Mining Co-location Patterns in Incremental Spatial Databases

Ye-In Chang
Dept. of Computer Science and Engineering
National Sun Yat-Sen University
Kaohsiung, Taiwan
changyi@cse.nsysu.edu.tw

Chen-Chang Wu
Center for Fundamental Science
Kaohsiung Medical University
Kaohsiung, Taiwan
wucc5501@kmu.edu.tw

Ching-Yi Yen
Dept. of Computer Science and Engineering
National Sun Yat-Sen University
Kaohsiung, Taiwan
yanjy@db.cse.nsysu.edu.tw

***Abstract*—An incremental database can be widely used in many areas due to the changes in data over time, including the location-based services (LBS), environmental ecology, and also the business behavior patterns. Looking for the spatial co-location pattern that appears frequently nearby over an incremental database has become an interesting and essential topic. Many spatial co-location pattern mining approaches are for traditional spatial databases. Therefore, they do not need to consider candidate instances generated and update their participation index in the process. Yoo $et\ al.$ have proposed the EUCOLOC algorithm to mine co-location patterns in the incremental database. The EUCOLOC algorithm not only needs large storage to store points in the database and their relationships with each other, but also generates many unnecessary candidate instances. In this paper, we propose an approach to mine the co-location patterns for incremental database. Our approach can avoid generating non-incremental candidate instances and non-clique instances. Moreover, we also avoid storing the duplicated data. From our experimental results, we show that our method performed more efficiently than the EUCOLOC algorithm.**

***Index Terms*—Incremental Database, Spatial Co-location Patterns, Spatial Co-location Rules, Spatial Database, Spatial Data Mining**

I. INTRODUCTION

Spatial co-location patterns mining is a new branch of spatial data mining, which is the process of discovering interesting and implicit, but potentially valuable patterns from spatial databases. In recent years, the huge quantity spatial data shows that it is increasingly important and difficult to discover co-location patterns with valuable information. It is essential to extract what we want from geo-spatial data and apply it to many domains. Furthermore, we can make decisions based on large spatial datasets by using the extracted information.

There are many approaches to discover co-location patterns in spatial data mining. Huang $et\ al.$ propose the full-join approach [1] which is the Apriori-like method. It can have good performance for the sparse spatial dataset, but it is inefficient for the dense spatial dataset. As the increased number of the co-location patterns, it needs the long computation time for co-location pattern mining. Huang and Shekhar propose the partial-join approach [2] and the join-less approach [3]. Nevertheless, the methods mentioned above may be restricted to short length patterns or sparse spatial database. However, as the amount of data increased rapidly, the number of long length patterns which often present with large or dense spatial database also increases. For the purpose of resolving the performance under a dense dataset and the long prevalence co-location patterns, the maximal co-location pattern mining was developed [4]–[7].

In addition to approaches of mining co-location patterns which we have mentioned above, there are some other useful methods in co-location patterns mining. For example, there are some researchers who focus on the spatial co-location mining without the threshold which is pre-determined by users [8], [9]. Furthermore, there are also some researchers who study about the Top-k closed co-location patterns [10], [11] and the other topics are about mining maximal cliques [12], [13], maximum frequent itemsets and spatial high utility co-location patterns [14].

Recently, spatial co-location pattern mining in the incremental database [15]–[17] is a new issue in the spatial mining field. The approaches of mining spatial co-location patterns which we have mentioned above are suitable for the static database situation. Once new data is inserted into the database, those approaches will take a lot of time to scan the database for several times and even need to reconstruct the data structures. However, in the real world, we can find that the data information in various fields will increase over the time. For example, development of business, biological symbiotic species, crime activity and interdependent events such as traffic jam, car accidents , policemen and ambulances in transportation [17], [18]. For the above reasons, it is not enough effective for previous algorithms to deal with incremental databases for mining spatial co-location patterns. Therefore, in [17], Yoo $et\ al.$ propose the EUCOLOC algorithm to generate candidate instances and update the co-location patterns in the incremental database. However, their method has same problems which include that it stores the data twice or even many times and generates redundant and useless candidate instances ($i.e$ non-incremental candidate instances and non-clique candidate instances).

Therefore, in this paper, we propose a method to update the co-location patterns and then mine the database. There are several advantages in our method. First, we propose a new approach to rearrange the relations in order to use less storage to store data information and also can avoid

generating the non-incremental candidate instances. Second, we will check the relation of subsets before generating size-k candidate instances. In this way, we can avoid generating non-clique instances completely. Therefore, our approach can avoid generating non-incremental candidate instances and non-clique instances. Moreover, we also avoid storing the duplicated data. From our experimental results, we show that our method needs less time and generates fewer number of candidate instances than the EUCOLOC algorithm.

The rest of the paper is organized as follows. In Section 2, we give a survey of some well-know approaches for mining spatial co-locations patterns in the spatial database. In Section 3, we present our proposed approach. In Section 4, we present the performance of our approach and make a comparison between our approach and the EUCOLOC algorithm. Finally, we give a conclusion in Section 5.

II. RELATED WORK

In recent years, mining spatial co-location pattern is one of the prominent problems in the spatial data mining field. It is different from the traditional association rule mining problem [19] because of mining spatial co-location pattern without the nature notion of transactions. First, we will describe the bisic concept of spatial co-location mining. Next, we will introduce $EUCOLOC$ algorithm [17] for increamental co-location pattern mining.

