like in the OMCP algorithm. At the same time, OCSCP-BS and OCSCP-AD directly generate table instances of patterns after finding cliques. They are not like OMCP to enumerate maximal cliques before collecting table instances. Hence, OCSCP-BS and OCSCP-AD show better performance.

The space cost of each algorithm is also listed in Table 7. Under the same distance threshold, the denser data sets, the larger the number of neighboring instances of an instance, the bigger the table instance supporting the pattern, hence these four algorithms require more and more storage space. The joinless algorithm needs to hold the table instances of all size k patterns to construct the table instances of size $(k + 1)$ patterns at the same time, the table instances of patterns that their size is smaller than k can be released. While OMCP, OCSCP-BS, and OCSCP-AD keep the table instances of all size patterns, therefore they require more storage space than the joinless algorithm.

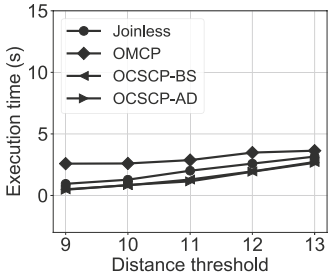
Overall, the newly proposed algorithms OCSCP-BS and OCSCP-AD significantly improve the mining performance, especially for dense data.

Comparison of data set density We evaluate the performance of algorithms when increasing data set density. By fixing the spatial framework size, the amount of instances is continuously increased, thereby increasing the data density. Figures 11a and 11b show the execution time of the four algorithms on dense and sparse data sets with different numbers of instances. The four algorithms have similar the execution time in the sparse data sets. However, with the increase in the density of dense data sets, OCSCP-BS and OCSCP-AD shows more superior performance.

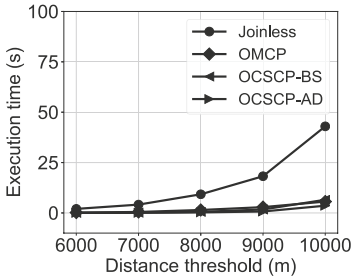
Further examining the performance of the proposed algorithms, we observe the mining effect of the algorithms in different clumpiness values. Clumpiness represents the number of co-location instances generated in the same neighborhood areas over total instances. The higher clumpiness is, the larger table instances are obtained. Figure 11c summarizes the examination results. With the increase of clumpiness, the execution times of the joinless, OMCP, OCSCP-BS, and OCSCP-AD algorithms also increase accordingly. However, the newly proposed algorithms increase more slowly.



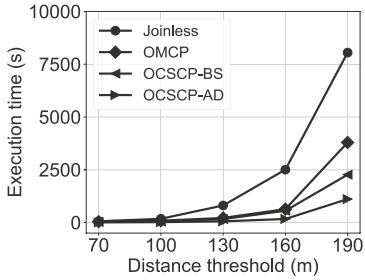
(a) Dense synthetic data($min\_prev = 0.2$)



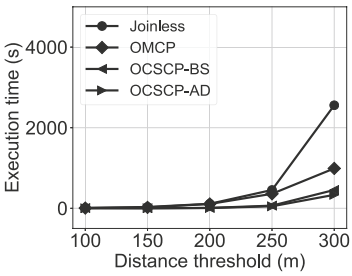
(b) Sparse synthetic data($min\_prev = 0.2$)



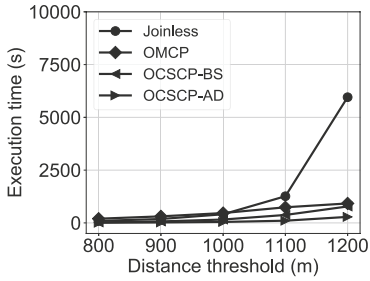
(c) Real-1 ($min\_prev = 0.2$)



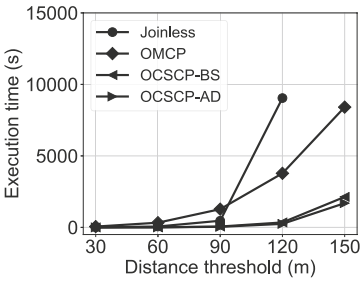
(d) Real-2 ($min\_prev = 0.1$)



