Information Sciences 490 (2019) 244–264

Contents lists available at ScienceDirect

Information Sciences

journal homepage: www.elsevier.com/locate/ins

A clique-based approach for co-location pattern mining

Xuguang Bao, Lizhen Wang*

Department of Computer Science and Engineering, School of Information Science and Engineering, Yunnan University, Kunming 650091, China

ARTICLE INFO

Article history:
Received 26 July 2018
Revised 26 March 2019
Accepted 27 March 2019
Available online 29 March 2019

Keywords:
Spatial data mining
Co-location pattern
Spatial neighbor relationship
Row-instance
Clique-based approach

ABSTRACT

Co-location pattern mining refers to the task of discovering the group of features (geographic object types) whose instances (geographic objects) are frequently located close together in a geometric space. Current approaches on this topic adopt a prevalence threshold (a measure of a user's interest in a pattern) to generate prevalent co-location patterns. However, in practice, it is not easy to specify a suitable prevalence threshold. Thus, users have to repeatedly execute the program to find a suitable prevalence threshold. Besides, the efficiency of these approaches is limited because of the expensive cost of identifying row-instances of co-location patterns. In this paper, we propose a novel clique-based approach for discovering complete and correct prevalent co-location patterns. The proposed approach avoids identifying row-instances of co-location patterns thus making it much easier to find a proper prevalence threshold. First, two efficient schemas are designed to generate complete and correct cliques. Next, these cliques are transformed into a hash structure which is independent of the prevalence threshold. Finally, the prevalence of each co-location pattern is efficiently calculated using the hash structure. The experiments on both real and synthetic datasets show the efficiency and effectiveness of our proposed approaches.

1. Introduction

Co-location pattern mining is a new branch in spatial data mining recently. A prevalent co-location pattern represents a subset of spatial features whose instances are frequently located together in a spatial neighborhood. A set {hospital, pharmacy} may be an example of a prevalent co-location pattern because hospitals and pharmacies are frequently located near each other. Prevalent co-location patterns can be used in the decision making of various domains (e.g., earth science, public health, biology, transportation). For example, a mobile service provider may be interested in mobile service patterns frequently requested by neighboring geographical users. Botanists may be interested in symbiotic plant species in an area [16].

Discovering prevalent co-location patterns is an important task in spatial data mining and many co-location pattern mining approaches have been proposed. These approaches can be classified into two groups. The first group [16,19,23,30,31] exploits an Apriori-like [1] way to generate prevalent co-location patterns, but this way needs considerable time to check the co-location row-instances. Besides, the size-wised manner makes approaches in this group inefficient in getting longer-size co-location patterns. The second group [2,17,18,20,22] suggests using maximal cliques [6,7,11,13,21] generated from spatial datasets to discover prevalent co-location patterns, since instances in a row-instance of a co-location pattern form a clique

* Corresponding author.
E-mail address: lzhwang@ynu.edu.cn (L. Wang).

https://doi.org/10.1016/j.ins.2019.03.072
0020-0255/

relationship. To generate prevalent co-location patterns, they first transform maximal cliques into transaction-type data, and then use association analysis methods (e.g., Apriori [1], FP-Growth [15]) to generate co-location patterns. However, large amounts of computing time are required because the problem of extracting maximal cliques from a graph is known to be NP-Hard. Besides, association analysis methods may be inefficient with large amounts of transaction-type data. Thus, an effective way is needed to make co-location pattern mining effective and flexible.

In current works for prevalent co-location pattern mining, the prevalence threshold is commonly a subjective value which determines whether a co-location pattern is prevalent (interesting) or not. However, a lower measure value may generate numerous redundant (known) co-location patterns while a higher measure value may lose some rare co-location patterns [14]. Additionally, the interestingness of a co-location pattern is not absolute. One user may be interested in a co-location pattern while another may not, and different users may prefer different prevalence thresholds. As users using an Apriori-like approach must restart their mining task for each different prevalence threshold, it may take a long time to find an appropriate measure value. Thus, an effective way to find a proper prevalence threshold without restart is urgently needed.

Many extended works [4,5,24–29,32] on mining different kinds of co-location patterns have been proposed and most of these extended works adopt prevalent co-location pattern mining algorithms (e.g., join-based [16], join-less [31]) in their approaches. If there exists a novel prevalent co-location pattern mining approach more efficient and flexible than traditional common works in certain situations, the extended works can also be made more effective and efficient.

Motivated by the above considerations, this paper makes four contributions to the field of the spatial co-location mining:

1. Two schemas using a tree structure are proposed to generate cliques that exist in a spatial dataset. Most clique generation approaches adopt a tree structure [1,2,17] mainly because it is a compressed structure for storage and various effective pruning strategies can be performed effectively. We prove that the cliques generated by the two schemas are complete and correct.
2. Unlike other approaches which transform cliques as transaction-type data for discovering prevalent co-location patterns, the cliques are transformed into a hash structure in our approach. The hash structure is independent of the prevalence threshold and once the hash structure is constructed, changes of prevalence threshold do not cause a complete restart. Thus, our approach effectively assists the user in finding a proper prevalence threshold.
3. Using the hash structure, we have designed an efficient algorithm for discovering prevalent co-location patterns instead of the traditional association analysis methods. This algorithm avoids the expensive operations of identifying row-instances of co-location patterns, commonly used in traditional approaches to co-location pattern mining. We prove that the prevalent co-location patterns generated by our algorithm are complete and correct.
4. Experimental evaluations on both real and synthetic datasets were designed to show the effectiveness and efficiency of our proposed algorithms.

The rest of this paper is organized as follows. Section 2 first gives an overview of the basic concepts on co-location pattern mining and the problem definition, and then discusses related works and frameworks for co-location pattern mining. A clique-based approach for co-location pattern mining is developed in Section 3. Section 4 illustrates the experimental setup and discusses the results. Section 5 concludes this paper.

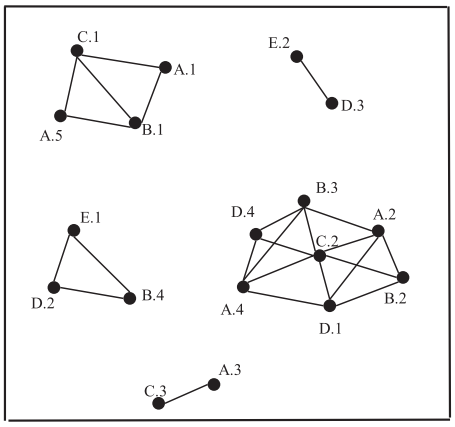
2. Co-location pattern mining

2.1. Basic concepts

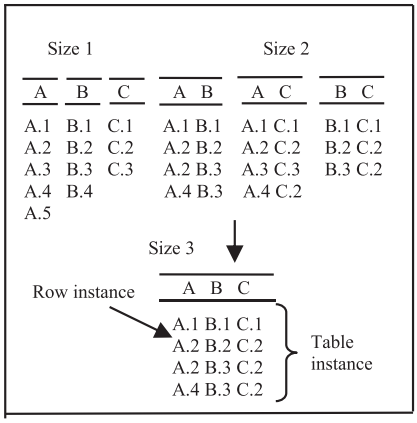
A **feature** describes a geographic object type with a name (e.g., Carrefour group) in a geometric space. An **instance** is a physical geographic object (e.g., Carrefour No.2 branch in Beijing) of a particular feature with locations. A **spatial dataset** is a set of instances with locations and types (features) in a geometric space. Suppose F is a set of features, S is a set of their instances and R is a **spatial neighbor relationship** over S. If the Euclidean metric is used as the spatial neighbor relationship R, two instances s and s' satisfy the spatial neighbor relationship R if the Euclidean distance between them is no more than a given threshold min_*dist*, i.e., $R(s, s') = \text{true}$. A **clique** *cl* is a set of instances such that every two distinct instances in this set are neighbors. A **maximal clique** is a clique that is not a subset of any other cliques. The *size* of a clique is the number of instances in it. A **co-location (pattern)** c ($c \subseteq F$) is a subset of features whose instances form cliques frequently under the spatial neighbor relationship R. The *size* of c is the number of features in c. A **co-location instance** I ($I \subseteq S$, also called **row-instance**) of a co-location c is a set of instances containing all instances of features in c and forming a clique under R. The **table instance** of c is the collection of all the row-instances of c, denoted as $T(c)$.

In co-location pattern mining, **participation index (PI)** is commonly used as the measure of the prevalence of a co-location. The PI value of a co-location c is defined as $\text{PI}(c) = \min_{f_i \in c}\{\text{PR}(c, f_i)\}$, where $\text{PR}(c, f_i)$ is the **participation ratio (PR)** of the feature f_i in a k-size co-location $c = \{f_1, f_2, ..., f_k\}$, that is the fraction of instances of f_i that occur in $\text{T}(c)$, i.e., $\text{PR}(c, f_i) = \frac{|set of\ instances\ of\ f_i\ in\ T(c)|}{|set\ of\ instances\ of\ f_i|}$. A PI measure indicates that wherever a feature in c is observed, with a probability of at least $\text{PI}(c)$, all other features in c can be observed in its neighborhood. A co-location c is a **prevalent co-location (pattern)** if $\text{PI}(c) \geq$ min_*prev* holds, where min_*prev* is a user-specified minimum prevalence threshold.

Example 1. Fig. 1(a) shows an example of a spatial dataset which has five features named A, B, C, D and E, and their instances. A.1 stands for the first instance of feature A, and the number 1 is the *index* of this instance in feature A. The two



(a) An example graph



(b) Table instances

Instance	Neighbor instances
A.1	A.1,B.1,C.1
A.2	A.2,B.2,B.3,C.2,D.1
A.3	A.3,C.3
A.4	A.4,B.3,C.2,D.1,D.4
A.5	A.5,B.1,C.1
B.1	B.1,C.1
B.2	B.2,C.2,D.1
B.3	B.3,C.2,D.4
B.4	B.4,D.2,E.1
C.1	C.1
C.2	C.2,D.1,D.4
C.3	C.3
D.1	D.1
D.2	D.2,E.1
D.3	D.3,E.2
D.4	D.4
E.1	E.1
E.2	E.2

(c) Star neighborhood list

Fig. 1. An example graph with its table instances and star neighborhood list.

instances that are neighbors are connected by a line in Fig. 1(a). {A.1, B.1, C.1} is a maximal clique and is a row-instance of a 3-size co-location {A, B, C}. The table instance of {A, B, C} is {{A.1, B.1, C.1}, {A.2, B.2, C.2}, {A.2, B.3, C.2}, {A.4, B.3, C.2}}. Fig. 1(b) shows some co-location patterns and their row-instances (table instances) of the example spatial dataset shown in Fig. 1(a), PR{{A, B, C}, A} = |{A.1, A.2, A.4}|/ |{A.1, A.2, A3, A.4, A.5}| = 3/5, PR{{A, B, C}, B} = 3/4, PR{{A, B, C}, C} = 2/3, thus PI({A, B, C}) = min{PR{{A, B, C}, A}, PR{{A, B, C}, B}, PR{{A, B, C}, C}} = 3/5. If min_*prev* is set as 0.5, {A, B, C} is a prevalent co-location pattern.

2.2. Problem definition

Our approach aims at utilizing cliques to generate correct and complete prevalent co-locations whilst reducing the cost of computation and discovery of a suitable prevalence threshold. The formal problem definition is as follows.

Inputs:
1) A set of features $F = \{f_1, f_2, ..., f_n\}$
2) A set of instances $S = \{S_1 \cup S_2 \cup ... \cup S_n\}$, where S_i $(1 \le i \le n)$ is a set of instances of feature f_i. Each instance $s \in S_i$ has vector information of <instance id j, feature f_i, location<x, y>>, where $1 \le j \le |S_i|$.
3) A spatial neighbor relationship R.
4) A minimum prevalence threshold (min_*prev*).
Output:
A correct and complete set of prevalent co-locations enabling the user to discover a suitable prevalence threshold, whilst also reducing the cost of computation.
Constraints:
R is a distance metric-based spatial neighbor relationship with a symmetric property. A distance threshold (min_*dist*) parameter is required to check the spatial neighbor relationship between two instances, i.e., given two instances s_i and s_j with different features, if the Euclidean distance between them is no more than min_*dist*, they are neighbors (i.e., $R(s_i, s_j)$ = true).

2.3. Related work

Huang et al. [16] first defined prevalent co-location patterns and developed an Apriori-like algorithm, called join-based approach, for mining prevalent co-location patterns. This approach works well on sparse spatial databases. However, expensive join operations to identify co-location row-instances of candidate co-location patterns make it inefficient when dealing with dense spatial databases. To improve the efficiency of join-based approach, Yoo et al. proposed two algorithms — called partial-join approach [30] and join-less approach [31], respectively — to overcome the efficiency disadvantage of join-based approach. Partial-join approach uses a transaction-based Apriori algorithm as a building block and adopts the instance join method for residual instances not identified in transactions. Join-less approach puts the spatial neighbor relationships into a compressed star neighborhood list storing the star neighborhood of each instance from a spatial dataset (shown in Fig. 1(c)), and then uses an instance-lookup way instead of the expensive join operation. Although join-less performs better than join-based approach in any situation and partial-join approach in most situations, it is still costly, especially in dense spatial datasets. Moreover, join-less is inefficient in finding a proper prevalence threshold because any change of prevalence threshold causes a restart.