A. SPATIAL CO-LOCATION PATTERNS MINING

Fig. 1 shows an example of the spatial data, and we use the spatial example to explain the related definition. Object $E.i$ represents an instance i of event type E. The spatial example contains events A, B, C, with 5, 3, 4 instances for each event, respectively. We can see that if there is a solid line between two different types of objects, then the two objects have a neighbor relationship. It means that $A.1$ has a neighbor relationship with $B.1$, but $B.3$ does not have the neighbor relationship with $B.1$.

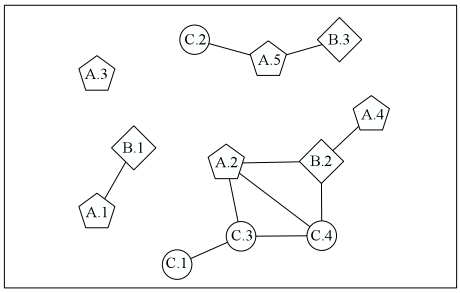


Fig. 1: The definition of Participation Index (PI).

The Participation Index $PI(X)$ of $X = \{E_1, ..., E_k\}$ is defined as $PI(X) = min_{E_i \in X}\{PR(X, E_i)\}$, $1 \leq i \leq k$ [3]. The Participation Ratio $PR(X, E_i)$ is defined as $PR(X, E_i) = \frac{Number\ of\ distinct\ objects\ of\ E_i\ in\ instances\ of\ X}{Number\ of\ objects\ of\ E_i}$. If $PI(X)$ is greater than a given minimum prevalent threshold, X is a *prevalent co-location*. In Fig. 1, $(A.2, C.3)$, $(A.2, C.4)$ and $(A.5, C.2)$ are neighbor relations. There are two distinct instances $A.2$ and $A.5$ in the co-location $\{A, C\}$, so we can calculate $PR(\{A, C\}, A) = \frac{2}{5}$. Similarly, in the co-location $\{A, C\}$, there are three distinct instances $C.2$, $C.3$ and $C.4$, so we can calculate $PR(\{A, C\}, C) = \frac{3}{4}$. Therefore, $PI(\{A, C\})$ is $\frac{2}{5}$, which is the minimum value between $PR(\{A, C\}, A)$ and $PR(\{A, C\}, C)$.

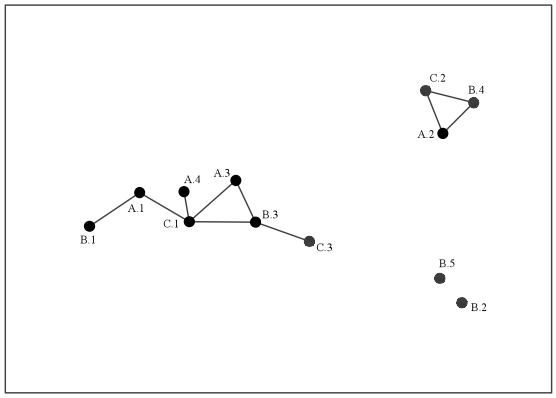
B. INCREMENTAL SPATIAL CO-LOCATIONS MINING

The problem of updating spatial co-location patterns presents more challenges than updating frequent itemsets in a traditional transaction database. In classic association analysis, database updates mean simply adding new transaction records, or deleting existing ones. The newly added transaction records are separately handled from existing records because the database is a collection of disjoint transaction records. In contrast, when a spatial database is updated, the new data point can establish neighbor relationships with existing data points and other new data points in continuous space. Therefore, all neighbor relationships in the updated database should be checked to maintain co-location patterns [17].

Let $S_{old} = O_1, ..., O_n$ be a set of old data points and $S_{inc} = O_{n+1}, ..., O_{n+h}$ be a set of new data points added in the database. Let S be all data points in the updated database, $S = S_{old} \cup S_{inc}$. There are two types of co-location instances in the updated database. The retained co-location is a prevalent event set in both S_{old} and S. The emerged co-location is an event set which is not prevalent in S_{old} but is prevalent in S. They propose an Efficient Update algorithm for the COLOCation patterns (EUCOLOC algorithm) with the addition of spatial data points [17].

In Fig. 2-(a), the original spatial database has three different event types, A, B and C. The total number of instances of each event type, A, B and C is 4, 5, 3 respectively. Fig. 2-(b) shows the neighbor objects that the instance in the first column has the neighbor relationship with the instance in the second column. For example, object A.1 has the neighbor relationship with object B.1 and object C.1, but they can not sure that object B.1 and object C.1 have neighbor relationship. After they insert new data points, A.5, B.6, B.7, B.8, B.9 and C.4, C.5, they can get the incremental data points and their relationships in Fig. 2-(c) and the incremental neighbors database in Fig. 2-(d). They denoted '*' for new data points. Then, they union the old neighbors database and incremental neighbors database which can get the updated neighbors database in Fig. 3-(b). Later, they use the incremental neighbors database to generate candidate instances and use the updated neighbors to check the relation of the subset.