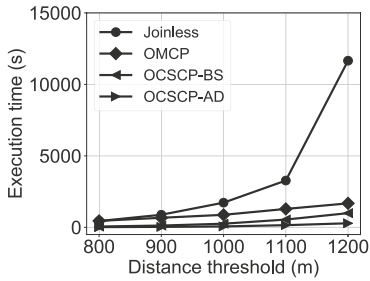
(e) Real-3 ($min\_prev = 0.1$)



(f) Real-4 ($min\_prev = 0.1$)



(g) Real-5 ($min\_prev = 0.2$)



(h) Real-6 ($min\_prev = 0.2$)

Fig. 12 The execution times of the compared algorithms with changes of the distance threshold

Comparison of distance thresholds. Figure 12 charts the changes in the execution times with changes in the distance threshold. As the distance threshold increases, the computational cost of the four algorithms also increases. Since the larger the distance threshold, the more neighborhoods and row instances, the joinless algorithm needs more execution time to collect table instances of candidates, particularly when data sets are dense, in a large value of distance threshold, it has not completed because it takes too much execution time (Fig. 12a, g). While the two newly proposed algorithms take less execution time in dense data sets than joinless and OMCP, and the OCSCP-AD algorithm has the lowest execution time in most cases.

5.2.2 Analysis of sensitivity to the minimum prevalence threshold

In this section, the sensitivity to the minimum prevalence threshold is evaluated. Figure 13 shows the evaluation results. The execution time of the joinless algorithm reduces with the increase of the minimum prevalence threshold. Since in the larger value of the minimum prevalence threshold, the number of size k prevalent patterns is smaller, the number of size $(k + 1)$ candidates generated is also smaller. However, the minimum prevalence threshold is not suitable for too large, the patterns that users may be interested in may be lost.

While the execution time of the proposed algorithms is constant because when users change the minimum prevalence threshold, our algorithms only need to perform the last two phases of the framework shown in Fig. 4, the proportion of the execution time of these two phases is very small in the total execution time (it has been proven in Table 7). Besides, OCSCP-AD and OCSCP-BS are more efficient than OMCP. Since the two algorithms find cliques instead of maximal cliques and they directly generate all table instances

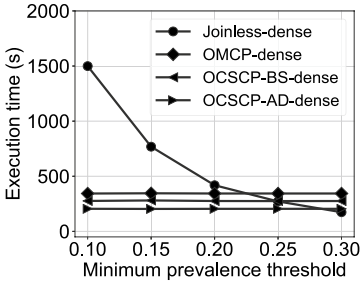
of patterns, while in OMCP, after obtaining maximal cliques, it has to runs through the maximal cliques in every possible combination to generate row instances.

Besides, for a smaller minimum prevalence threshold, the joinless algorithm that is based on the generate-test candidate framework takes more execution time. Since the numbers of size k prevalent patterns and size $(k + 1)$ candidates are bigger, thus it needs more execution time to collect table instances of the candidates. In all cases of the dense data sets (Fig. 13a–f), the computational cost of the joinless algorithm is very expensive.

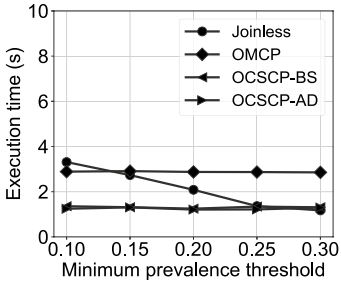
To sum up, the proposed algorithms are insensitive to the minimum prevalence threshold.