Maximal clique finding [21] is a classical issue in the field of graph theory that has been investigated for decades and has several variations [6]. The first algorithm utilizing a depth-first search strategy for generating maximal cliques has been proposed by Bron and Kerbosch [7]. Since then, numerous algorithms have been proposed to improve the Bron-Kebosch

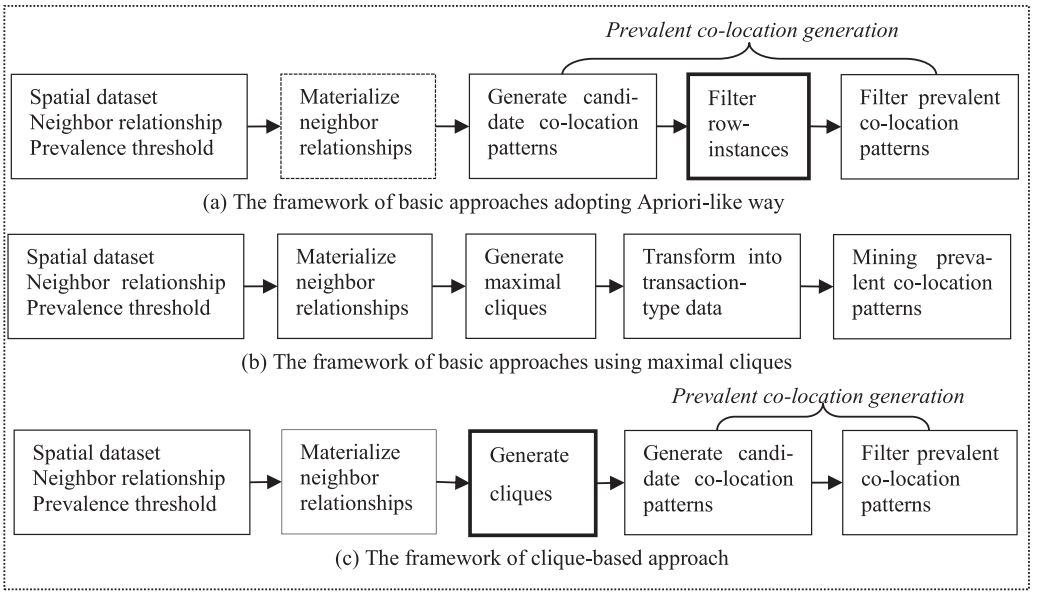


Fig. 2. Comparison of mining frameworks.

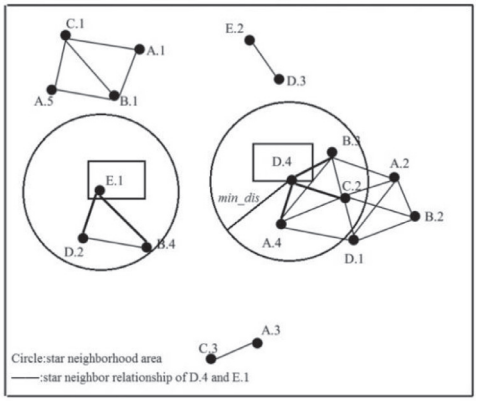
algorithm [11,13]. However, the overall idea of utilizing a recursive iteration procedure based on depth-first search algorithm has not been changed. There also exist several approaches focusing on generating all maximal cliques especially for co-location pattern mining. An efficient algorithm called GridClique was proposed in [2] to generate maximal cliques from spatial datasets. It is inefficient because of too many intersection operations in it, especially in dense spatial datasets. In [17], a polynomial algorithm called AGSMC was proposed to generate all maximal cliques from spatial datasets. However, in a spatial dataset not sufficiently sparse, its computational complexity reaches as high as $O(m^5)$, where m is the number of instances. To make use of association analysis methods performed on transaction-type data, some approaches [18,20,22] have provided different methods to transform maximal cliques into transaction-type data, but they may lose some prevalent co-location patterns [17].

There exist some other interesting works on mining different kinds of co-location patterns based on prevalent co-location patterns, summarized here: If a prevalent co-location pattern c satisfies that for any of its superset $c'(c \subseteq c')$, $\mathrm{PI}(c) \neq \mathrm{PI}(c')$, c is a closed co-location pattern [32]; If a prevalent co-location pattern c satisfies that for any of its superset $c'(c \subseteq c')$, c' is not prevalent, c is a maximal co-location pattern [24,29]; interesting co-location patterns are subsets of prevalent co-location patterns interactively selected by the user [4,5,25]; the N-closed co-location patterns [26] are subsets of co-locations to improve closed co-location patterns [32]; non-redundant co-location patterns [27] are subsets of prevalent co-location patterns where each co-location pattern cannot cover any other one, obtained by considering their row-instances; high utility co-location patterns [28] are subsets of prevalent co-locations having high values.

As compared with the above works on co-location pattern discovery, our approach efficiently generates *cliques* instead of *maximal cliques*, transforms cliques into a hash structure instead of transaction-type data, and designs an efficient method to discover prevalent co-location patterns instead of traditional association analysis methods. Thus, our approach reduces the cost of clique generation, effectively finds a proper prevalence threshold, and avoids identifying row-instances for co-location pattern discovery. Furthermore, previous extended works based on prevalent co-location discovery can be made more efficient using our approach.

2.4. Mining frameworks

In the literature of co-location pattern mining algorithms using an Apriori-like way, they commonly involve several steps as shown in Fig. 2(a). In these approaches, most of the computational time is devoted to identifying row-instances (bold in Fig. 2(a)). To reduce the cost of identifying row-instances, join-less [31] adds a step which models the space by materializing the spatial neighbor relationships into star neighborhoods (dotted in Fig. 2(a)). There exist many interesting and effective structures for modeling spatial data, for example, [9] builds a topological space of terms by forming incremental neighborhoods on each term and generates a term Simplicial Complex to extract concepts from the text; [10] builds a fuzzy hierarchical space by using Conditional Random Field (CRF) and co-current relations among features for document clustering, etc. Although most of them are efficient and have good performances, because only the distance between two instances is considered in co-location pattern mining, a simple and effective model is preferred. Thus, in recent years many approaches on different kinds of co-location pattern mining adopted star neighborhood to model spatial data.



(a) Star neighbor partition

SNs	Instance	BNs
-	A.1	{B.1,C.1}
-	A.2	{B.2,B.3,C.2,D.1}
-	A.3	{C.3}
-	A.4	{B.3,C.2,D.1,D.4}
-	A.5	{B.1,C.1}
{A.1,A.5}	B.1	{C.1}
{A.2}	B.2	{C.2,D.1}
{A.2,A.4}	B.3	{C.2,D.4}
-	B.4	{D.2,E.1}
{A.1,A.5,B.1}	C.1	-
{A.2,A.4,B.2,B.3}	C.2	{D.1,D.4}
{A.3}	C.3	-
{A.2,A.4,B.2,C.2}	D.1	-
{B.4}	D.2	{E.1}
-	D.3	{E.2}
{A.4,B.3,C.2}	D.4	-
{B.4,D.2}	E.1	-
{D.3}	E.2	-

(b) Neighborhood list

Fig. 3. Spatial neighbor relationship materialization.

The common framework for traditional co-location pattern mining approaches using maximal cliques is shown in Fig. 2(b). These approaches first model the spatial datasets as star neighborhoods as proposed in join-less, and then generate maximal cliques using a star neighborhood list, next they transform the generated maximal cliques into transaction-type data, and finally they adopt association analysis methods to generate prevalent co-location patterns from the transaction-type data.

The framework of our proposed approach is shown in Fig. 2(c). Compared with related Apriori-like approaches, our proposed approach avoids the expensive process of filtering row-instances (bold in Fig. 2(a)). Meanwhile, our clique-based approach can find an appropriate prevalence threshold effectively. Compared with related maximal clique based approaches, our approach can efficiently generate cliques. Furthermore, an efficient prevalent co-location generation method is proposed instead of association analysis methods.

3. The clique-based approach

In this section, we discuss our approach in four aspects according to Fig. 2(c): neighborhood materialization (Section 3.1), clique generation (Section 3.2), prevalent co-location generation including candidate generation (Section 3.3) and prevalent co-location filtering (Section 3.4). To better illustrate our proposed approach, all the features are sorted in a predefined order, as well as instances. In this paper, we sort all the features in lexical order of their names. Instances of different features are sorted by their features, and instances of the same features are sorted by their *index* values. For example, feature named A $<$ feature named B, instance B.1 $>$ instance A.1, instance B.1 $<$ instance B.10.

3.1. Neighborhood materialization

Given a spatial neighbor relationship R, a spatial dataset can be represented as a neighbor graph in which a node is an instance, and an edge between two nodes represents the spatial neighbor relationship (shown in Fig. 1(a)). In our approach, the star neighborhood partition method in [31] is modified as our neighborhood partition model. The **star neighborhood** of an instance s is defined as a set of s and instances in its neighborhood whose feature names are greater than the feature name of s. To use star neighborhood in designing our approach for generating cliques efficiently, we enhance and extend the definition of star neighborhood as follows:

Definition 1. Given an instance $s \in S_i$, a set of instances $Ns(s) = \{s\} \cup \{s' \in S_j | j \neq i \cap R(s', s)\}$ is defined as the **neighborhood** of s, where R is the spatial neighbor relationship.

Definition 2. Given an instance $s \in S_i$, a set of instances $SNs(s) = \{s' \in S_j | j < i \cap R(s', s)\}$ is defined as the **small neighborhood** of s.

Definition 3. Given an instance $s \in S_i$, a set of instances $BNs(s) = \{s' \in S_j | j > i \cap R(s', s)\}$ is defined as the **big neighborhood** of s.

Example 2. Fig. 1(c) gives the compressed star neighborhood list storing the star neighborhood of each instance based on the example graph shown in Fig. 1(a). Fig. 3(a) shows the possible neighborhood areas of instances D.4 and E.1, which are represented by circles whose radiuses are the user-specific neighbor distance min_*dist*. The bold solid lines in each circle represent a star spatial neighbor relationship with the central instance. D.4 has star spatial neighbor relationships with A.4, B.3, and C.2. For the instance B.3, the traditional star neighborhood of B.3 is {B.3, C.2, D.4}, Ns(B.3) = {A.2, A.4, B.3, C.2, D.4},

SNs(B.3) = {A.2, A.4}, *BNs* (B.3) = {C.2, D.4}, it is obvious that for an instance s, $Ns(s) = SNs(s) \cup \{s\} \cup BNs(s)$. Fig. 3(b) lists the *SNs* and *BNs* of all the instances in Fig. 3(a).

To generate the neighborhood list containing the neighborhood of each instance from a spatial dataset, a direct and straightforward way is to check spatial neighbor relationships of each instance with all the others, thus, if the number of instances is m, the total computation time is $O(m^2)$. To efficiently generate the neighborhood list, we designed an efficient method which divides the whole space into grids. Algorithm 1 gives its pseudo code. This method first divides the whole space into several grids with area min_*dist* * min_*dist*, and each grid includes instances whose positions are located within it (Step 1). And then, for an instance s in a grid g, only instances in grids around g (8 directions and g itself, Step 3) are to be checked (Step 6), if two instances are neighbors (the distance between them is no more than min_*dist*), they will be added to the neighbor list (Steps 6–11).

Algorithm 1 Neighborhood materialization.

```
Input:
  F: set of features
  S: set of instances
  min_dist: neighbor distance threshold
Output:
  nbs: neighborhood list
Steps:
 1. grids = DivideSpace(min_dist, S) /*Divide the instances into grides*/
 2. For Each gird g In grids Do   /*Traverse each grid */
 3.   ngrids=GetNerighborGrids(g);   /*Get grids of g's 8 directions and g itself*/
 4.   For Each instance s In g Do
 5.     For Each instance s' In ngrids Do
 6.       If Is_Neighbor(s, s', min_distance) Then
 7.         If s < s' Then   /* s < s' => s' ∈ BNs(s), s ∈ SNs(s') */
 8.           nbs[s].Add_BNs(s'); nbs[s'].Add_SNs(s);
 9.         Else     /* s > s' => s ∈ BNs(s'), s' ∈ SNs(s) */
10.           nbs[s'].Add_BNs(s); nbs[s].Add_SNs(s');
11.       End If
12.     End For
13.   End For
14. End For
```

Suppose the area of the whole space is A, and the number of instances is m. The number of grids is $A/\text{min\_}dist^2$, and for each instance in a grid, it will be checked with instances of 9 other grids, and the average number of instances per grid is $\text{min\_}dist^2 * m/A$. Thus, the number of total comparisons is $9 * m * \text{min\_}dist^2 * m/A = m^2 * (9\text{min\_}dist^2/A)$. In general, $\text{min\_}dist^2 << A$, our method is much more efficient than direct traversal.