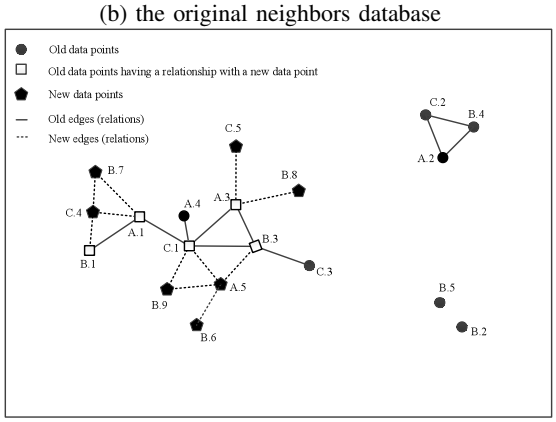
They use the A-incremental neighborhood transactions in incremental neighbors database to generate candidate instance [17]. They generate 11 candidate instances, including 2 non-incremental instances and 5 non-clique instances. In Fig. 3-(a), we can find the non-incremental instances are {A.1, B.1, C.1} and {A.3, B.3, C.1}. The non-clique instances are {A.1, B.7*, C.1}, {A.3, B.3, C.5*}, {A.3, B.8*, C.1}, {A.3, B.8*, C.5} and {A.5, B.6*, C.1}. If the points of the candidate instance all exist in the original neighbor database, they will become non-incremental instance. Otherwise, they have to use the updated



(a) old data points and their relationships

Neighbor objects	
A.1	B.1, C.1
A.2	B.4, C.2
A.3	B.3, C.1
A.4	C.1
B.1	
B.2	
B.3	C.1, C.3
B.4	C.2
B.5	
C.1	
C.2	
C.3	

(b) the original neighbors database



(c) incremental data points and their relationships

Neighbor objects	
A.1	B.1, B.7*, C.1, C.4*
A.3	B.3, B.8*, C.1, C.5*
A.5*	B.3, B.6*, B.9*, C.1
B.1	C.4*
B.6*	
B.7*	C.4*
B.8*	
B.9*	C.1*
C.4*	
C.5*	

(d) the incremental (changed and new*) neighbors database

Fig. 2: An example of original database and incremental spatial database.

A-incremental neighborhood transactions	
A.1	B.1, B.7*, C.1, C.4*
A.3	B.3, B.8*, C.1, C.5*
A.5*	B.3, B.6*, B.9*, C.1

~~{A.1, B.1, C.1}~~ non incremental instance
{A.1, B.1, C.4*}
~~{A.1, B.7*, C.1}~~ non clique instance
{A.1, B.7*, C.4*}
~~{A.3, B.3, C.1}~~ non incremental instance
~~{A.3, B.3, C.5*}~~ non clique instance
~~{A.3, B.8*, C.1}~~ non clique instance
~~{A.3, B.8*, C.5*}~~ non clique instance
{A.5*, B.3, C.1}
~~{A.5*, B.6*, C.1}~~ non clique instance
{A.5*, B.9*, C.1}

instance lookup (subset)

(a) generating candidates and checking relation of subset

Neighbor objects	
A.1	B.1, B.7*, C.1, C.4*
A.2	B.4, C.2
A.3	B.3, B.8*, C.1, C.5*
A.4	C.1
A.5*	B.3, B.6*, B.9*, C.1
B.1	C.4*
B.2	
B.3	C.1, C.3
B.4	C.2
B.5	
B.6*	
B.7*	C.4*
B.8*	
B.9*	C.1*
C.1	
C.2	
C.3	
C.4*	
C.5*	

(b) updated neighbors database

Fig. 3: An example of mining incremental spatial database.

database to check the relation of subset. For example, {A.1, B.7*, C.4*}, they have to check whether if there is a neighbor relationship between B.7* and C.4*. If B.7* and C.4* have neighbor relationship, then they call it as the clique instance; otherwise, they call it as the non-clique instance.

III. PROPOSED METHOD

In this section, we use one example of the spatial dataset to illustrate our method. In Fig. 4, the original database, which contains 17 points and several relations (edges) in the database. These points are composed of five different event types.

When new data points are inserted, many new relations (edges) will also be added at this time. For the example shown in Fig. 5, nine new data points and seventeen new relations (edges) are inserted, where the dotted lines represent the new relations in the incremental database and the solid lines represent the old relations in the original database. In additional to the relations, there are three types of points, where squares represent the old data points having relationships with the new data points in the incremental database, pentagons represent new data points and circles represent the old data points.

Input:
1. A set of spatial event types $ES = \{E_1, E_2, ..., E_n\}$.

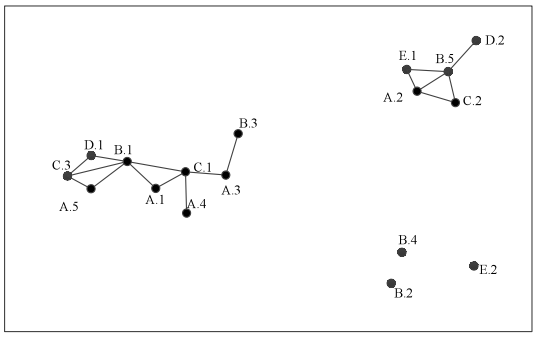


Fig. 4: Old data points and their relationships in the original database.

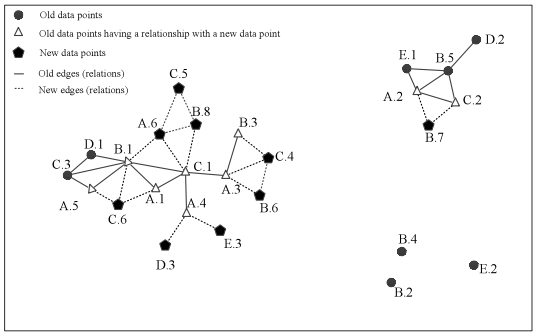


Fig. 5: Incremental data points and their relationships.

Event type $ES = \{A, B, C, D, E\}$.

2. A set of spatial instances $IS = I_1 \cup I_2 \cup ... \cup I_n$, where $I_k (1 \leq k \leq n)$ is a set of instances of event E_k.
   Spatial dataset $IS = \{A.1, A.2, A.3, A.4, A.5, B.1, B.2, B.3, B.4, B.5, C.1, C.2, C.3, D.1, D.2, E.1, E.2\}$
3. The set of relations of size-2 instances.
4. The minimum prevalence threshold Min_prev.
5. Inserting new spatial instances (Points, 9 points)
   $\{A.6, B.6, B.7, B.8, C.4, C.5, C.6, D.3, E.3\}$
6. Inserting new edges (Relations, 17 edges).