5.2.3 The efficiency of discovering high-size patterns

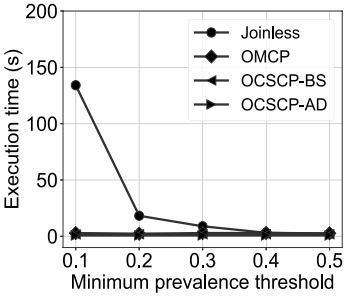
In this experiment, we show the efficiency improvements for mining certain size k patterns or high-size patterns. This is very practical because sometimes users only care about certain size k patterns and the question is how to quickly respond to their needs. Figure 14 shows the execution times of the four algorithms when they discover size 3, 4, 5, and all size $(k > 3)$ patterns on both synthetic and real data sets.



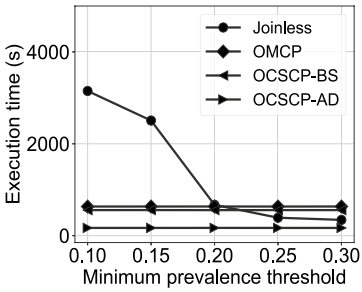
(a) Dense synthetic data($d = 10$)



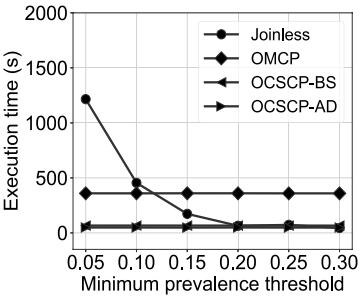
(b) Sparse synthetic data($d = 10$)



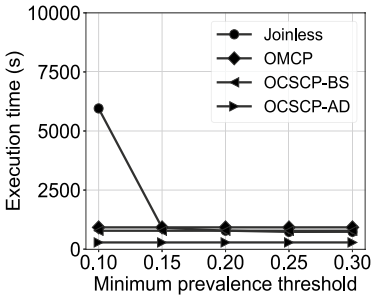
(c) Real-1 ($d = 900$m)



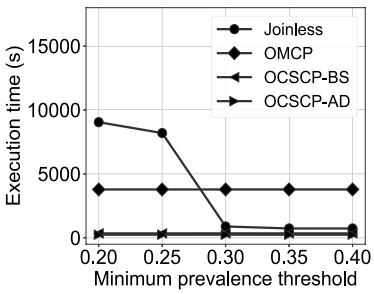
(d) Real-2 ($d = 160$m)



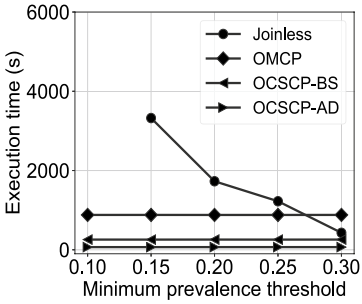
(e) Real-3 ($d = 250$m)



(f) Real-4 ($d = 1200$mm)