3.2. Clique generation

Given the neighborhood list of a spatial dataset, a tree structure can be built to find cliques. Thus, to make our approach efficient in different kinds of spatial datasets (e.g., different densities, different volumes), two efficient schemas are presented to find cliques — Instances Driven Schema (IDS for short) and Neighborhood Driven Schema (NDS for short).

Definition 4. Given a clique cl, the first instance of this clique is called the **head** of cl, denoted as H_{cl}, and the set of other instances in cl is called **body** of cl, denoted as B_{cl}. The clique cl is called H_s-Clique if its head is s, and all cliques starting with an instance s are called H_s-Cliques.

Example 3. Given two cliques cl_1= {A.1, B.1, C.2}, cl_2= {A.2, B.2, C.1}, H_{cl_2} = A.2, B_{cl_2} = {B.2, C.1}, cl_1 is a $H_{\text{A.1}}$–Clique, and cl_2 is a $H_{\text{A.2}}$–Clique.

Given the neighborhood list of a spatial dataset, the two schemas aim to find H_{s_i}-Cliques for each instance s_i ($1 \le i \le m$, where m is the total number of instances).

3.2.1. Instances driven schema (IDS)

In this subsection, a simple schema for generating cliques by using a tree structure is presented. This tree structure is defined as below.

Definition 5. An **Instances-Driven-Schema-Based clique tree** (I-tree for short) is a tree structure defined as below.

1. It consists of one *root* labeled as "*root*".
2. Each node except *root* contains two fields: *instance-name* and *node-link*, where *instance-name* registers which instance this node represents, and *node-link* links to the right sibling node representing a bigger instance or *null* if there is none.
3. The node whose parent is *root* is called a **head-node**. A head-node representing an instance s is denoted as hn_s. The node that is a descendant of a head-node and represents an instance s is denoted as n_s.

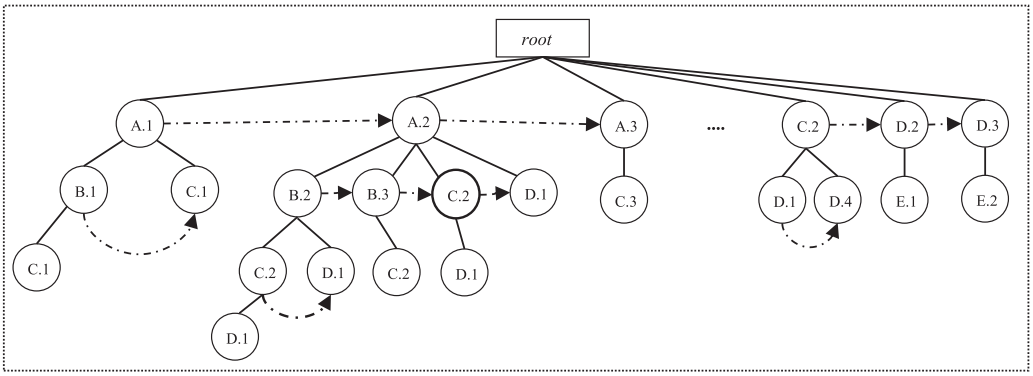


Fig. 4. An example I-tree.

4. The set of instances represented by a leaf node and all its ancestors is called an **I-clique**. Any two instances from an I-clique satisfy the neighborhood relationship.

For constructing an I-tree, some lemmas and definitions are given as below.

Lemma 1. *Given an instance s, all the H_s-Cliques can be generated from BNs(s).*

Proof. (1) H_s-Cliques are cliques starting with instance s, and the following instances in each H_s-clique are bigger than s and have the spatial neighbor relationship with s; (2) BNs(s) contains all the instances that are bigger than s and have spatial neighbor relationships with s. Thus, all the H_s-Cliques can be generated from BNs(s).

Lemma 2. *Given an n-size candidate clique $cl = \{s_1, s_2, \ldots, s_n\}$, $\forall s_i \in cl(1 \le i \le n-1)$, $\forall s_j(j > i), R(s_i, s_j) = true \Rightarrow cl$ is a clique.*

Proof. For an instance s_i in cl, it only needs to check the spatial neighbor relationships with its bigger instances in cl because the instances in cl are sorted in order and the spatial neighbor relationship is symmetric. When dealing with s_i in order, for any s_j $(1 \le j < i)$ in cl, $R(s_i, s_j) = R(s_j, s_i)$, and $R(s_j, s_i)$ has been checked when dealing with s_j.

Definition 6. For a node n_s in an I-tree, its **right sibling instances** is a set of instances represented by its right siblings with different features, denoted as $RS(n_s)$.

Example 4. Fig. 4 shows an example of the I-tree constructed based on Fig. 1(a). The I-cliques generated from the I-tree with the head-node $hn_{A.2}$ are {A.2, B.2, C.2, D.1}, {A.2, B.2, D.1}, {A.2, B.3, C.2}, {A.2, C.2, D.1} and {A.2, D.1}. The set of right siblings of the bold node $n_{C.2}$ is $RS(n_{C.2}) = \{D.1\}$, for each node n_s, $RS(n_s)$ can be easily obtained from the *node-link* field.

Based on the above lemmas and definitions, we can get the children of each node according to the following lemma.

Lemma 3. For a head-node hn_s in an I-tree the children of hn_s are $BNs(hn_s)$, and for one of the other non-root nodes n_s, the children of n_s are $BNs(n_s) \cap RS(n_s)$.

Proof. For an instance s represented by a head-node hn_s, only its neighbors are needed to generate 2-size H_s-Cliques, and because all the instances in cliques are sorted in order, each 2-size H_s-Clique should comprise s and one of its bigger neighbor included in $BNs(hn_s)$. For one of the other non-root nodes n_s, the siblings are instances neighboring with n_s's ancestor nodes, and the spatial neighbor relationship between it and its every sibling should be checked, according to Lemma 2, only nodes with bigger instances are considered. Thus, the checked result is $BNs(n_s) \cap RS(n_s)$.

The complete I-tree can be constructed according to Lemma 3. IDS uses a size-wised manner to generate I-cliques from an I-tree. Once an I-clique is generated, a pruning operation is used to make full use of the memory space: when a node n_s is a leaf node, n_s can be safely pruned. If n_s's parent node becomes a leaf node when n_s is pruned, n_s's parent node also can be safely pruned, and this pruning operation continues until one of n_s's ancestor node has one or more children after one of its children is pruned. The pseudo code of IDS is shown in Algorithm 2. To generate I-cliques staring with an instance *s*, IDS constructs a head-node hn_s for *s* (Step 4), then IDS uses a breadth-first manner to traverse each node of hn_s's descendants (Steps 6–15). If the current traversing node n_s doesn't have any child, an I-clique consisting of instances from hn_s to n_s can be generated from the I-tree (Step 10), and then the pruning operation is performed on n_s and its ancestors (Step 11). Otherwise, IDS adds the children of n_s according to Lemma 3. The whole process cannot stop until all non-root nodes in this I-tree have been traversed.

Example 5. An example shown in Fig. 5 is used to explain IDS algorithm in detail. To generate $H_{A.2}$−Cliques from an I-tree, a head-node $hn_{A.2}$ is added to the root of the I-tree, and then IDS finds its children according to Lemma 3, BNs (A.2) = {B.2, B.3, C.2, D.1}, thus, the nodes representing BNs(A.2) are added to $hn_{A.2}$ as shown in Fig. 5(a). For the next iteration, four

Algorithm 2 IDS algorithm.

```
Input:
  nbs: neighborhood list (including BNs of each instance)
  S: set of instances
Output:
  Cls: list of I-cliques
Steps:
  1. iTree = Initialize_Itree();      /*To create a root node on the tree*/
  2. For Each instance s In S Do    /*To get Hs-Cliques for each s*/
  3.   queue = Initialize_queue();   /*breadth-first manner*/
  4.   headNode = iTree.Root.AddHeadNode(s);   /*To get Hs-Cliques*/
  5.   queue.In(headNode);   /*Add headNode to the rear of queue*/
  6.   While NotEmpty(queue) Do
  7.     currNode = queue.Out;   /*Get a node from the front of queue*/
  8.     childrenNodes= GetChildren(currNode);   /*Get children according to Lemma 3*/
  9.     If IsEmpty(childrenNodes) Then/*The node is a leaf node, clique can be generated*/
  10.      Cls.Add(GetClique(currNode));   /*Add clique to the result set*/
  11.      RemoveAncestors(currNode);  /*Remove currNode and its ancestors recursively*/
  12.    Else
  13.      iTree.AddNodes(currNode, childrenNodes);   /*Add children to currNode*/
  14.      queue.In(childrenNodes);   /*Add currNode's children to the rear of queue*/
  15.    End If
  16.  End While
  17. End For
```

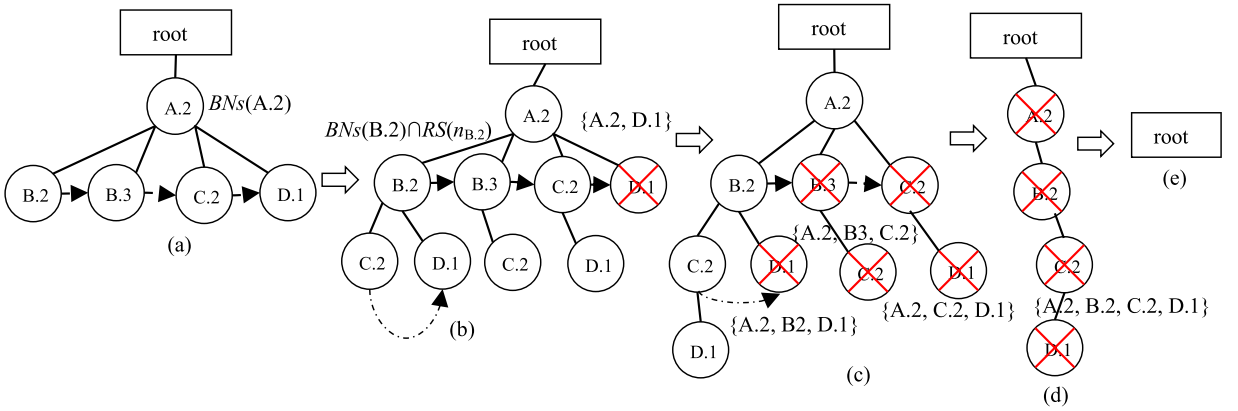


Fig. 5. The IDS approach.

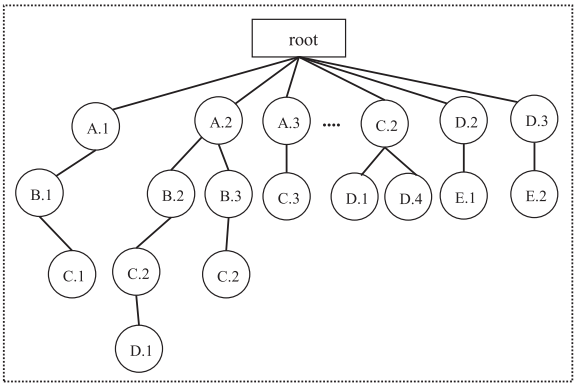
children nodes of $hn_{A.2}$ are treated as the current node, respectively. When treating $n_{D.1}$, $BNs(D.1) \cap BS(n_{D.1})$ is an empty set, thus an I-clique {A.2, D.1} is generated, and $n_{D.1}$ is pruned. Its parent $n_{A.2}$ has other children, thus, $n_{A.2}$ cannot be pruned, as shown in Fig. 5(b). In this way, 3-size and 4-size I-cliques are generated shown in Fig. 5(c) and Fig. 5(d), respectively. The whole process of generating $H_{A.2}$–Cliques stops when the I-tree only contains the *root,* shown in Fig. 5(e).

To analyze the computational complexity of the IDS algorithm, for an instance s, in the worst case $\{s\} \cup BNs(s)$ is a clique, which means that the children of the node n_s are $RS(n_s)$. We define a function $f(x)$ which calculates the number of descendants of a node n_s whose $|RS(n_s)| = x$, thus, $f(1) = 1$, $f(2) = 2 + f(1) = 3$, $f(x) = x + f(x\text{-}1) + f(x\text{-}2) + \ldots + f(1) = 2^x\text{-}1$. This formula can be explained as follows: The number of children of n_s is x, the children number of each child of n_s from left to right is $f(x-1)$, $f(x\text{-}2)$,..., $f(1)$. The main cost of IDS is the process of breadth-first, thus, the computational complexity to generate H_s-Cliques is $O(f(|BNs(s)|)) = O(2^{|BNs(s)|})$. Suppose the number of edges in a spatial dataset is E, then for each instance s in S, $\sum |BNs(s)| = E$ and if all the instances are distributed evenly, then the worst computational complexity of IDS is $O(|S| * 2^{E/|S|})$, but note that as there exists no spatial dataset where each instance has the spatial neighbor relationship with the others, the average number of nodes in each height from the tree structure is more or less equal, as shown in Fig. 4. Thus, the average computational complexity of IDS is $O(E^2/|S|)$. When the spatial dataset is sparse enough (e.g., $E = |S|$), the computational complexity of IDS can be as efficient as $O(E)$.

