Mining Maximal Co-Location Patterns Based on the Count-Ordered Instances-Tree in Spatial Databases

Ye-In Chang, Wen-Hsiu Chung and Kuan-Chieh Lin

***Abstract*—Looking for the spatial co-location that appears frequently in nearby space is widely used in many areas, including mobile phone services and traffic management. To achieve this goal, the *SGCT* algorithm improves other algorithms which use tables to discover candidate sets. It uses an undirected graph to mine candidates of the maximal co-location patterns first, then uses a condensed-tree structure to store instance cliques of candidates. However, as the amount of data grows, the *SGCT* algorithm may store large number of nodes in the process of generating the tree. In this paper, we propose a new strategy which will consider the number of instances of each event. We propose a Count-Ordered Instances-tree to record candidates of relation sets. From our experimental results, we show that our approach needs shorter time and costs less storage space than the *SGCT* algorithm.**

***Keywords*—Maximal Co-location Patterns, Spatial Co-location Patterns, Spatial Co-location Rules, Spatial Database, Spatial Data Mining**

I. Introduction

Given a set of boolean spatial features, the co-location pattern discovery process finds the subsets of features frequently located in close geographic proximity. Boolean spatial features describe the presence or absence of geographic object types at different locations in the two dimensional or three dimensional metric space, such as the surface of the earth. Examples of boolean spatial features include plant or animal species, mobile service request, road types, diseases, climate crime, and business types. Fig. 1 shows a data set consisting of instances of several boolean spatial features, each represented by a distinct shape. A careful review reveals two co-location patterns: ('o', 'x') and ('+', '◇').

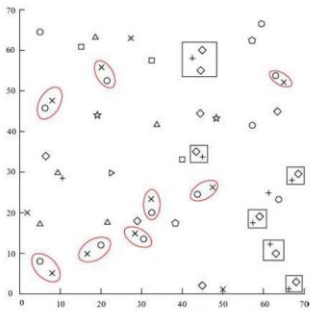

Fig. 1. Co-location patterns illustration: (o, x) and (+, ◇)

Co-location rule discovery is a process to identify co-location patterns from a spatial dataset. Co-location rule mining presents challenges due to the following reasons. First, it is important to adopt association rule mining algorithms [1, 2] to mine co-location instances, since spatial objects are embedded in the continuous space and share a variety of spatial relationships. A large fraction of the computation time is devoted to identifying the instances of co-location patterns. Second, it is non-trivial to reuse association rule mining algorithms which may require transactionizing spatial datasets for co-location pattern mining. It is a challenge due to the risk of transaction boundaries splitting co-location pattern instances across distinct transactions. Unlike market-basket data, spatial datasets often have no predefined transactions.

In spatial data mining, discovering maximal co-location patterns is an important issue. Huang *et al.* [3] purposed a general mining approach called the full-join approach. This *Apriori*-like method do well for sparse spatial datasets, but it is inefficient for dense spatial datasets. Because with the increasing number of co-location patterns, the computation time would be expensive. Huang and Shekhar proposed two approaches called the partial join approach [4] and the join-less approach [5] to improve the computation time. For the two approaches which are also join-based approaches, they use table instances as their data structures. Different from those join-based approaches, Wang *et al.* [6] proposed an order-clique approach, which uses four trees (*P*2-tree, *CPm*-tree, *Neib*-tree, *Ins*-tree) to mine the maximal co-location patterns and get the better performance than those join based approaches. To address the problems in other maximal co-location methods [6, 7], Yao *et al.* [8] proposed *SGCT* algorithm. They convert the prevalent *size*-2 co-locations into a sparse undirected graph to find maximal co-location candidates. Moreover, they devise a condensed-tree structure to store the instance clique of the candidate. The performance of the *SGCT* algorithm is better than the two algorithms [6, 7]. However, their method has a problem which stores instances in an alphabetic order in building the condensed-tree, and it is not efficient in some situations.

Therefore, in this paper, we propose the Count-Ordered Instances-Tree algorithm to mine maximal spatial co-location patterns. In our proposed method, we have three advantages. First, we propose a new approach to prune the candidates whose participation indices are smaller than the threshold defined by the user before we construct the tree for mining. Second, we use the formula in *9D-SPA* [9] to represent the relation of event pairs by an unique key value and then store its instance of such event pairs in a hash table. Thus, the advantage is that we can use the key to find the information which we want in the hash table that records each instance relation in constant time. Moreover, we propose the Count-Ordered Instances-Tree, which stores the instance relationships of maximal co-location candidates. The advantage of the Count-Ordered Instances-Tree is that the

number of nodes of such a tree is much smaller than that of the *SGCT* algorithm when generating the tree of the maximal co-location candidate. From our experimental results, we show that our approach to mine co-location patterns requires shorter time and costs less storage space than the *SGCT* method both in dense and sparse spatial datasets.