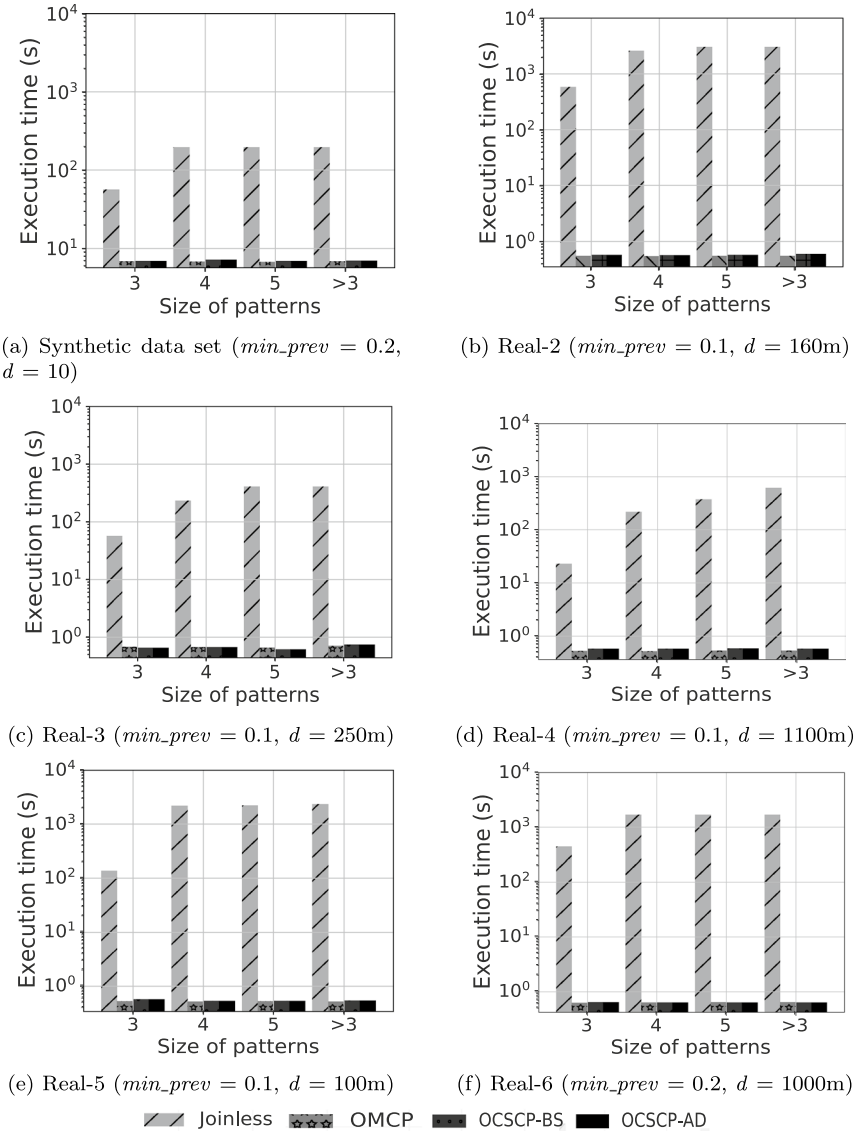


(g) Real-5 ($d = 120$m)



(h) Real-6 ($d = 1000$m)

Fig. 13 The execution times of compared algorithms with changes in the minimum prevalence threshold



(a) Synthetic data set ($min\_prev = 0.2$, $d = 10$)

(b) Real-2 ($min\_prev = 0.1$, $d = 160$m)

(c) Real-3 ($min\_prev = 0.1$, $d = 250$m)

(d) Real-4 ($min\_prev = 0.1$, $d = 1100$m)

(e) Real-5 ($min\_prev = 0.1$, $d = 100$m)

(f) Real-6 ($min\_prev = 0.2$, $d = 1000$m)

Fig. 14 The execution times when mining high-size co-location patterns of the compared algorithms in different data sets

The result shown in Fig. 14 demonstrates that our algorithms significantly improve mining performance. OCSCP-AD and OCSCP-BS can swiftly give the mining results. Since in our algorithms, the materializing neighbor relationships, enumerating overlapping cliques, and constructing the co-location pattern hash map structure phases are performed only once and the structure is stored in a JSON file. When users want to mine certain size k patterns, the proposed algorithms only need

Table 8 The required storage space in each phase of the OMCP, OCSCP-BS, and OCSCP-AD algorithms

Algorithm	OMCP			OCSCP-BS			OCSCP-AD		
F. (MB)	# inst.								
	30000	50000	70000	30000	50000	70000	30000	50000	70000
Maximal cliques	24.238	44.309	82.512	–	–	–	–	–	–
Cliques	–	–	–	137.664	300.082	803.289	137.238	300.125	796.625
Table instances	439.867	956.414	3146.113	–	–	–	–	–	–
CPHashmap	–	–	–	474.840	818.430	3151.855	474.746	817.832	3151.984

F.: Factor, # inst. : the number of instances, – no values

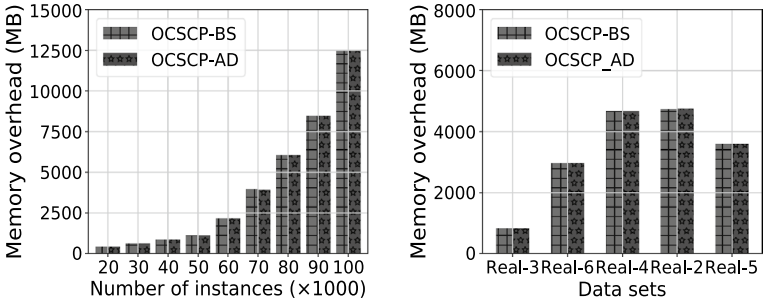
to load the structure, calculate participation indexes and filter patterns with their size equal to k and their participation index is larger than the minimum prevalence threshold. Thus, the proposed algorithms can rapidly respond to requirements of users.