IDS uses a breadth-first way which can perform adding and pruning operations at the same time. Thus it has good space performance. It is independent of other instances when getting the H_s-Cliques of each instance s. Thus, IDS is quite easy to be transformed into an efficient parallel or multi-thread program.

Definition 7. Given a set of cliques *scl* of a spatial dataset materialized by spatial neighbor relationships, and the set containing all cliques of this dataset is marked as *ascl*, if the following formula is satisfied, *scl* is a set of **complete cliques**.

$$\forall cl \in ascl, \exists cl' \in scl \Rightarrow cl \subseteq cl'$$



(a) An example N-tree

SNs	Instance	BNs
-	A.1	{B.1,C.1}
-	A.2	{B.2,B.3,C.2,D.1}
-	A.3	{C.3}
-	A.4	{B.3,C.2,D.1,D.4}
-	A.5	{B.1,C.1}
{A.1,A.5}	B.1	{C.1}
{A.2}	B.2	{C.2,D.1}
{A.2,A.4}	B.3	{C.2,D.4}
-	B.4	{D.2,E.1}
{A.1,A.5,B.1}	C.1	-
{A.2,A.4,B.2,B.3}	C.2	{D.1,D.4}
{A.3}	C.3	-
{A.2,A.4,B.2,C.2}	D.1	-
{B.4}	D.2	{E.1}
-	D.3	{E.2}
{A.4,B.3,C.2}	D.4	-
{B.4,D.2}	E.1	-
{D.3}	E.2	-

(b) Neighborhood list

Fig. 6. An example N-tree and neighborhood list for the construction of N-tree.

Lemma 4. *I-cliques are complete.*

Proof. According to Lemma 1, to generate Hs-Cliques for an instance s, the candidate clique generated from BNs(s) is complete. Meanwhile, according to Lemma 3, the children nodes generation operation will not lose any clique. If a node ns is a leaf node, supersets of the clique cl generated from ns cannot be cliques. Thus, the I-cliques generated from ancient nodes of ns are subsets of cl. If ns has children nodes, the clique generated from ns must be a subset of one of the I-cliques generated from its descendants. Thus, the generated I-cliques contain all possible cliques and are complete.

According to Lemma 3 and Lemma 4, the following theorem holds:

Theorem 1. *I-cliques are correct and complete.*

3.2.2. Neighborhood driven schema (NDS)

In IDS, the number of I-cliques may be very large, especially in dense data sets, which may reduce the efficiency of the following operations of our approach. To find maximal cliques is costly, but if we can find an efficient way to reduce the number of result cliques as much as possible, it will make our approach more efficient. Thus, in this subsection, another schema called Neighborhood Driven Schema (NDS for short) is detailed.

Definition 8. A **Neighborhood-Driven-Schema-Based clique tree** (N-tree for short) is a simplified I-tree where each node in N-tree contains only one *instance-name* field, and the clique generated from one of its leaf nodes is called an **N-clique**. Any two instances from an N-clique satisfy the neighborhood relationship.

Fig. 6(a) shows an example N-tree constructed based on Fig. 1(a). Compared with the I-tree shown in Fig. 4 with the same example spatial data in Fig. 1(a), the number of cliques generated by the N-tree is less than that by the I-tree.

Lemma 5. *Given an n-size clique* $cl=\{s_1, s_2,..., s_n\}$*, for an instance* $s \notin cl$ *and* $s > s_n$*, if* $\forall s_i \in cl$, $R(s,s_i)$= *true,* $cl \cup s$ *is a new clique.*

Proof. A clique is a set of instances such that every two distinct instances in this set are neighbors. With this condition, it is obvious that $cl \cup s$ is a new clique.

To build an N-tree, for each H_s-Clique cl, $B_{cl} \subseteq BNs(s)$ (Lemma 1), which means that s has occurred in $SNs(s')$ where $s' \subseteq BNs(s)$, thus, $SNs(s')$ is used to check clique. Given a clique $cl=\{s_1, s_2, ..., s_n\}$, for each instance $s_i \in cl$, $\{s_{i+1}, s_{i+2},...,s_n\}$ is included in $BNs(s_i)$, and for each $s_j \in \{s_{i+1}, s_{i+2},...,s_n\}$, $\{s_1, s_2,...,s_{i-1}\}$ is included in $SNs(s_j)$, thus, to check whether an instance s ($s \in BNs(s_1)$) can be added to cl to form a clique. If $\{s_2,s_3,...,s_n\} \subseteq SNs(s) \cap BNs(s_1)$, which means that s has spatial neighbor relationships with each instance in cl, s can be added to cl. According to differing results of $SNs(s) \cap BNs(s_1)$, four strategies are devised as follows.

Given a clique $cl=\{s_1, s_2, ..., s_n\}$ with head s_1, an instance $s \in BNs(s_1)$;

Strategy 1. if $|cl|=1, SNs(s) \cap BNs(s_1) = \Phi$, s can be added to cl because $s \in BNs(s_1)$ and the relationship measure is symmetric.

Strategy 2. if $|cl| > 1, SNs(s) \cap BNs(s_1) \supseteq \{s_2,s_3, ...,s_n\}$, s can be added to cl because $\forall s_i \in cl, R(s, s_i) = \text{true}$.

Strategy 3. if $|cl| > 1, cl' = SNs(s) \cap BNs(s_1) \subseteq \{s_2,s_3, ...,s_n\}$, cl' can be added to s_1, $s_1 \cup cl'$ is a new clique because $\forall s_i \in s_1 \cup cl', R(s, s_i) = \text{true}$.

Strategy 4. if $|cl| > 1, SNs(s) \cap BNs(s_1) \not\subset \{s_2,s_3, ...,s_n\}$ and $\{s_2,s_3, ...,s_n\} \not\subset SNs(s) \cap BN(s_1)$ but $cl' = SNs(s) \cap BN(s_1) \cap \{s_2,s_3, ...,s_n\} \neq \Phi$, cl' can be added to s_1, and $s_1 \cup cl'$ is a new clique because $\forall s_i \in s_1 \cup cl', R(s, s_i) = \text{true}$.

Based on the above strategies, NDS first builds an N-tree and then finds N-cliques. The pseudo code for NDS is shown in Algorithm 3. NDS traverses each neighborhood to build an N-tree (Steps 2–23). For an instance s of a neighborhood, NDS first gets small neighbors of s ($SNs(s)$, step 3), and then creates a head-node representing s_1 from each instance in SNs(s) (Step 5). Next, according to the different results of $SNs(s) \cap BN(s_1)$ (Steps 6–7), four strategies are used to complete the N-tree (Steps 8–22). After all neighborhoods in the neighborhood list are traversed, the whole N-tree is built. Finally, N-cliques are generated from leaf nodes of N-tree (Step 25).

Algorithm 3 NDS algorithm.

```
Input:
  nbs: neighborhood list (including BNs and SNs of each instance)
  S: set of instances
Output:
  Cls: list of N-cliques
Steps:
 1. nTree = Initialize_Ntree();      /*To create a root node on the tree*/
 2. For Each instance s In nbs.Instances Do   /*traverse each neighborhood*/
 3.   queue = SNs(s);   /*Get small neighbors of s*/
 4.   While NotEmpty(queue) Do  /* Take each instance in queue as a head */
 5.     headNode = nTree.AddHeadNode(queue.Out);  /*Create a head-Node from queue*/
 6.     bodies = GetCurrentBodies(headNode); /*Get the current bodies of head*/
 7.     relation = queue∩BNs(headNode.instance); /*Get the candidate instances*/
 8.     If bodies == null || relation == null Then  /*Strategy 1 */
 9.       headNode.AddNode(s)
10.     Else  /*different relations with different strategies*/
11.       For Each list l in bodies do
12.         If l==relation Then   /*Strategy 2 */
13.           l.AddNode(s);
14.         Else If relation ⊆ l Then  /*Strategy 2*/
15.           head.AddNodes(relation∪s);
16.         Else If l ⊆ relation Then /*Strategy 3 */
17.           head.AddNode(relation∩l∪s);
18.         Else If NotEmpty(l∩relation) Then   /*Strategy 4 */
19.           head.Add(l∩relation∪s);
20.         End If
21.       End For
22.     End If
23.   End While
24. End For
25. Cls = GetResultFromLeaves();
```

Example 6. An example is used to explain NDS algorithm in detail. Fig. 6(b) shows the neighborhood list of Fig. 1(a). To generate $H_{A.2}$–Cliques, the body of each $H_{A.2}$–Clique should be selected from $BNs(A.2) = \{B.2, B.3, C.2, C.1\}$ highlighted in gray. B.2 is checked first, and $SNs(B.2) \cap BNs(A.2) = \Phi$, which means that {B.2} can be directly added to A.2 (Strategy 1). Then B.3 is treated, $SNs(B.3) \cap BNs(A.2) = \Phi$, thus B.3 can be directly added to A.2. Next, for C.2, $SNs(C.2) \cap BNs(A.2) = \{B.2, B.3\}$, C.2 can be added to {A.2, B.2} and {A.2, B.3}, respectively (Strategy 3). Finally, $SNs(D.1) \cap BNs(A.2) = \{B.2, C.2\}$, D.1 can be added to {A.2, B.2, C.2} (Strategy 2). By traversing each item of the neighborhood list, an N-tree is built, as shown in Fig. 6(a).

To analyze the computational complexity of NDS, suppose the number of edges in a materialized spatial dataset is E, if the execution time of Steps 5–22 is t, the executing time of Steps 3–23 is $|SNs(s)| * t$, The total execution time of NDS is $n^*|SNs(s)|^*t$, where n is the number of neighborhood. For each instance s, $|SNs(s)|$ is the number of edges connecting with its smaller instances, while the other edges are included in each $SNs(s')$ where s' is an instance bigger than s and neighbors with s, and $n^*|SNs(s)| = E$, thus, the total execution time of NDS is E^*t. The main cost in the *while* procedure (Steps 4–23) is Step 7, and the computational complexity of the intersection operation for two sets is $O(p+q)$, where p and q are the length of the two sets respectively. In the worst situation, all the instances are neighbors with each other, and the average number of edges of each instance is $E/|S|$. Thus, the computational complexity of NDS in the worst case is $O(E^2/|S|)$, which is equal to the average computational complexity of IDS. In general situations, the computation complexity of NDS can be $O(E)$.

Lemma 6. *There is no subset or duplication in N-cliques starting with an instance s.*

Proof. The tree structure is a kind of compressed way for storing cliques. Thus, the same cliques cannot be expressed in a tree as two nodes because each node in N-tree cannot have two same children, they will be shown as one child node. Thus, there is no duplication in N-cliques. Four strategies (Strategy 1-Strategy 4) make the decisions concerning which node(s) can be added to the head-node hn_s. If a new clique is generated which is a subset of existing cliques starting with s, then the adding operation from hn_s makes no change at all to N-tree. Thus, there is no subset in N-cliques starting with s.

Lemma 7. *N-cliques are complete.*

Proof. Given a clique cl= $\{s_1, s_2, ..., s_n\}$, if cl is a maximal clique, $\forall s \notin cl$, s cannot be added to cl, $\exists s' \in cl, R(s,s') = false \Rightarrow s' \notin SNs(s) \Rightarrow$NDS won't add s to the clique. If cl is not a maximal clique, $\exists s \notin cl$, s can be added to cl, as $s \in BNs(s_1), \forall s' \in cl, R(s,s') = true \Rightarrow s' \in SNs(s) \Rightarrow s \in BNs(s_1) \cap SNs(s) \Rightarrow$NDS can add s to cl. Thus, NDS can generate all the maximal cliques. According to Definition 7, the set containing all maximal cliques is complete. Thus, N-cliques are complete.

Table 1
Comparison of IDS and NDS.

Function	Meaning	Comparison
$NoC(X)$	Number of cliques generated by algorithm X	Noc(IDS) $\geq$ Noc(NDS)
$Et(X)$	Execution time of algorithm X	Rt(IDS) $\geq$ Rt(NDS)
$Sp(X)$	Space cost of algorithm X	Sp(IDS) $\leq$ Sp(NDS)

Based on Lemma 7 and the four strategies in this subsection, the following theorem holds:

Theorem 2. The cliques generated by NDS are correct and complete.