Fig. 6 shows all instances of each type A, B, C, D and E and its related count, old data points and new data points after inserting new data points. Here, we mark the new data points with '*'. For example, event C has six instances in the incremental database, where C.1, C.2, C.3 are old data points and C.4, C.5, C.6 are new data points with an asterisk annotation.

Event	Count	Old data points	New data points
A	6	A.1, A.2, A.3, A.4, A.5	A.6*
B	8	B.1, B.2, B.3, B.4, B.5	B.6*, B.7*, B.8*
C	6	C.1, C.2, C.3	C.4*, C.5*, C.6*
D	3	D.1, D.2	D.3*
E	3	E.1, E.2	E.3*

Fig. 6: Instance sets of each spatial event (IS) after inserting new data points.

Inserting new relations of size-2 Instances (Edges)			
(A.6*, B.8*)	(A.1, C.6*)	(A.4, D.3*)	(B.1, C.6*)
(A.6*, C.5*)	(A.2, B.7*)	(A.4, E.3*)	(B.3, C.4*)
(B.6*, C.4*)	(A.3, B.6*)	(A.5, C.6*)	(B.7*, C.2)
(B.8*, C.5*)	(A.3, C.4*)	(A.6*, B.1)	(B.8*, C.1)
		(A.6*, C.1)	

Fig. 7: Inserting new relations of size-2 instances ($IRI2$) (17 Edges).

In Fig. 7, $IRI2$ shows the insertion of new relations of size-2 instances. In the first column, we can find that every instances in each pair have asterisk annotation. It represent that the relations (edges) are made by two new data points. Other relations in $IRI2$ are made by one new data point and one original data point.

The whole processing steps are described as follows:

Step 1: ***Rearrange IRI2***

In this step, we rearrange each related instance pair of IRI2. We exchange the position of two instances, if one of the instances has the asterisk annotation.

For example, for relation (A.1, C.6*) in Fig. 7, we exchange the two instances and get (C.6*, A.1). Because instance C.6* has '*' symbol, which represents the new data point. Then, we sort each pair, based on the first position of the instance in the alphabetical order. After the process of rearranging IRI2, the results are shown in Fig. 8.

Rearrange *IRI2*	
(A.6*, B.1)	(C.4*, A.3)
(A.6*, B.8*)	(C.4*, B.3)
(A.6*, C.1)	(C.6*, A.1)
(A.6*, C.5*)	(C.6*, A.5)
(B.6*, A.3)	(C.6*, B.1)
(B.6*, C.4*)	(D.3*, A.4)
(B.7*, A.2)	(E.3*, A.4)
(B.7*, C.2)	
(B.8*, C.1)	
(B.8*, C.5*)	

Fig. 8: Rearranging ($IRI2$) by * and the alphabetical order.

Step 2: ***Construct the relations of the incremental neighbors***

In this step, we construct the table of relations in the incremental neighbors database based on the following policies: (a) Sort elements according to the asterisk annotation priority, then the alphabetical order. (b) Store the incremental neighbor relationships between different types.

We construct a table to store the incremental neighbor relationships between different event types after we rearrange the position of the two instance in each neighbor pair which has the asterisk annotation in the incremental database as shown in Fig. 9. In this table, we can find that there are nine referenced points and seventeen neighbor instances, as compared to the EUCOLOC algorithm which has sixteen referenced points and thirty neighbor instances. However, in the incremental database of our approach, we also insert only those nine new data points and seventeen new relations (edges). Since the table of original neighbors database and incremental neighbors

database are completely non-overlapping, our approach is called *non-overlapping* approach. The updated database can be obtained by just combining the above two databases as shown in Fig. 10.

Event	Referenced point	Neighbor instances
A	A.6*	B.8*, C.5*, B.1, C.1
B	B.6*	C.4*, A.3
	B.7*	A.2, C.2
	B.8*	C.5*, C.1
C	C.4*	A.3, B.3
	C.5*	
	C.6*	A.1, A.5, B.1
D	D.3*	A.4
E	E.3*	A.4

Fig. 9: The incremental neighbors database.

Event	Referenced point	Neighbor instances
A	A.1	B.1, C.1
	A.2	B.5, C.2, E.1
	A.3	B.3, C.1
	A.4	C.1
	A.5	B.1, C.3
	A.6*	B.8*, C.5*, B.1, C.1
B	B.1	C.1, C.3, D.1
	B.2	
	B.3	
	B.4	
	B.5	C.2, D.2, E.1
	B.6*	C.4*, A.3
	B.7*	A.2, C.2
	B.8*	C.5*, C.1
C	C.1	
	C.2	
	C.3	D.1
	C.4*	A.3, B.3
	C.5*	
	C.6*	A.1, A.5, B.1
D	D.1	
	D.2	
	D.3*	A.4
E	E.1	
	E.2	
	E.3*	A.4

Fig. 10: The updated neighbors database.

In Fig. 10, the table is used to store all neighbor relationships between different event types. We can find that there are 26 referenced points, 34 neighbor instances. Morever, the updated neighbors database also be used to check the relations between referenced point and neighbor instances before we generate size-k candidate instances.

Step 3: ***Generate size-k candidate instances***

In this step, the key point of the process is as follows: (a) Sort elements according to the asterisk annotation priority, then the alphabetical order. (b) Check each instance of neighbor instances except the last one and confirm that it is followed by other instances when it is a referenced point.