As can be seen from Fig. 14, our previous algorithm OMCP also can quickly mine user-specified size patterns. Because like OCSCP-AD and OCSCP-BS algorithms, under a given distance threshold, table instances of patterns have been stored already. When users need to mine a certain size of patterns, it just searches for the satisfied patterns from the stored table instances.

Whereas, based on the generate-test mining framework, the joinless algorithm is an incremental mining approach. The mining process of size $(k + 1)$ patterns can only be performed after finishing mining size k patterns. If users want to mine a certain size pattern, all patterns smaller than the certain size pattern have to be discovered first, thus it will take more execution time.

5.2.4 The storage space

In the final experiment, we survey the storage space of the proposed algorithms. In our algorithms, there are two main parts occupying storage space, one is storing the two-layer clique structure found in each block, and the other is the co-location hash map structure that holds all row instances. Table 8 lists the change of the storage space of the two parts with increasing numbers of instances of input data sets. Besides, the storage space of the OMCP algorithm is also plotted. It can be seen that storing maximal cliques in OMCP takes less storage space than OCSCP-BS and OCSCP-AD because the newly proposed algorithms have to store all cliques, the number of the cliques is larger than the number of maximal cliques. Furthermore, in the two final phases (phases 5 and 6 in Fig. 4), the storage space of OMCP is slightly less than OCSCP-BS and OCSCP-AD since the former employs a one-layer hash map structure (with its key is a pattern and its value is a list storing the table instance of the pattern) to save these table instances, while the latter uses a two-layer hash map structure.



(a) Synthetic data sets **(b)** Real data sets

Fig. 15 The requirement of the storage space of the proposed algorithms with the change in the numbers of instances

In furthermore, the change in the storage space of the proposed algorithm with increases in the number of instances is also investigated. In this experiment, the distance and minimum prevalence thresholds are set to a suitable value to ensure that at least size 5 patterns are discovered. Figure 15a plots the required storage space of the proposed algorithms on synthetic data sets when instances are increased. As can be seen that the larger the number of instances, the more the storage space is required. Since when the framework area of data sets and the distance threshold are fixed, increasing in the number of instances, more and more neighboring instances are formed. This means that the number of row instances (cliques) also becomes larger, thus the proposed algorithms need more storage space to save these cliques.

The storage space required by OCSCP-BS and OCSCP-AD when they execute on the real data sets is also shown in Fig. 15b. These data sets are sorted by their number of instances. The distance threshold is set to 250m, 1000m, 1100m, 160m, and 120m for Real-3, Real-6, Real-4, Real-2, and Real-5, respectively, and the minimum prevalence threshold is set to 0.1 for all data sets. Overall, the storage space required by the proposed algorithms increases with the number of instances in data sets. However, the storage space also depends on data distribution. For example, although the number of instances of Real-5 is larger than Real-2 and Real-4, Real-5 requires less storage space than Real-2 and Real-4. Since the distributions of Real-2 and Real-4 are concentrated dense and (dense + zonal) as shown in Fig. 10. Although dataset 1 is also dense, instances are evenly distributed throughout the space. The data density (number of instances in an area unit) of Real-2 and Real-4 is higher than Real-3.

In conclusion, on our experimental hardware platform, we can process dense data up to about 120.000 instances. If the hardware resources are expanded, our algorithm can quickly deal with larger data.