3.2.3. Schema comparison

Both of the two schemas aim to generate complete and correct cliques using a tree structure. Table 1 gives a summary comparison of the two schemas on the number of cliques (Noc), the execution time (Et), and the space cost (Sp), respectively. According to Lemma 6, there is no subset or duplication in H_s-Cliques generated by NDS, thus, Noc(IDS) $\geq$ Noc(NDS). As analyzed in Section 3.2.1 and 3.2.2, the average computation complexity of IDS is $O(E^2/|S|)$, and the average computation complexity of NDS is $O(E)$, in general situations, $E \geq |S|$, thus, Et(IDS) $\geq$ Et(NDS). IDS uses a pruning-while-constructing way to make full use of the limited space, while NDS needs to construct a complete N-tree to generate N-cliques. Thus, Sp(IDS) $\leq$ Sp(NDS).

According to the comparisons above, NDS is fit to be performed in dense but not big spatial datasets, and IDS is fit to be performed in big but not dense spatial datasets. With a dense and big dataset, either of the two schemas can be used, but NDS may cause a memory overflow exception in a limited space and IDS may require a long running time.

3.3. Candidate generation

In this subsection, a hash structure is introduced to compress cliques and generate candidates instead of transforming cliques into transaction-type data.

Definition 9. A **Compressed clique hash** (C-hash for short) is a hash structure with key-value pairs <*key, value*>, where

(1) The *key* is a set of features F_c.
(2) The *value* is a list of hash structures with key-value pair <*key*', *value*'> where the *key*' is a feature $f \in F_c$ and the *value*' is a set of instances of f. $\cup key' = key$.

The process for constructing C-hash is quite simple, as shown in Algorithm 4. Suppose $F(s)$ represents the feature of an instance s, then given a clique cl, a feature set $\cup F(s_i)$ is the *key* of C-hash where $s_i \in cl$(Step 2). The new key will be added if the *key* doesn't exist in C-hash (Steps 3–4), and then each key-value pair <$F(s_i)$, s_i> will be added as a *value* (Steps 5–7). The process will not stop until all cliques are traversed (Steps 1–8).

Algorithm 4 Candidate Generation.

```
Input:
  cls: Cliques generated by IDS or NDS
Output:
  chash: c-hash structure
Steps:
  1. For Each clique cl In cls Do
  2.   newKey=GetFeatures(cl)   /*Get the features of instances in cl as a key*/
  3.   If Not chash.ContainsKey(newKey) Then  /*If candidates didn't contain newKey*/
  4.     chash.AddKeyAndInitialize(NewKey); /*Do initialize operations for newKey*/
  5.   For Each feature f in newKey Do
  6.     chash[newKey][f].AddInstances(cl);
  7.   End For
  8. End For
```

The computational complexity of candidate generation is $O(L)$, where L is the number of I-cliques or N-cliques. Using keys of C-hash as candidates is reasonable because, for each candidate pattern cp from C-hash, at least one row-instance of cp can be found. The following lemma holds.

Lemma 8. For each candidate pattern cp generated from C-hash, PI(cp) > 0

Example 7. Fig. 7 shows an example C-hash, for a clique cl = {A.2, B.4, C.1, D.3}, the *key* which is the feature of instances in cl is {A, B, C, D}, the *value* of *key* {A, B, C, D} is a list [<A, {A.2}>, <B, {B.4}>, <C, {C.1}>, <D, {D.3}>]. For another clique cl'={A.3, B.1, C.1, D.1} with the same *key*{A, B, C, D}, the *value* of *key* {A, B, C, D} is updated as [<A, {A.2, A.3}>, <B, {B.1, B.4}>, <C, {C.1}>, <D, {D.1, D.3}>]. Thus the candidates can be found from the *keys* of C-hash, as shown in bold in Fig. 7.

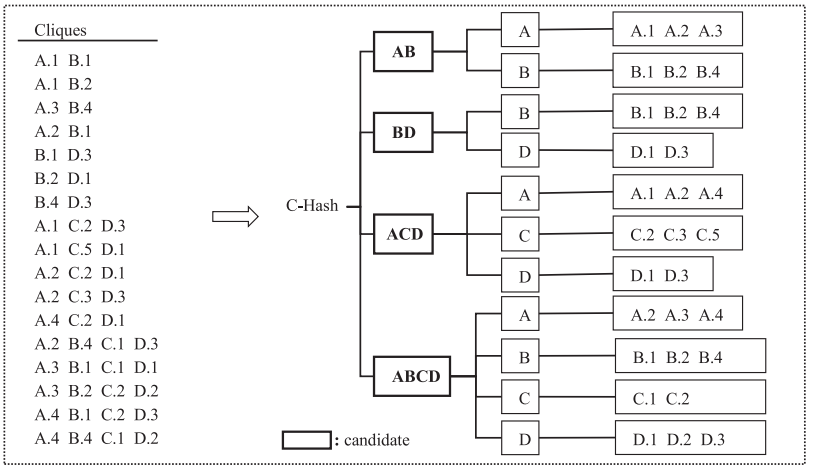


Fig. 7. An example C-hash.

3.4. Prevalent co-location filtering

In order to generate complete and correct prevalent co-locations using candidates generated from C-hash, a lemma is given below.

Lemma 9. *For any prevalent co-location c, there exists a candidate cp that c is a subset of cp.*

Proof. According to Theorem 1 and Theorem 2, the cliques generated by IDS and NDS are complete and correct. For a prevalent co-location c, for each co-location row-instance ci of c, ci is a clique, thus, there exists one or more cliques containing ci. According to the construction of C-hash, a set of features of instances in any clique is deemed as a key, and the candidates are a set of keys in C-hash. Thus, there exists a candidate cp that c is a subset of cp.

Based on Lemma 9, the pseudo code for prevalent co-location filtering is shown in Algorithm 5. The candidates are sorted by sizes (Steps 1–2) and checked from long sizes to short sizes (Step 4). If a candidate *cp* is prevalent, which means that all subsets of *cp* are prevalent, the PI values of all subsets of *cp* are calculated, and then, subsets of *cp* are removed from candidates and added to the result set (Steps 6–10). If a *k*-size candidate *cp* is not prevalent, *cp* is removed from candidates, and its $k-1$-size subsets are added to candidates (Steps 12–14). This procedure continues until the candidate set is empty (Steps 3–16). The worst cost of this process is $O(l*2^{w}*x)$, where l is the number of candidates, w is the average length of candidates, and x is the computational complexity of Step 5.

Algorithm 5 Prevalent co-locations filtering.

```
Input:
  chash: c-hash structure
  min_prev: prevalence threshold
Output:
  cs: list of prevalent co-locations with PI value
Steps:
 1. candidates = chash.Keys;      /*Get candidates from c-hash*/
 2. candidates.Sort();  /*Sort the candidates with sizes from high to low */
 3. While NotEmpty(candidates) Do    /*traverse each candidate from highest size*/
 4.    currCandidate = candidates.First;
 5.    pi = CalculatePI(currCandidate, candidates, chash)     /*Calculate PI value of currCan-
                                                              didate */
 6.    If pi >= min_prev Then  /*If currCandidate is prevalent */
 7.       subsets = GetAllSubsets(currCandidate); /*Get all subsets of currCandidate*/
 8.       PIs = CalculatePIs(subsets);   /*Calculate PIs of all subsets*/
 9.       cs.Add(subsets, PIs); /*Add all subsets and their PIs to the result*/
10.       candidates.Remove(subsets); /*Remove all subsets from candidates*/
11.    Else  /*currCandidate is not prevalent*/
12.       directSubs = GetDirectSub(currCandidate); /*Get size-1 subsets*/
13.       candidates.Remove(currCandidate); /*Remove currCandidate from candidates*/
14.       candidates.AddWithSort(directSubs); /*Add directSubs in order*/
15.    End If
16. End While
```

If the instances participating in *cp* are known, the PI value of *cp* can be calculated. Thus, given a candidate co-location *cp*, the PI value of *cp* can be calculated as follows.

Lemma 10. *Given a k-size candidate pattern cp = {f1,..., fk}, l={cp1,..., cpn} (cpi⊇cp) is a list where each candidate in l is a superset of cp. Let cp[f] mean the set of instances of f when the key is cp from the C-hash, and then the PI value of cp is* $PI(cp) = \min_{i=1}^{k}\{|\bigcup_{j=1}^{n} cp_j[f_i]|/|f_i|\}$.

Proof. In order to calculate the PI value of a k-size candidate pattern cp = {f_1, ..., f_k}, the row-instances of cp are cliques comprising all instances of features in cp, and can be included in its superset cliques which can be found in C-hash with keys including cp.

The pseudo code for PI value calculation is shown in Algorithm 6. Given a candidate *cp*, the supersets of *cp* can be accessed from C-hash (Step 1). For each superset of *cp*, instances of features included in *cp* are collected (Steps 2–6), and then the PR value of each feature in *cp* is calculated (Steps 7–9). The minimum PR value is the PI value (Step 10). The computational complexity of this process is $O(l^*w)$, where *l* is the number of supersets of *cp*, and *w* is the size of *cp*. Thus the computational complexity of the whole process of filtering prevalent co-locations is $O(l^*w^*2^w)$. Note that the average length of candidates is small in most situations. Thus, the process can perform efficiently in most situations.

Algorithm 6 Calculate PI value.

```
Input:
  cp: candidate pattern to be calculated
  chash: c-hash structure
Output:
  pi: the PI value of candidate cp
Steps:
  1. supersets = GetSuperSets(c-hash, cp);   /*Get supersets of cp from C-hash*/
  2. For Each pattern cp In supersets Do   /*Traverse each candidate from supersets*/
  3.     For Each feature f in candidate do   /*For each feature in candidate */
  4.         Ins[f].Add(chash.cp[f]);     /*Add instances of feature f to Ins[f] */
  5.     End For
  6. End For
  7. For Each feature f in candidate Do
  8.         PRs[f]=|Ins[f]|/|f    |/*Get PR(f) for each feature in candidate*/
  9. End For
  10. Return min(PRs);
```

Example 8. If the PI value of a candidate pattern *cp* = {A, B} is to be calculated, cliques like {A_i, B_j, ...} are to be searched in C-hash, shown in Fig. 7. The *keys* containing *cp* are listed as *l* = {AB, ABCD}, AB[A] = {A.1, A.2, A.3}, ABCD[A] = {A.2, A.3, A.4}, AB[A]∪ABCD[A] = {A.1, A.2, A.3 A.4}. If A has five instances and B has four instances, PR(A, AB) = 4/5, and in the same way, PR(B, AB) = |AB[B]∪ABCD[B]| / 4 = |{B.1, B.2, B.4}∪{B.1, B.2, B.4}| / 4 = 3/4, thus, PI(AB) = min{PR(A, AB), PR(B, AB)}= 3/4.

3.5. Discussion

Time performance: As explained in previous subsections, the computational complexity of our approach is $O(E^2/m)$ with IDS and $O(E)$ with NDS, where *m* is the number of instances and *E* is the number of edges in the materialized spatial datasets.

Space performance: In general works, the main space cost is to store the row-instances of co-location patterns [16,30,31]. In our approach, the generated cliques are transformed as C-hash, which can effectively avoid storing co-location row-instances.

Completeness and correctness: Section 3.1 gives a materialization method which cannot miss any spatial neighbor relationship. The cliques generated from methods proposed in 3.2 are correct and complete, the candidates generated in Section 3.3 are reasonable, and finally, in Section 3.4, the full candidate generation operation is complete, and the PI calculation is correct. Thus, the following theorem holds.

Theorem 3. The prevalent co-location patterns generated by our clique-based approach are correct and complete.

Flexibility: In our clique-based approach, the PI value of any candidate co-location can be calculated flexibly. Furthermore, given a C-hash, our method (proposed in 3.5) can efficiently discover complete and correct prevalent co-locations no matter what values the prevalence thresholds are given and how many times it runs.

4. Experimental evaluation

In this section, we evaluated the clique-based approach proposed in this paper using synthetic and real datasets. We conducted the following experiments:

- **Effects**. We examined the effects of the clique-based approach for co-location pattern mining. Specifically, we compared the computation costs of neighborhood materialization step (Section 3.1) with and without the Grid-Based (Algorithm 1)

Table 2
Experimental parameters and their values in each experiment.