After we construct the table of the incremental neighbors, we will generate the size-k candidate. First, we will use the referenced point to be the first element and then followed by other neighbor instances behind. In the size-3 case, for example, A.6* will be the first element and two different instances of the set of {B.8*, C.5*, B.1, C.1} will be behind A.6*. There are four candidates of referenced A.6* in the size-3 case, including {A.6*, B.8*, C.1}, {A.6*, B.8*, C.5}, {A.6*, B.1, C.5*} and {A.6*, B.1, C.1}. However, we find that {A.6*, B.1, C.5*} is a non-clique candidate after we check the instances of the subset and we find that B.1 and C.5* do not have relationship in the updated neighbors database. To resolve this problem, we proposed the method, which can avoid generating non-clique candidates, instead of generating non-clique candidates and then delete them soon.

For the above method which we have mentioned, the first key point is that the last element of the third colomn, neighbor instances, in the incremental neighbors database can be ignored in the process of generating candidates of size-k by focusing it as the main role, where k$\geq$3. The reason is that it has no other following neighbor in the column with which the neighbor instance could be considered. For example, for referenced point A.6*, we will consider the combination of (B.8*, C.5*), (B.8*, C.1), (C.5*, B.1) and (B.1, C.1) in addition to the referenced point A.6*. The second key point is that in order to check the combination of elements, for example, (B.8*, C.1, C.5*) exist in the database, $i.e.$, existing the neighbor relationship in the database, we could check updated the neighbors database. That is, we do not have to construct another data structure to store the information and then check it.

Event	Referenced point	Neighbor instances
A	A.6*	B.8*, C.5*, B.1, ~~C.1~~
B	B.6*	C.4*, ~~A.3~~
	B.7*	A.2, ~~C.2~~
	B.8*	C.5*, ~~C.1~~
C	C.4*	A.3, ~~B.3~~
	C.6*	A.1, A.5, ~~B.1~~
~~D~~	~~D.3*~~	~~A.4~~
~~E~~	~~E.3*~~	~~A.4~~

Fig. 11: The checking relation table.

Fig. 11 shows that the incremental neighbors database which has been further processed. We can find that each referenced point corresponds to the last instance of the neighbor instances will be deleted, including C.1, A.3, C.2, B.3, B.1 and so on. Morever, the referenced points which are not followed by any neighbor instances will also be deleted from this table. For example, the referenced points are C.5*, D.3* and E.3*. Each referenced point corresponding to the neighbor instances needs to be checked except the last one before we generate size-k candidate instances.

For example, the referenced point B.6* will combine the neighbor instances C.4* and A.3 to be a candidate, we will check the instance C.4* at first. The instance C.4* is followed by A.3 and B.3, when it is a referenced point in the updated neighbors database. Then, we can get the relation of (C.4*, A.3). Therefore, the result of generating candidate instance {B.6*, C.4*, A.3} is valid.

Let's see another example in Fig. 12, for the referenced

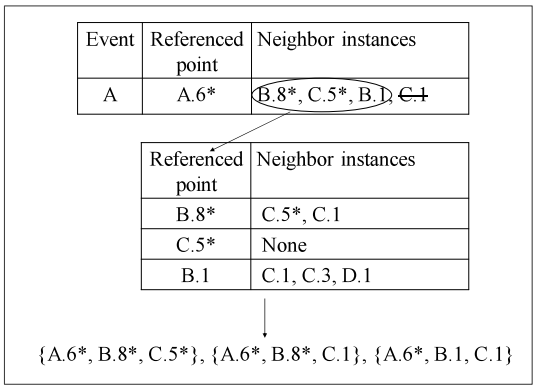


Fig. 12: The example of generating size-k candidate instances of event A.

point A.6*, we will check B.8*, C.5* and B.1 at first. The instance B.8* is followed by C.5* and C.1 when it is a referenced point in the updated database, too. Therefore, we can generate the candidate instances {A.6*, B.8*, C.5*} and {A.6*, B.8*, C.1}. However, C.5* is not followed by any other neighbor instances. We can not generate the candidate instance {A.6*, C.5*, B.1}. Then, we check B.1 and can find that B.1 is followed by C.1, C.3 and D.1. we can get the relation of (B.1, C.1). Therefore, the result of generating candidate instance is {A.6*, B.1, C.1}. After finishing the process of the referenced point of event A, for events B, C, D and E, we follow the above steps and so on.

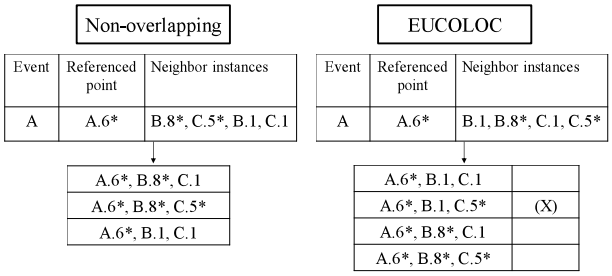


Fig. 13: The comparison of generating size-k candidate instances of A.6*.

Fig. 13 shows the comparison of generating size-k candidate instances of A.6*. We can find that there are only three clique candidate instances in our non-overlapping approach as compared to four candidate instance (including one non-clique candidate instance which will be pruned soon) in the EUCOLOC algorithm. In addition, if the neighbor instances which have the same referenced point and are the same event type, we will not generate size-k candidate. For example, for the referenced point B.8*, there are two neighbor instances C.5* and C.1. If we generate them in the size-3 case, the results will be {B.8*, C.5*, C.1}. It is invalid, because we do not consider relations of the same event type. After generating all candidate instances in Step 3, we can get eight size-3 candidate instances. The resulting table is shown in Fig. 14. The first column of this table represents the referenced point in the incremental neighbors database. The second column of the table represents the generated candidate instances of size-k, and the last column is the result of rearranging contents of the second column by the alphabetical order.