6 Conclusions and future work

In this paper, a spatial co-location pattern mining method which is not sensitive to the prevalence threshold is studied. We design a two-level filter mechanism to discover all overlapping cliques and store them in a co-location pattern hash map structure. Participation indexes are calculated and prevalent patterns are filtered quickly from this hash map structure. We evaluate the proposed algorithms on different distributions of both synthetic and real data sets. The evaluation results demonstrate that our methods significantly improve mining performance, especially when dealing with dense data sets. The proposed method can also rapidly respond to user requirements when the distance and prevalence thresholds are adjusted to fit their application. Moreover, our method shows better performance when directly discovering high-size co-location patterns.

However, the proposed algorithms need to store the whole table instances of all patterns in a co-location hash map structure, thus they need a large storage space. In further work, we focus on compress the structure to handle larger data sets.

Acknowledgements This work is supported by the National Natural Science Foundation of China (61966036, 61662086) and the Project of Innovative Research Team of Yunnan Province(2018HC019).

References

1. Akbari, M., Samadzadegan, F., Weibel, R.: A generic regional spatio-temporal co-occurrence pattern mining model: a case study for air pollution. Journal of Geographical Systems **17**(3), 249–274 (2015). https://doi.org/10.1007/s10109-015-0216-4
2. Al-Naymat, G.: Enumeration of maximal clique for mining spatial co-location patterns. In: 2008 IEEE/ACS International Conference on Computer Systems and Applications, IEEE, pp 126–133 (2008)
3. An, S., Yang, H., Wang, J., Cui, N., Cui, J.: Mining urban recurrent congestion evolution patterns from gps-equipped vehicle mobility data. Information Sciences **373**, 515–526 (2016)
4. Andrzejewski, W., Boinski, P.: Efficient spatial co-location pattern mining on multiple gpus. Expert Systems with Applications **93**, 465–483 (2018)
5. Boinski, P., Zakrzewicz, M.: Algorithms for spatial collocation pattern mining in a limited memory environment: a summary of results. Journal of Intelligent Information Systems **43**(1), 147–182 (2014)
6. Chang, X., Ma, Z., Lin, M., Yang, Y., Hauptmann, A.G.: Feature interaction augmented sparse learning for fast kinect motion detection. IEEE transactions on image processing **26**(8), 3911–3920 (2017)
7. Cheng, J., Zhu, L., Ke, Y., Chu, S.: Fast algorithms for maximal clique enumeration with limited memory. In: Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining, pp 1240–1248 (2012)
8. Huang, Y., Pei, J., Xiong, H.: Mining co-location patterns with rare events from spatial data sets. Geoinformatica **10**(3), 239–260 (2006)
9. Kim, S.K., Kim, Y., Kim, U.: Maximal cliques generating algorithm for spatial co-location pattern mining. In: FTRA International Conference on Secure and Trust Computing, Data Management, and Application, Springer, pp 241–250 (2011)
10. Leibovici, D.G., Claramunt, C., Le Guyader, D., Brosset, D.: Local and global spatio-temporal entropy indices based on distance-ratios and co-occurrences distributions. International Journal of Geographical Information Science **28**(5), 1061–1084 (2014)
11. Li, J., Zhang, W., Yu, J., Chen, H.: Industrial spatial agglomeration using distance-based approach in beijing, china. Chinese Geographical Science **25**(6), 698–712 (2015)