Symbols	Meaning	Experiment no. (F means Figure, T means Table)								
		F.8a	F.8bF.11c	F.8cF.9aF.11bF.13	F.9bF.10a	F.10b	T.2	F.11a	F.11d	F.12
P	Number of [a]co-locations	20								50
I	Average number of [a]co-location instances	500								1000
D	Spatial area size (*D***D*)	*	5000				1000 10,000	5000		*
F	Number of features	20						*	20	50
Q	Average size of [a]co-locations	5				*	5			10
m	Number of instances	50,000		*	50,000					500,000
min_*prev*	Prevalence threshold	0.2							*	0.1
min_*dist*	Neighborhood distance threshold	50	*	50						100
clumpy	Number of co-location instances generated in a neighborhood area	1			*	1				

[a] initial core co-location.
* variable values.

method. We examined the efficiency of the clique generation step (Section 3.2) compared with GridClique [2] and AGSMC [17]. The efficiency of the step for generating prevalent co-location patterns using cliques (Sections 3.3 and 3.4) was compared with Apriori [1] and FP-Growth [15]. As a complete approach for co-location pattern mining, we compared the effects of the clique-based approach with join-based [16] and join-less [31].

- **Scalability**. We compared the scalability of our clique-based approach with join-based and join-less concerning the number of instances, the number of features, the distance neighbor threshold, and the prevalence threshold.
- **Behavior**. We examined the behaviors of IDS and NDS with different densities in a big spatial dataset.
- **Flexibility**. We evaluated the flexibility of the clique-based approach compared with join-based and join-less in searching for an appropriate prevalence threshold.
- **On real datasets**. Finally, we evaluated our approach with three real datasets, a POI (Point of Interest) dataset from Beijing, a vegetation distribution dataset from the Three Parallel Rivers of Yunnan Protected Area and a rare plant distribution data of the Three Parallel Rivers of Yunnan Protected Area.

4.1. Synthetic data generation

Synthetic datasets were generated using a spatial data generator similar to [16,31]. Table 2 describes the parameters used for the data generator. First, the distribution area of instances was determined with *D***D*. The whole area was divided into grids of size min_*dist**min_*dist*, where min_*dist* is the spatial neighbor distance threshold. Then *P* initial core patterns whose average size was *Q* were generated. The features of each core pattern were randomly chosen from the feature set containing *F* features. An average of *I* row-instances per core pattern were generated. The total number of instances was *m*. For locating a co-location row-instance, we first randomly chose a grid, and then all instances of the row-instance were randomly located within the chosen grid.

In order to generate our specialized datasets, overall data density was controlled by spatial area size *D***D*. Under a fixed total number of instances, a smaller area generates more dense datasets. The data density in neighborhood areas (the number of edges) was controlled by a *clumpy* degree. When a new grid was chosen randomly for locating a co-location row-instance, a *clumpy* number of row-instances were generated in this grid. The default *clumpy* value was 1. If the number of edges of all the instances is *E* when *clumpy* value is 1, then the number of edges of all the instances can be approximately estimated as *E***clumpy*. Thus, the *clumpy* degree can be used to test the affection of edges to our clique-based approach. The parameter values for the synthetic dataset used in each experiment are shown in Table 2. Based on the value setting for each parameter in Table 2, the mean number of instances per feature is 2500 (50,000/20), and because the average number of row-instances of each core co-location pattern is 500, for each core co-location pattern the minimum PI value is about 0.2 (500/2500). This parameter setting can guarantee that even in sparse datasets, the core co-location patterns can be discovered as prevalent co-location patterns, which makes our experiment both reasonable and effective.

All of our experiments were implemented in Visual C# and performed on an Intel PC running Windows 10 with Intel Core i5 4590 @3.30 GHz with 4GB memory.

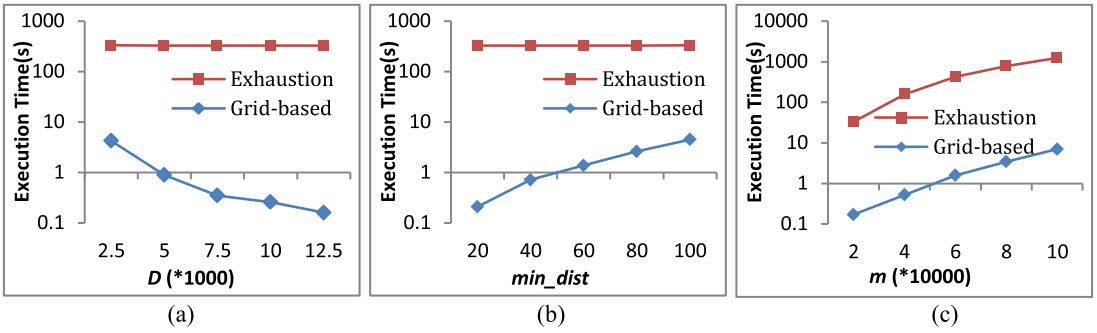


Fig. 8. Effects of neighborhood materialization.

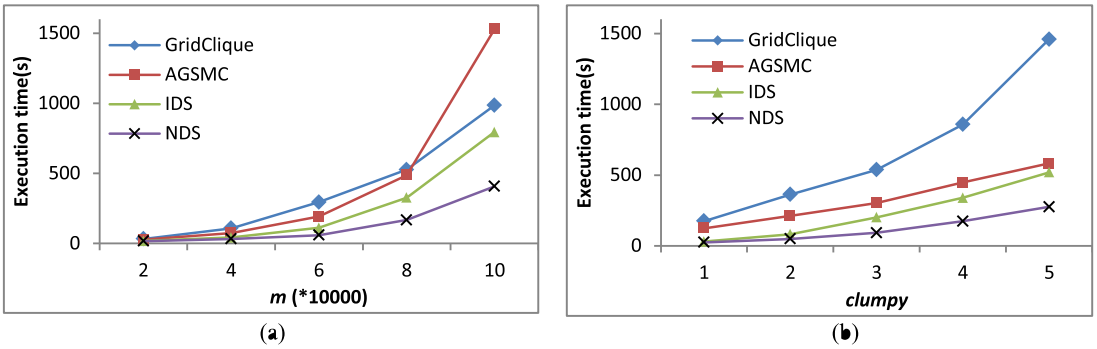


Fig. 9. Effects of clique generation.

4.2. Effects of clique-based approach

4.2.1. Comparison of neighborhood materialization methods

We compared the Grid-Based method (Algorithm 1) with an exhaustive search method. The Exhaustion method traverses each instance and checks its spatial neighbor relationships with all the other instances. Fig. 8 shows the execution times of the two methods with different space area size D, neighborhood distance threshold min_*dist* and the number of instances m. As can be seen, the Grid-Based method performs much more efficiently than the Exhaustion method because the number of instances checked with an instance in Grid-Based method is much less than that in Exhaustion method. As shown in Fig. 8(a), when spatial area size D increases, the average number of instances in each grid decreases, thus the execution time of Grid-Based method decreased. In Fig. 8(b) and (c), an increase in the distance threshold min_*dist* or in the number of instances m makes the average number of instances in each grid increase, thus, the execution time of Grid-Based method increases. The time consumption of the Exhaustion method only depends on the number of instances. As shown in Fig. 8, when m increases, the execution time of the Exhaustion method increases because both the number of instances to be checked and the number of instances checked with an instance increase. Thus, our neighborhood materialization method is efficient.

4.2.2. Comparison of clique generation methods

We examined the efficiency of our clique generation methods IDS and NDS over GridClique method [2] and AGSMC method [17]. The reasons why the two methods are used to be compared with our methods are that both of the two methods aim to generate maximal cliques from spatial datasets for co-location pattern mining and use “neighbor” relationships also adopted in our approach. Fig. 9 shows the execution times of the four methods with different numbers of instances m and *clumpy*. As shown in Fig. 9, GridClique costs more time than the other three methods in most situations because its computational complexity is as much as $O(E^2)$, where E is the number of edges in a materialized spatial dataset. As the number of instances m increases, the density of spatial datasets increases, and the number of edges also increases. Note that in Fig. 9(a) when the value of m is 100,000, AGSMS has a sharp increase in its execution time, because its computational complexity can reach as much as $O(m^5)$ when the dataset gets dense, while in normal situations its complexity is $O(m^2)$. As the computational complexity of IDS is $O(E^2/m)$, IDS runs faster than GridClique and AGSMC. The computational complexity of NDS is $O(E)$, which is the most efficient method of all. In Fig. 9(b), E increases when *clumpy* increases, thus the execution times of IDS ($O(E^2/m)$), NDS ($O(E)$) and GridClique ($O(E^2)$) increase. NDS is the most efficient, and GridClique is the most costly ($E^2 > E^2/m > E$). For AGSMC, although its computational complexity is not related to E, the increase of E can make the

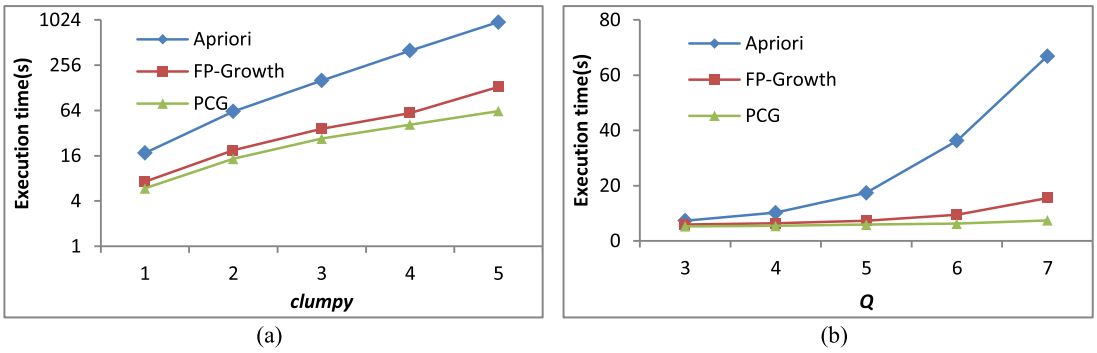


Fig. 10. Effects of prevalent co-location generation.

Table 3
Effects of the clique-based approach.

method	join-based		join-less		clique-based(IDS)		clique-based(NDS)	
data type / factor (%)	sparse	dense	sparse	dense	sparse	dense	sparse	dense
T_materNei	-	-	6.1	3.1	1.8	4.1	3	9.2
T_cliqueGen	-	-	-	-	64.3	74.1	74.2	80.1
T_candiGen	2.1	0.2	3.5	0.5	9.8	5.7	10.1	4.4
T_instanceFilter	93.6	98.7	82.6	94.8	-	-	-	-
T_prevFilter	4.3	1.1	7.8	1.6	24.1	16.1	12.7	6.3
Total execution time(s)	26.9	598.6	19.2	212.9	29.2	158.3	21.5	89.4
Total space cost(M)	159	1978	121	954	178	324	291	578

*Execution time is sec. Space cost is MByte. Other data values are %

size of neighbors of each instance increase, and because of many intersection operations in AGSMC, its execution time also increases. Thus, our proposed clique generation methods are shown to be efficient.

4.2.3. Comparison of prevalent co-location generation

We examined the efficiency of our prevalent co-location generation method (PCG for short) over Apriori [1] and FP-Growth [15] with different core co-location sizes Q and *clumpy*. As shown in Fig. 10, Apriori is the most time consuming because it needs to traverse all transactions for each candidate, while FP-Growth and PCG traverse only once to build FP-tree and C-hash respectively. When *clumpy* increases smoothly, the number of cliques increases rapidly. Thus, the execution times of the three methods increase as shown in Fig. 10. When the average number of core co-location size Q increases, the reasons why the execution times of the three methods increase are as follows. (1) In Apriori, the increase of the average size of transaction-type data makes it cost more time on traversing. (2) In FP-Growth, the increase of the average size of transaction-type data makes the FP-tree deeper, needing more time to operate FP-Growth. (3) In PCG, the increase in the average size of cliques increases the average size of candidates, which requires more time to filter prevalent co-location patterns. The main reason why PCG runs faster than FP-Growth is that FP-Growth is a recursive procedure, and when FP-tree gets bigger and deeper, this kind of procedure cannot perform well. Thus, our prevalent co-location generation method is efficient.