A.6*	A.6*, B.8*, C.1	A.6*, B.8*, C.1
	A.6*, B.8*, C.5*	A.6*, B.8*, C.5*
	~~A.6*, C.5*, B.1~~	non-clique
	A.6*, B.1, C.1	A.6*, B.1, C.1
B.6*	B.6* C.4*, A.3	A.3, B.6*, C.4
B.7*	B.7*, C.2, A.2	A.2, B.7*, C.2
C.4*	C.4*, A.3, B.3	A.3, B.3, C.4*
C.6*	C.6*, A.1, B.1	A.1, B.1, C.6*
	C.6*, A.5, B.1	A.5, B.1, C.6*

Fig. 14: The set of candidate of size-3 instances.

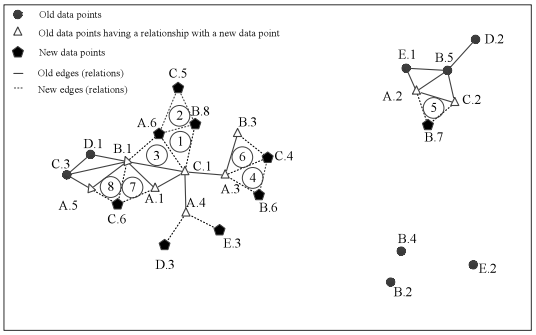


Fig. 15: The final results of size-3 candidate instances.

Step 4: ***Update the IncPI and IncPR***

In the Step 4, we will check whether the candidate instances exist at first. In Fig. 15, we can find that there are really eight candidate instances in the spatial database. Then, we will update the participation index and the partipation ratio in the incremental database. The formula of participation index and participation ratio as follows: The Incremental Participation Index $IncPI(X)$ of a co-location $X = \{E_1, ..., E_k\}$ is updated with $IncPI(X) = min_{E_i \in X}\{IncPR(X, E_i)\}$, $1 \leq i \leq k$. The Incremental Participation Ratio $IncPR(X, E_i)$ of event type Ei with the incremental co-location instances of X, $IncPR(X, E_i) = \frac{|O_i \bigcup NI_i|}{|SOLD_i| + |SINC_i|}$, where $| SOLD_i |$ is the total number of original objects of E_i. $| SINC_i |$ is the total number of new objects of E_i. O_i is a set of distinct objects of E_i in the original co-location instances of X. NI_i is a set of distinct objects of E_i in the incremental co-location instances of X.

In Fig. 14, we can get eight candidate instances, including {A.1, B.1, C.6*}, {A.2, B.7*, C.2}, {A.3, B.3, C.4*}, {A.3, B.6*, C.4*}, {A.5, B.1, C.6*}, {A.6*, B.1, C.1}, {A.6*, B.8*, C.1} and {A.6*, B.8*, C.5*} as shown in Fig. 16. They belong to the co-location pattern {A, B, C}. Then, we can calculate the incremental participation ratio and incremental participation index of co-location pattern {A, B, C}.

In Fig. 16, there are five distinct instances A.1, A.2, A.3, A.5 and A.6* (including 4 original ones, 1 incremental one) in the co-location {A, B, C}, so we can calculate $IncPR(\{A, B, C\},$

A	B	C
A.1	B.1	C.6*
A.2	B.7*	C.2
A.3	B.3	C.4*
A.3	B.6*	C.4*
A.5	B.1	C.6*
A.6*	B.1	C.1
A.6*	B.8*	C.1
A.6*	B.8*	C.5*
5/6	5/8	5/6

Incremental Participation Ratio

(1) IncPR({A, B, C}, A)
= (4+1) / (5+1)
= 5/6
(2) IncPR({A, B, C}, B)
= (2+3) / (5+3)
= 5/8
(3) IncPR({A, B, C}, C)
= (2+3) / (3+3)
= 5/6

Incremental Participation Index
= min(5/6, 5/8, 5/6) = 5/8

Fig. 16: The incremental participation index and incremental participation ratio of co-location pattern {A, B, C}.

A)= $\frac{5}{6}$. In the co-location $\{A, B, C\}$, there are five distinct instances B.1, B.3, B.6*, B.7* and B.8* (including 2 original ones, including 3 incremental ones), so we can calculate $IncPR(\{A, B, C\}, B) = \frac{5}{8}$. Similarly, there are five distinct instances C.1, C.2, C.4*, C.5* and C.6* (including 2 original ones, including 3 incremental ones), so we also can calculate $IncPR(\{A, B, C\}, C) = \frac{5}{6}$. Therefore, $IncPI(\{A, B, C\})$ is $\frac{5}{8}$, which is the minimum value between $IncPR(\{A, B, C\}, A)$, $IncPR(\{A, B, C\}, B)$ and $IncPR(\{A, B, C\}, C)$.

Step 5: ***The minimum prevalence threshold***

Finally, in this step, after updating the incremental participation, we will check whether the co-location pattern is prevalence. The criterion is that the co-location pattern is the prevalence if the participation index of candidate is greater than the min_prev threshold, Otherwise, the co-location pattern is called non-prevalence.

IV. Performance

In this section, we present the experiments to evaluate the performance between our non-overlapping approach and the EUCOLOC algorithm. All of the experiments were performed on a PC with an Intel Core i5 2.7 GHz $\times$ 2 CPU and 8 GB main memory, running on the Windows 10 platform. Both of the algorithms were implemented in Oracle Java SDK 7 and coded in Java language.