12. Li, J., Adilmagambetov, A., Jabbar, M.S.M., Zaïane, O.R., Osornio-Vargas, A., Wine, O.: On discovering co-location patterns in datasets: a case study of pollutants and child cancers. GeoInformatica **20**(4), 651–692 (2016)
13. Ouyang, Z., Wang, L., Wu, P.: Spatial co-location pattern discovery from fuzzy objects. International Journal on Artificial Intelligence Tools **26**(02), 1750003 (2017)
14. Qian, F., He, Q., Chiew, K., He, J.: Spatial co-location pattern discovery without thresholds. Knowledge and Information Systems **33**(2), 419–445 (2012)
15. Sainju, A.M., Aghajarian, D., Jiang, Z., Prasad, S.K.: Parallel grid-based colocation mining algorithms on gpus for big spatial event data. IEEE Transactions on Big Data (2018)
16. Shekhar, S., Huang, Y.: Discovering spatial co-location patterns: A summary of results. In: International symposium on spatial and temporal databases, Springer, pp 236–256 (2001)
17. Sierra, R., Stephens, C.R.: Exploratory analysis of the interrelations between co-located boolean spatial features using network graphs. International Journal of Geographical Information Science **26**(3), 441–468 (2012)
18. Tran, V., Wang, L., Zhou, L.: Mining spatial co-location patterns based on overlap maximal clique partitioning. In: 2019 20th IEEE International Conference on Mobile Data Management (MDM), IEEE, pp 467–472 (2019)
19. Verhein, F., Al-Naymat, G.: Fast mining of complex spatial co-location patterns using glimit. In: Seventh IEEE International Conference on Data Mining Workshops (ICDMW 2007), IEEE, pp 679–684 (2007)
20. Wang, L., Bao, Y., Lu, J., Yip, J.: A new join-less approach for co-location pattern mining. In: 2008 8th IEEE International Conference on Computer and Information Technology, IEEE, pp 197–202 (2008)
21. Wang, L., Bao, Y., Lu, Z.: Efficient discovery of spatial co-location patterns using the icpi-tree. The Open Information Systems Journal **3**(1), (2009)
22. Wang, L., Zhou, L., Lu, J., Yip, J.: An order-clique-based approach for mining maximal co-locations. Information Sciences **179**(19), 3370–3382 (2009)
23. Wang, L., Chen, H., Zhao, L., Zhou, L.: Efficiently mining co-location rules on interval data. In: International Conference on Advanced Data Mining and Applications, Springer, pp 477–488 (2010)
24. Wang, L., Wu, P., Chen, H.: Finding probabilistic prevalent colocations in spatially uncertain data sets. IEEE Transactions on Knowledge and Data Engineering **25**(4), 790–804 (2011)
25. Wang, L., Bao, X., Zhou, L.: Redundancy reduction for prevalent co-location patterns. IEEE Transactions on Knowledge and Data Engineering **30**(1), 142–155 (2017)
26. Wang, L., Bao, X., Chen, H., Cao, L.: Effective lossless condensed representation and discovery of spatial co-location patterns. Information Sciences **436**, 197–213 (2018)
27. Yao, X., Peng, L., Yang, L., Chi, T.: A fast space-saving algorithm for maximal co-location pattern mining. Expert Systems with Applications **63**, 310–323 (2016)
28. Yoo, J.S., Bow, M.: Mining top-k closed co-location patterns. In: Proceedings 2011 IEEE International Conference on Spatial Data Mining and Geographical Knowledge Services, IEEE, pp 100–105 (2011)
29. Yoo, J.S., Bow, M.: A framework for generating condensed co-location sets from spatial databases. Intelligent Data Analysis **23**(2), 333–355 (2019)
30. Yoo, J.S., Shekhar, S.: A joinless approach for mining spatial colocation patterns. IEEE Transactions on Knowledge and Data Engineering **18**(10), 1323–1337 (2006)
31. Yoo, J.S., Shekhar, S., Smith, J., Kumquat, J.P.: A partial join approach for mining co-location patterns. In: Proceedings of the 12th annual ACM international workshop on Geographic information systems, pp 241–249 (2004)
32. Yu, W.: Spatial co-location pattern mining for location-based services in road networks. Expert Systems with Applications **46**, 324–335 (2016)
33. Zaki, M.J., Ogihara, M.: Theoretical foundations of association rules. In: 3rd ACM SIGMOD workshop, pp 71–78 (1998)

Authors and Affiliations

Vanha Tran[1] · Lizhen Wang[1] · Lihua Zhou[1]

Vanha Tran
tranvanha@mail.ynu.edu.cn

Lihua Zhou
lhzhou@ynu.edu.cn

[1] School of Information Science and Engineering, Yunnan University, Dongwaihuan South Road, Kunming, Yunnan 650091, China