4.2.4. Comparison of the clique-based approach

We examined the efficiency of the clique-based approach over join-based and join-less. Two synthetic datasets with different densities ($D = 1000$ and 10,000, respectively) were generated using the parameter settings according to Table 2. As shown in Table 3, in join-based and join-less, a large fraction of computational time is devoted to filtering co-location instances, while in the clique-based approach the major cost of computation is clique generation. In sparse datasets the gaps of execution times among the four approaches are small. The execution time of the clique-based approach is more than join-based and join-less because the clique-based approach is to find complete cliques regardless of min_*prev*, and in sparse datasets many candidate co-locations may be pruned because of their low PI value, whilst in our approach they are

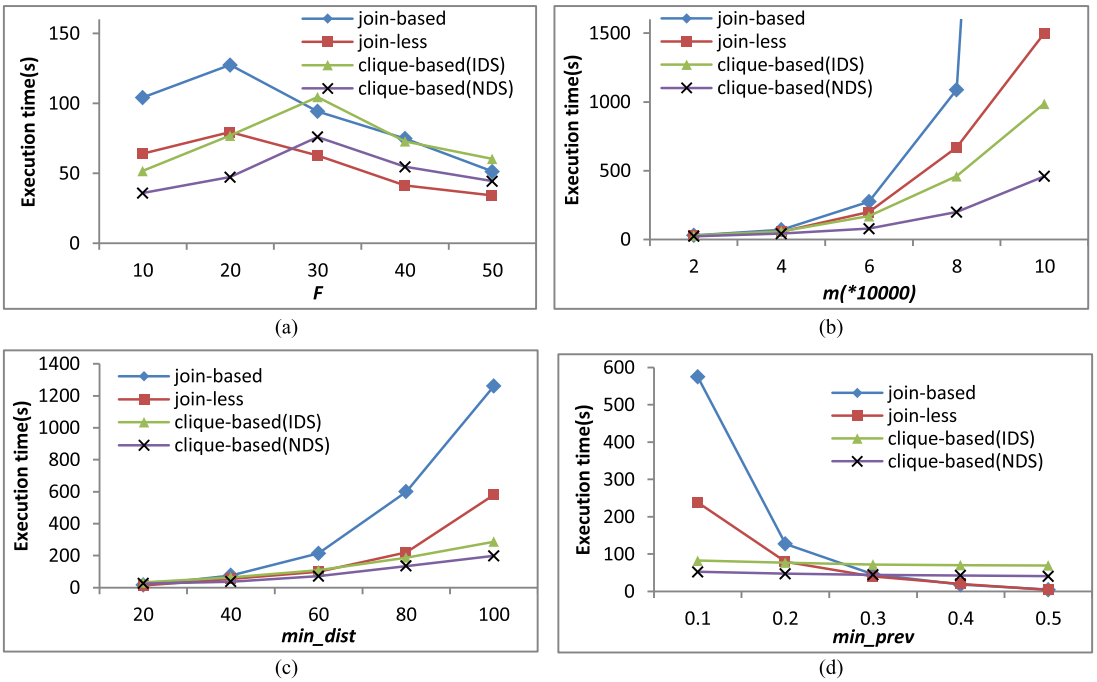


Fig. 11. Scalability of the clique-based approach.

kept in C-hash. In dense data, our approach is more efficient than join-less and join-based because the dense data makes the average size of prevalent co-locations long, and most candidate co-locations in join-based and join-less are prevalent, making their pruning strategies less effective, meaning that row-instance checking operations are necessary. The space cost was also examined as shown in Table 3. In a quite sparse dataset, the reason why the space costs of join-based and join-less are less than the clique-based approach is that both the average number of row-instances and the average size of prevalent co-locations are low while our approach discovers all complete cliques from a spatial dataset. In dense datasets, our clique-based approach costs much less memory space than join-based and join-less because of the effective condensing of cliques (C-hash). Thus, our proposed clique-based approach is efficient.

4.3. Scalability of the clique-based approach

4.3.1. Effect of the number of features

First, we compared the effect of the number of features on the execution time. As shown in Fig. 11(a), with the increase of the number of features, the execution times of the four algorithms first increase then decrease, because in join-based and join-less, the number of candidates increases when the number of features increases from 10 to 20, and the co-location instance checking operations increase. When the number of features increases continually, the average size of candidate co-locations and the average number of neighbors of each instance decrease, as both join-based and join-less can efficiently discover prevalent co-locations because of their pruning strategies on the downward closure of prevalent co-location patterns [16]. For a clique-based approach, the number of edges determines its execution time; when the number of features increases from 10 to 30, the number of edges increases; when the number of features increases from 30 to 50, the number of edges decreases. From Fig. 11(a), it can be seen that join-less is more efficient than the other three algorithms when $F > 30$ because many candidate co-locations can be pruned early by its pruning strategy while our approach is to find all complete cliques, but the time gaps of the four algorithms are not outstanding. In datasets which are not sparse (when $F = 10$ and 20 in Fig. 11(a)), our approach is more efficient than join-based and join-less.

4.3.2. Effect of the number of instances

In the second experiment, we compared the performance as a function of different numbers of instances. The results are shown in Fig. 11(b). With the increase of the number of instances, all the four algorithms increase their execution times, because the increased number of instances makes the dataset become dense. Note that when $m = 100{,}000$, join-based approach takes excessive time (more than one hour). Our approach performs much more efficiently than join-based and join-less because the dense dataset makes the average size of prevalent co-locations, as well as the average number of row-instances, large. These effects incur massive amounts of time to identify row-instances in join-based, and filtering instances from the neighborhood list in join-less. In our approach, efficient clique generation methods work well in avoiding the row-instance identification of each candidate, and the prevalent generation process is efficient using the C-hash structure.

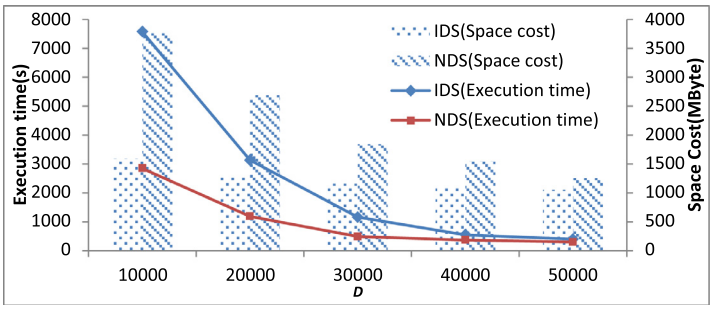


Fig. 12. The behavior of clique generation schemas.

4.3.3. Effect of the neighbor distance threshold

The third experiment examined the effect of different neighbor distance thresholds on the execution time. As shown in Fig. 11(c), with the increase of neighbor distance thresholds, the execution times of all the four algorithms increase, because the increased number of edges makes the dataset become dense. Our approach shows less increase in the execution time with the increase of distance thresholds, while join-based and join-less show a rapid increase, since the increase of neighbor distances makes the neighborhood areas larger and also increases the average number of row-instances per co-location.

4.3.4. Effect of the prevalence threshold

In the final experiment of scalability, we examined the performance effect as the prevalence threshold increases. Overall, the execution time decreases for all the four algorithms, as shown in Fig. 11(d). The reason why the execution times of join-based and join-less decrease is that, with a higher prevalence threshold, the two algorithms can prune more candidate co-locations. For our clique-based approach, the prevalence threshold only participates in the prevalent filtering process (Section 3.4), and with increases in the prevalence threshold, the number of candidate decreases, which makes the execution time of the clique-based approach decrease slowly. When min_*prev* > *0.3*, the execution times of join-based and join-less are less than the clique-based approach because the number of prevalent co-locations with high PI value is much less than those with low PI value, thus, with the downward closure of prevalent co-locations the two algorithms can perform efficiently. Overall, the stability of execution time of the clique-based approach with different prevalence thresholds makes the discovery of appropriate prevalence threshold effective.

4.4. Behavior of the clique generation schemas

We examined the behavior of IDS and NDS with different densities of the spatial dataset. The experimental settings can be found in Table 2. Fig. 12 shows the execution time and space cost of the two schemas, respectively.

- **Execution time behavior.** As the area size D increases, the execution times of the two schemas decrease because the dataset gets sparser as D increases. It can be seen from Fig. 12 that NDS runs much faster than IDS, especially in dense datasets, because IDS builds an I-tree for each instance from the spatial data, and to generate children nodes for a node n representing an instance s, IDS has to search $BNs(s)$ for instance s and get $RS(n)$ from the *node-link* fields, and then make an intersection operation on $BNs(s)$ and $RS(n)$, an operation which it performs on each node on the I-tree except leaf nodes causing multiple search operations on the neighborhood list. In comparison NDS only traverses the neighborhood list once, which makes it much more efficient than IDS on time performance.
- **Space cost behavior.** As the area size D increases, both of the two schemas decrease in space cost because the number of cliques generated decreases. IDS uses a constructing-while-pruning way that makes it have more effective space performance than NDS. Suppose the number of instances is m, and the instances set is $\{s_1,\ldots,s_m\}$, thus, for an I-tree, the space cost is $\max_{i=1}^{m}\{sp(hn_{s_i})\}$, where $sp(hn)$ is the space cost of a head-node hn and all its descendants. In NDS, the space cost is $\sum_{i=1}^{m}\{sp(hn_{s_i})\}$, although for each head-node hn, $sp(hn)$ in IDS is not more than that in NDS (Lemma 6), IDS still performs much better than NDS on space cost, especially in dense datasets.

From the above discussions based on experimental results, the time performance of NDS is better than IDS, while the space performance of IDS is better than NDS. Because IDS can be easily transformed into a parallel algorithm, in limited memory, IDS can deal with much bigger and denser spatial datasets than NDS.

4.5. Flexibility of the clique-based approach

In this subsection, we examined the flexibility of the clique-based approach over join-based and join-less. An appropriate prevalence threshold is very important for a decision maker because a low prevalence threshold may contain many redundant (known) prevalent co-location patterns (e.g., PI < 0.1) while a high prevalence threshold may lose some rare (with

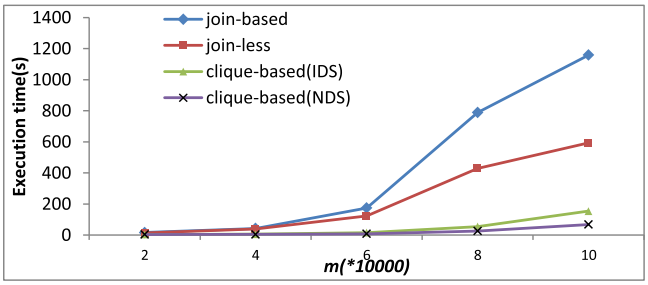


Fig. 13. Flexibility of the clique-based approach.

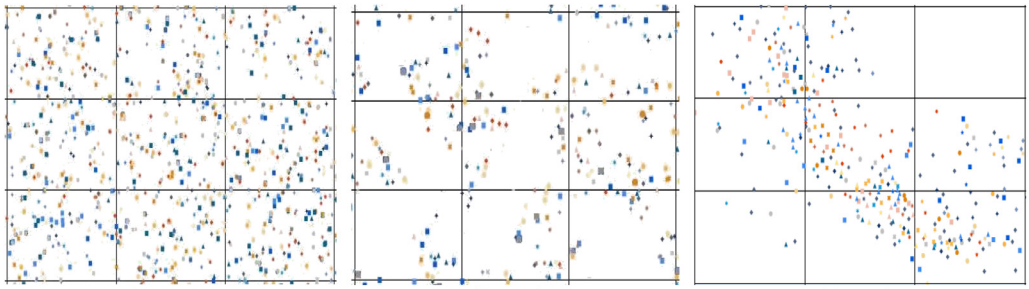


Fig. 14. Part of the visualization of distributions of three real datasets.

Table 4
A summary of the three real datasets.

Name	N. of features	N. of instances	(Max, Min)	The distribution area of instances (m)
Real-1	20	377,834	(60,000,347)	50,000 × 80,000
Real-2	15	501,046	(55,646,8706)	110,000 × 160,000
Real-3	32	335	(63,3)	80,000 × 130,000

(Max, Min): are respectively the maximum number and the minimum number of the feature's instances in the datasets.