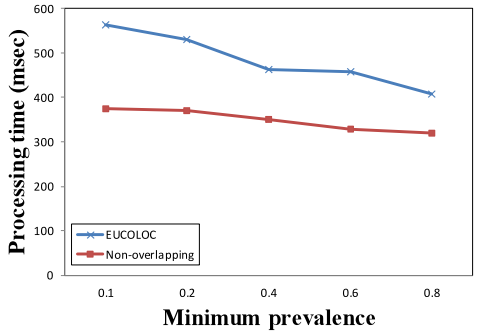
First, we present the evaluation of our non-overlapping approach and the EUCOLOC algorithm by varying the minimum prevalence threshold min_prev of the synthetic database. Second, we compare the performance by varying the distance between the points of the synthetic database. The following parameters used in the processing of generations synthetic data are shown in Table I. There are three datasets in our simulation as shown in table II.

TABLE I: Parameters Setting Used in The Experiment

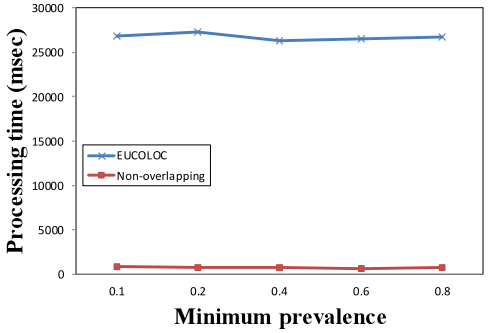
Parameters	Description
dis_thr	The threshold of the neighbor distance
min_prev	The threshold of minimum prevalence
$\|En\|$	The number of event types
$\|In\|$	The number of original instances
$\|NIn\|$	The number of incremental instances
$\|Rn\|$	The number of relations in the incremental database

TABLE II: The Datasets in Our Simulation

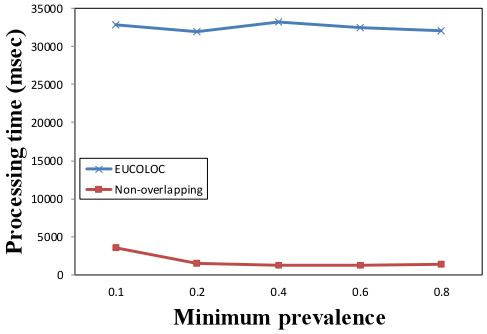
	$\|En\|$	$\|In\|$	$\|NIn\|$
Dataset 1	25	$8k$	$2k$
Dataset 2	40	$6k$	$4k$
Dataset 3	50	$6k$	$4k$



(a) dis_thr = 10, $|En|$ = 25, $|In|$ = $8k$, $|NIn|$= $2k$, $|Rn|$ = $3.6k$



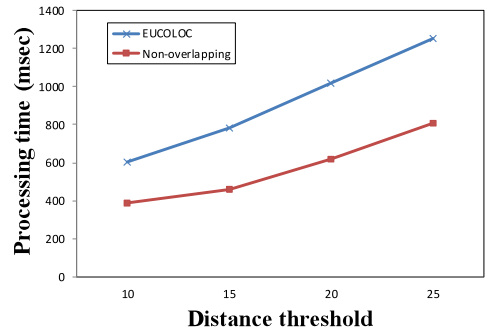
(b) dis_thr = 10, $|En|$ = 40, $|In|$ = $6k$, $|NIn|$= $4k$, $|Rn|$ = $7k$



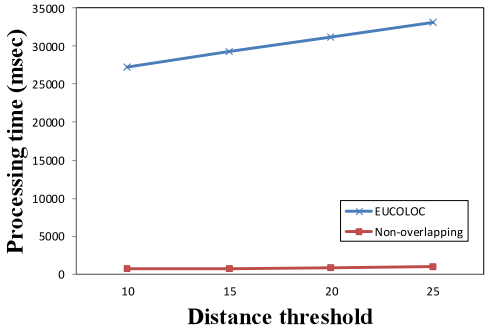
(c) dis_thr = 30, $|En|$ = 50, $|In|$ = $6k$, $|NIn|$= $4k$, $|Rn|$ = $11.2k$

Fig. 17: A comparison of the processing time under different minimum prevalence threshold.

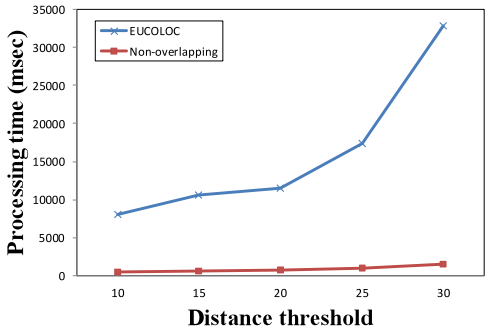
Note that the performance measure of the processing time includes the time for the generation of candidate instances and the mining step. From Fig. 17 and Fig. 18, we show that the processing time of our non-overlapping approach is faster than that of the EUCOLOC algorithm. Because our non-overlapping approach could generate fewer number of the candidate instances than the EUCOLOC algorithm, they need long time to deal with the generating of candidate instances



(a) Min_prev= 0.2, $|En|$ = 25, $|In|$ = 8k, $|NIn|$= 2k, $|Rn|$ = 3.6k



(b) Min_prev= 0.2, $|En|$ = 40, $|In|$ = 6k, $|NIn|$= 4k, $|Rn|$ = 7k



(c) Min_prev= 0.2, $|En|$ = 50, $|In|$ = 6k, $|NIn|$= 4k, $|Rn|$ = 11.2k

Fig. 18: The processing time under different distance threshold.

and then prune those are useless candidate instances soon. Therefore, the performance of our non-overlapping approach is better than the EUCOLOC algorithm.