The rest of paper is organized as follows. In Section 2, we give a survey of the *SGCT* algorithm. In Section 3, we present our proposed approach. Section 4 presents the performance study of our approach and make a comparison between our approach and the *SGCT* algorithm. Finally, we give a conclusion in Section 5.

II. A Survey of the SGCT Algorithm

In this section, we give a brief description of the *SGCT* algorithm [8]. In the spatial dataset, each point contains a feature (event) type, an instance, and the coordinate of its location. The *SGCT* algorithm proposed by Yao *et al*. [8] uses a two-dimensional table, the size-2 instance table *InsTable*2. First, according to the distance threshold defined by the user, they store the information of instance pairs of different types which have neighbor relationships in space. In table *InsTable*2, the instance pairs will be stored in the corresponding types. The candidate of size-2 co-location is used to calculate the prevalence index defined later and prune those candidates whose prevalence indices are less than the minimum prevalence threshold defined by the user. Then, they use a modified sparse graph from prevalent size-2 co-locations to find all candidates of the maximal co-locations. The *SGCT* algorithm uses a condensed instance tree to store the information and gets its instance cliques of each maximal co-location candidate. Next, they calculate its prevalence index, and verify whether it is not smaller than the prevalence threshold, and reserve the candidate as a real co-location pattern. Otherwise, they replace it with its subsets. Next, they construct the condense instance tree (*CInsT ree*) based on the size-2 instance table to confirm the instance cliques of the maximal co-location candidates that actually exist in the spatial database. Then, they build a two-level tree containing instance pairs of two types according to *InsTable*2(A, B), *i.e.* the alphabetical order. Later, for nodes of type B in level 2, they search for their neighbor instances of type C from *InsTable*2(B, C) and store them in a list. Then, the hierarchical verification is performed between instances of event types A and C. The similar step is processed for event types D and E.

III. The Proposed Algorithm

In this section, we present our proposed Instances-Tree based on Count-Ordered to find the instances of candidate patterns and describe how to prune the large number of co-location candidates using *CountEP*.

In the preprocessing step for the input of the spatial Database, we use an example of the spatial dataset to illustrate our method. Fig. 2 contains a number of points in a database with two-dimensional coordinates. These points are composed of five different event/feature types (ES), A,B,C,D, and E. Each event type may have different number of instances. Moreover, for such a set of instances in Fig. 2, we record them in a table *IS* which lists all instances of each event type and their count (*i.e.*, the number of instances for each type). For example, A has four instances $A.1, A.2, A.3$ and $A.4$. The total instances of each event type B, C, D, and E are 4, 6, 5, 3, respectively.

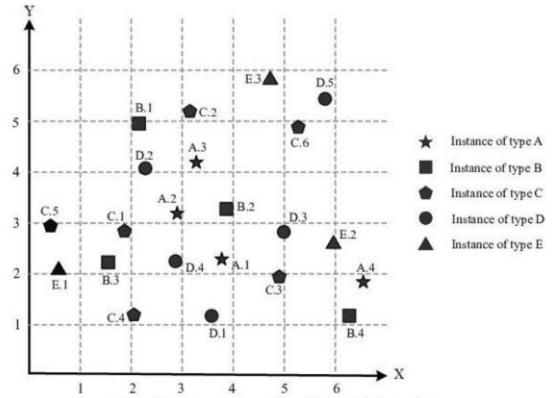

Fig. 2. An example of spatial points

In the input data, there are two threshold values defined by the user. One is the distance threshold (*dis_thr*) and the other one is minimum prevalence threshold (*Min_prev*). The range of *Min_prev* is defined between 0 and 1. In our example, the *dist_thr* and *Min_prev* are set as 15cm and 0.3, respectively.

First of all, we calculate the distance between two distinct points in Fig. 2 through the formula of Euclidean Distance. We choose pairs of points whose distances are not larger than the *dist_thr*. Then, for those neighbor relations, we connect the related points with a solid line in Fig. 3. Note that we are only interested in the relation between different types. We represent the points in relation as relations of *size*-2 instances pair by pair, for example, ($A.1$, $B.2$) and record them (*i.e.* 30 edges) in Table RI_2.