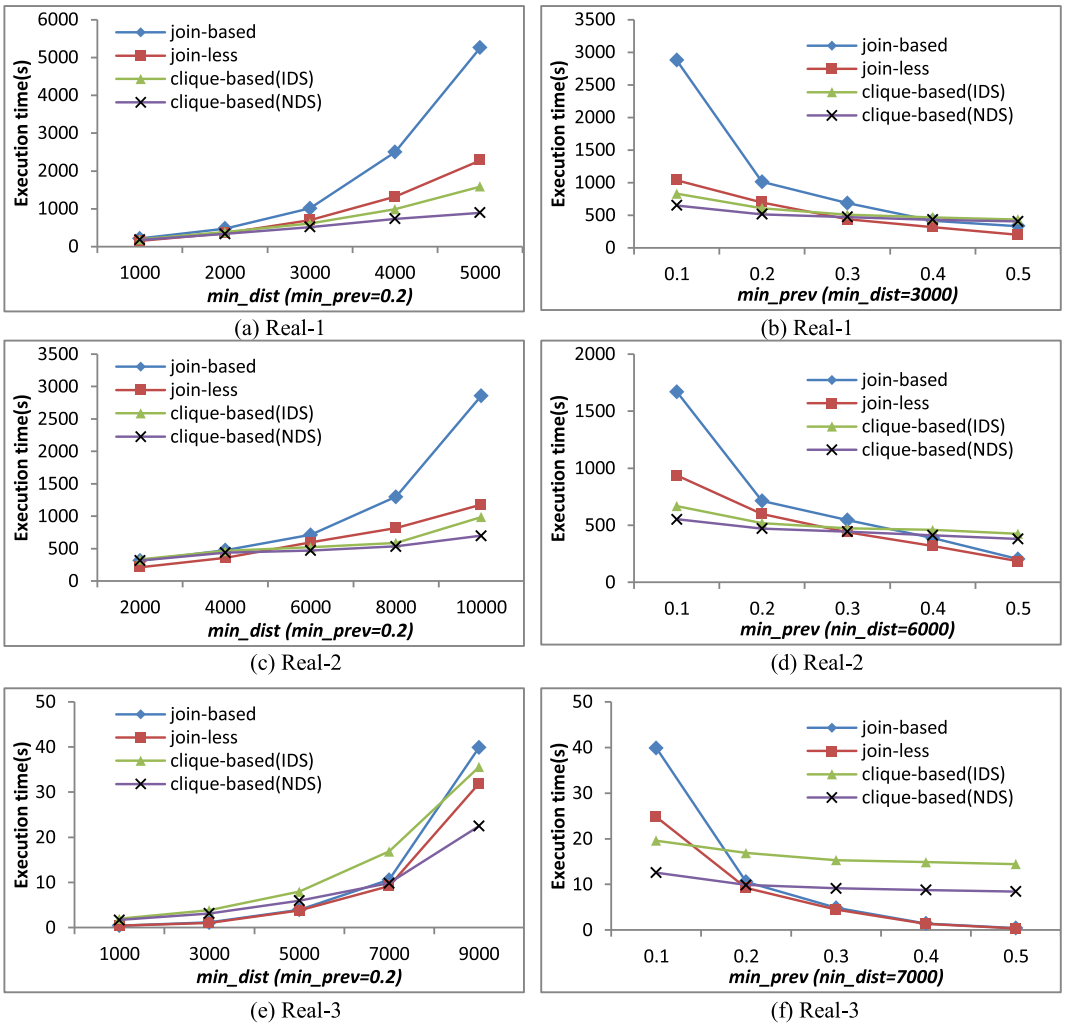
low PI value but valuable) co-location patterns (e.g., PI > 0.6). Besides, for the same spatial datasets, there exist no stable prevalence thresholds that can make each user satisfied. Thus, it is necessary to present an effective method to help the user find an appropriate prevalence threshold.

In order to examine the flexibility of the clique-based approach, different numbers of instances were used, and the parameter settings can be found in Table 2. For each number of instances, we randomly vary the number of prevalence thresholds from 0.1 to 0.6 for 20 times, and the average execution time is recorded for each number of instances. The results can be found in Fig. 13. The average execution time of clique-based approach is much less than that of join-based and join-less because once C-hash is constructed for a spatial dataset, from the second time of evaluation onwards, the prevalent co-location patterns can be generated from C-hash only. Thus, for the following evaluations, the execution time of clique-based approach is almost exclusively due to prevalent co-location generation (Sections 3.3 and 3.4), but join-based and join-less have to restart once the prevalence threshold changes. Thus, our proposed clique-based approach is more flexible than join-based and join-less on searching an appropriate prevalence threshold.

4.6. On real datasets

In this subsection, we examined the performance of clique-based approach over join-based and join-less using three real datasets. The first real dataset (Real-1 for short) is a spatial distribution dataset of POI data in Beijing with a both even and dense distribution as shown in Fig. 14(a), the second real dataset (Real-2 for short) is a vegetation distribution dataset of the Three Parallel Rivers of Yunnan Protected Area, whose instances form various clusters as shown in Fig. 14(b), and the last real dataset (Real-3 for short) is from the rare plant data of the Three Parallel Rivers of Yunnan Protected Area whose instances form a zonal distribution as shown in Fig. 14(c). Real-1 is the densest and Real-3 is the sparsest. A summary of the selected three real datasets is presented in Table 4.

Fig. 15 shows the experimental results on the three real datasets with different neighbor distance thresholds and prevalence thresholds, respectively. Fig. 15(a), (c) and (e) show the execution times of the four algorithms with different neighbor distance thresholds in Real-1, Real-2, and Real-3, respectively. With the increase of the neighbor distance threshold the exe-



(a) Real-1 (b) Real-1

(c) Real-2 (d) Real-2

(e) Real-3 (f) Real-3

Fig. 15. Experiments on real data.

cution times of the four algorithms increases. The clique-based approach increases most slowly, and when the dataset gets dense enough (when min_*dist* = 5000 in Fig. 15(a)), the clique-based approach using NDS schema runs twice as fast as join-less and more than 5 times faster than join-based. In a sparse dataset like Real-3, join-less and join-based run faster than IDS and NDS, but the gap is not large ((when min_*dist* < 7000 in Fig. 15(e))). Fig. 15(b), (d), and (f) show the execution times of the four algorithms with different prevalence thresholds in Real-1, Real-2, and Real-3, respectively. With the increase of the prevalence threshold, all the four algorithms decrease their execution times, although a high prevalence threshold may make the clique-based approach take more time than join-based and join-less, but a high prevalence threshold has a greater chance of losing interesting co-location patterns. Thus, our proposed clique-based approach is effective and efficient.

5. Conclusion

In traditional works on prevalent co-location pattern mining, most of the computational time is devoted to identifying row-instances of co-location patterns. In this paper, we have presented a novel and efficient clique-based approach to discover prevalent co-location patterns without identifying row-instances, as all prevalent co-location patterns can be calculated using the proposed C-hash structure which is a condensed structure for cliques generated by our approach. Thus, our proposed clique-based approach is efficient, especially in dense datasets. Further, our approach is much more flexible than traditional mining approaches in finding an appropriate prevalence threshold. When the prevalence threshold is changed, traditional approaches must rediscover prevalent co-location patterns entirely, while our approach only needs to restart the process of prevalent co-location generation, effectively avoiding the process of generating cliques which is the main cost of our clique-based approach. Although the traditional co-location pattern mining approaches using maximal cliques can also find a proper prevalence threshold, as does our proposed approach, the generation of maximal cliques is quite expensive. In

order to adapt to different densities of spatial datasets, two schemas of clique generation (IDS and NDS) were presented in this paper, which make our clique-based approach robust to the density of spatial datasets.

Due to globalization and digitization, both geographical search space and the amount of spatial data, has enormously increased, and this exponential growth of spatial data may limit the performance of our proposed approach in the future, due to the limitations of memory or processor performances. Thus, there are several interesting directions to plan in our future work: (1) more effective pruning or optimizing strategies can be investigated to make our clique-based approach more efficient, especially for the generation of complete and correct cliques. (2) in IDS, the process of getting H_s-Cliques for an instance s is independent with other instances. Thus, it is quite easy to transform IDS into an efficient parallel or distributed program Using GPUs (Graphic Processing Units) to implement the scalability of our approach for handling large-scale spatial datasets [3,8,12]. (3) other different kinds of spatial datasets may be used to validate our proposed approach, such as fuzzy spatial datasets [19], or uncertain datasets [23].

Conflict of interest

We declare that we do not have any commercial or associative interest that represents a conflict of interest in connection with the work submitted.

Acknowledgments

This work was supported by grants (No. 61472346, No. 61662086) from the National Natural Science Foundation of China, by grants (No. 2016FA026, No. 2015FB114) from the Science Foundation of Yunnan Province and by the Project of Innovation Research Team of Yunnan Province (No. 2018HC019).

References

[1] R. Agarwal, R. Srikant, Fast algorithms for mining association rules, in: Proc. VLDB, 1994, pp. 487–499.
[2] G. Al-Naymat, Enumeration of maximal clique for mining spatial co-location patterns, in: Proc. ICCSA, 2008, pp. 126–133.
[3] W. Andrzejewski, P. Boinski, Parallel approach to incremental co-location pattern mining, Inf. Sci. (2018). https://doi.org/10.1016/j.ins.2018.09.016.
[4] X. Bao, L. Wang, Discovering interesting co-location patterns interactively using ontologies, in: Proc. DASFAA Workshop, 2017, pp. 75–89.
[5] X. Bao, L. Wang, Q. Xiao, OICPM: an interactive system to find interesting co-location patterns using ontologies, in: Proc. APWeb-WAIM, 2017, pp. 329–332.
[6] A. Berry, R. Pogorelcnik, A simple algorithm to generate the minimal separators and the maximal cliques of a chordal graph, Inf. Process. Lett. 111 (11) (2011) 508–511.
[7] C. Bron, J. Kerbosch, Algorithm 457: finding all cliques of an undirected graph, Commun. ACM 16 (9) (1973) 575–577.
[8] A. Cano, A survey on graphic processing unit computing for large-scale data mining, Wiley Interdiscip. Rev. 8 (1) (2018) e1232.
[9] D. Cavaliere, S. Senatore, V. Loia, Context-aware profiling of concepts from a semantic topological space, Knowl.-Based Syst. 130 (2017) 102–115.
[10] I. Chiang, C. Liu, Y. Tsai, et al., Discovering latent semantics in web documents using fuzzy clustering, IEEE Trans. Fuzzy Syst. 23 (6) (2015) 2122–2134.
[11] N.S. Dasari, R. Desh, M. Zubair, pbitMCE: a bit-based approach for maximal clique enumeration on multicore processors, in: ICPADS, 2014, pp. 478–485.
[12] Y. Djenouri, D. Djenouri, A. Belhadi, A. Cano, Exploiting GPU and cluster parallelism in single scan frequent itemset mining, Inf. Sci. (2018). https://doi.org/10.1016/j.ins.2018.07.020.
[13] D. Eppstein, M. Löffler, D. Strash, Listing all maximal cliques in sparse graphs in near-optimal time, in: Proc. Algorithms and Computation, Berlin, Heidelberg, Springer, 2010, pp. 403–414.
[14] L. Feng, L.Z. Wang, S.J. Gao, A new approach of mining co-location patterns in spatial datasets with rare features, J. Nanjing Univ. 48 (1) (2012) 99–107.
[15] J. Han, J. Pei, Y. Yin, Mining frequent patterns without candidate generation, in: Proc. ACM SIGMOD, 2000, pp. 1–12.
[16] Y. Huang, S. Shekhar, H. Xiong, Discovering colocation patterns from spatial datasets: a general approach, IEEE Trans. Knowl. Data Eng. 16 (12) (2004) 1472–1485.
[17] S. Kim, Y. Kim, U. Kim, Maximal cliques generating algorithm for spatial co-location pattern mining, in: Proc. FTRA International Conference on Secure and Trust Computing, Data Management, and Application, 2011, pp. 241–250.
[18] Y. Morimoto, Mining frequent neighboring class sets in spatial databases, in: Proc. ACM SIGKDD, 2001, pp. 353–358.
[19] Z. Ouyang, L. Wang, P. Wu, Spatial co-location pattern discovery from fuzzy objects, Int. Artif. Intell. Tools 26 (2) (2017) 1750003, doi:10.1142/S2018213017500038.
[20] S. Rinzivillo, F. Turini, Extracting spatial association rules from spatial transactions, in: Proc. ACM IWGIS, 2005, pp. 79–86.
[21] V. Stix, Finding all maximal cliques in dynamic graphs, Comput. Optim. Appl. 27 (2) (2004) 173–186.
[22] F. Verhein, G. Al-Naymat, Fast mining of complex spatial co-location patterns using GLIMIT, in: Proc. IEEE ICDM, 2007, pp. 679–684.
[23] L. Wang, J. Han, H. Chen, Finding probabilistic prevalent co-location mining in spatially uncertain datasets, IEEE Trans. Knowl Data Eng. 25 (4) (2013) 790–804.
[24] L. Wang, L. Zhou, J. Lu, et al., An order-clique-based approach for mining maximal co-locations, Inf. Sci. 179 (2009) 3370–3382.
[25] L. Wang, X. Bao, L. Cao, Interactive probabilistic post-mining of user-preferred spatial co-location patterns, in: Proc. ICDE, 2018, pp. 1256–1259.
[26] L. Wang, X. Bao, H. Chen, et al., Effective lossless condensed representation and discovery of spatial co-location patterns, Inf. Sci. 436 (2018) 197–213.
[27] L. Wang, X. Bao, L. Zhou, Redundancy reduction for prevalent co-location patterns, IEEE Trans. Knowl. Data Eng. 30 (1) (2018) 142–155.
[28] X. Wang, L. Wang, J. Lu, et al., Effectively updating high utility co-location patterns in evolving spatial database, in: Proc. WAIM, 2016, pp. 67–81.
[29] X. Yao, L. Peng, L. Yang, et al., A fast space-saving algorithm for maximal co-location pattern mining, Expert Syst. Appl. 63 (2016) 310–323.
[30] J.S. Yoo, S. Shekhar, A partial join approach for mining co-location patterns, in: Proc. ACM IWGIS, 2004, pp. 241–249.
[31] J.S. Yoo, S. Shekhar, M. Celik, A join-Less approach for co-location pattern mining: a summary of results, in: Proc. IEEE ICDM, 2005, pp. 813–816.
[32] J.S. Yoo, M. Bow, Mining top-k closed co-location patterns, in: Proc. IEEE International Conference on Spatial Data Mining and Geographical Knowledge Services, 2011, pp. 100–105.