V. Conclusion

In this paper, we have proposed an approach which generates size-k candidate instances efficiently. In our approach, we use less storage than the EUCOLOC algorithm to store the data points with neighbor relations because our incremental neighbors database does not overlap with the original neighbors database. We rearrange the relation by the asterisk annotation priority before we construct the incremental neighbors database. In this way, we can avoid generating non-incremental candidate instances. Moreover, we also check the subset relation of neighbor instances before we generate size-k candidate instances. In this way, we can avoid generating non-clique candidate instances completely. Therefore, we can generate fewer number of candidate instances than the EUCOLOC algorithm. The experimental results have shown that our approach is better than the EUCOLOC algorithm.

Acknowledgment

This research was supported in part by the Ministry of Science and Technology of Republic of China under Grant No. MOST-107-2221-E-110-064.

References

[1] Y. Huang, S. Shekhar, and H. Xiong, "Discovering colocation patterns from spatial data sets: a general approach," IEEE Trans. on Knowledge and Data Engineering, vol. 16, pp. 1472–1485, Dec, 2004.

[2] J. S. Yoo, S. Shekhar, J. Smith, and J. P. Julius, "A partial join approach for mining co-location patterns," Proc. of the 12th Annual ACM Int. Workshop on Geographic Information Systems, pp. 241–249, 2004.

[3] J. S. Yoo, and M. Bow, "A joinless approach for mining spatial colocation patterns," IEEE Trans. on Knowledge and Data Engineering, no. 10, vol. 18, pp. 1323–1337, Oct., 2006.

[4] L. Wang, Y. Bao, J. Lu, and J. Yip, "A New Join-less Approach for Colocation Pattern Mining," Proc. of the 8th IEEE Int. Conf. on Computer and Information Technology, pp. 197–202, 2008.

[5] L. Wang, L. Zhou, J. Lu, and J. Yip, "An Order-Clique-Based Approach for Mining Maximal Co-locations," Information Sciences, no. 19, vol. 179, pp. 3370–3382, Sept., 2009.

[6] X. Yao, L. Peng, L. Yang, and T. Liang, "A fast space-saving algorithm for maximal co-location pattern mining," Expert Systems with Applications, vol. 63, pp. 310–323, Nov., 2016.

[7] J. S. Yoo and M. Bow, "Mining Maximal Co-Located Event Sets," Proc. of the 15th Pacific-Asia Conf. on Advances in Knowledge Discovery and Data Mining, pp. 351–362, 2011.

[8] J. S. Yoo and M. Bow, "Finding N-most Prevalent Colocated Event Sets," Proc. of the 11th Int. Conf. on Data Warehousing and Knowledge Discovery, pp. 415–427, 2009.

[9] J. S. Yoo, and S. Shekhar, "Mining spatial colocation patterns: a different framework," Data Mining and Knowledge Discovery, vol. 24, pp. 159–194, Jan., 2012.

[10] J. S. Yoo and M. Bow, "Mining Top-k Closed Co-location Patterns," Proc. of IEEE Int. Conf. on Spatial Data Mining and Geographical Knowledge Services, pp. 100–105, 2009.

[11] X. Bao and M. Bow, "Mining Top-k-Size Maximal Co-location Patterns," Proc. of the Int. Conf. on Computer, Information and Telecommunication Systems, pp. 1–6, 2016.

[12] E. David, L. Maarten and S. Darren, "Listing All Maximal Cliques in Sparse Graphs in Near-Optimal Time," Proc. of the 21th Int. Symposium on Algorithms and Computation, pp. 403–414, 2010.

[13] S. Kwan Kim, Y. Kim and U. Kim, "Maximal Cliques Generating Algorithm for Spatial Co-location Pattern Mining," Proc. of the 8th FIRA Int. Conf. on Secure and Trust Computing, Data Management and Applications, pp. 241–250, 2011.

[14] S. Yang, L. Wang, X. Wang, and J. Lu "A Framework for Mining Spatial High Utility Co-Location Patterns," Proc. of the 12th Int. Conf. on Fuzzy Systems and Knowledge Discovery, pp. 595–601, 2015.

[15] J. He, Q. He, F. Qian, and Q. Chen "Incremental Maintenance of Discovered Spatial Colocation Patterns," Proc. IEEE Int. Conf. on Data Mining Workshops, ICDM Workshops 2008, pp. 399–407, 2008.

[16] J. Lu, L. Wang, Y. Fang, and X. Bao "A Novel Method on Incremental Mining of Spatial Co-locations," Proc. of the 3th IEEE Int. Conf. on Big Data and Smart Computing, pp. 69–72, 2016.

[17] J. S. Yoo, L. Wang, and J. Zhao, "Effectively updating co-location patterns in evolving spatial databases," Proc. of the 6th Int. Conf. on Pervasive Patterns and Applications, pp. 96–99, 2014.

[18] J. S. Yoo, B. Douglas, and K. David, "Proc. of IEEE Int. Congress on Big Data, BigData Congress," Proc. of the 6th Int. Conf. on Pervasive Patterns and Applications, pp. 25–31, 2014.

[19] R. Agrawal, and R. Srikant, "Fast algorithms for mining association rules in large databases," Proc. of the 20th Int. Conf. on Very Large Data Bases, pp. 487–499, 1994.