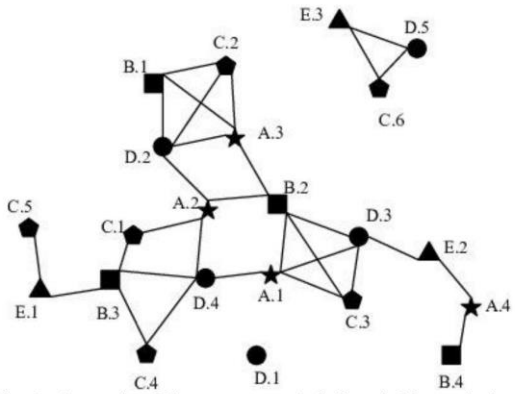
Fig. 3. A graph of five events and their neighbor relations

After finding all relations in Fig. 2, we want to find out the candidate sets whose participation indices are larger than or equal to *Min_prev*. The definitions are shown as follows.

The participation index $Pi(C)$ of a co-location $C = \{E1, ..., Es\}$ is defined as $Pi(C) = min_{Ei \in C} \{Pr(C, E_i)\}$, $1 \le i \le k$ [8]. Participation ratio $Pr(C, Ei)$ is defined as $Pr(C, Ei) = \frac{\text{Number of distinct objects of } E_i \text{ in instances of } C}{\text{Number of objects of } E_i}$.

Before calculating the participation index of a collocation $\{D, E\}$, we must calculate the participation ratio of each event in the co-location at first. In Fig. 3, ($D.3$, $E.2$) and ($D.5$, $E.3$) are neighbor relations. There are two distinct instances $D.3$

and $D.5$ in the co-location, so we can calculate $Pr(\{D,E\}, D)$= 2/5. Similarly, we can calculate $Pr(\{D,E\}, E)$= 2/3. Therefore, $Pi(D, E)$ is 2/5 , which is the minimum value between $Pr(\{D,E\}, D)$ and $Pr(\{D,E\}, E)$.

Now, we will describe our proposed steps and followed by an example to illustrate the idea. Table I shows the variables used in our method.

Table I: Variables

Variable	Definition
ES	The set of spatial event/feature types
IS	The set of spatial instances
IS_k	The set of instances of event k
dis_thr	A distance threshold
Min_prev	A minimum prevalence threshold
Ins_HT2	A hash table of size-2 instance pair
G	Size-2 co-location graph from hash table Ins_HT2
e	Edge set of graph G
v	vertex set of graph G event types.
$N(v)$	The neighboring vertex set of v.
$MCanP$	A set of all maximal co-location clique candidates.
$MCan$	The candidate of a maximal co-location clique
$CountEP$	The count of event pairs with size=2
$RatioEP$	The participation index of each event pair

Step 1: (*Determine an Event Order*)

First, we use the counts of instances from IS (the set of spatial instances) to determine an *Event Order*. In our example, there are five event types. Among the five event types, event C has the largest count. Event A has the same count of instances as Event B, so we sort them by the alphabet order, $A > B$. Finally, we get the *Event Order* [C, D, A, B, E].

Step 2: (*Sort* RI_2)

Based on the *Event Order* decided from Step 1, in each related instance pair of RI_2 (the relations of size-2 instances), we exchange the position of the two instances if necessary. We mark a '*' sign on the modified pairs. We use the *Event Order* [C, D, A, B, E] to rearrange RI_2. First, we rearrange the position of the two instances in each neighboring pair. For example, for relation ($A.1$, $C.3$), we exchange the two instances and get ($C.3$, $A.1$). Then, we mark the modified pairs with a '*' sign, and the result is shown in Table II.

Table II: Step 2: Rearranging $RI2$ by *Event Order*

Relations of size-2 instances		
(A.1, B.2)	*(C.2, A.3)	*(D.4, B.3)
*(C.3, A.1)	*(D.2, A.3)	(B.3, E.1)
*(D.3, A.1)	(A.4, B.4)	(C.2, D.2)
*(D.4, A.1)	(A.4, E.2)	(C.3, D.3)
(A.2, B.2)	*(C.2, B.1)	(C.4, D.4)
*(C.1, A.2)	*(D.2, B.1)	(C.5, E.1)
*(D.2, A.2)	*(C.3, B.2)	(C.6, D.5)
*(D.4, A.2)	*(D.3, B.2)	(C.6, E.3)
(A.3, B.1)	*(C.1, B.3)	(D.3, E.2)
(A.3, B.2)	*(C.4, B.3)	(D.5, E.3)

Step 3: (*Construct a size-2 instance table*)

Based on the RI_2, we construct a hash table to store the relations between different event types.

(a) We use the formula used in the *9D-SPA representation* [9] as the hash function to get the unique value of each combination of different event types.

(b) We add all the neighboring pairs into the hash table according to the corresponding unique value which represent the event-pair (E_1, E_2).

Given two events E_i and E_j , where $j > i$, then the unique value of E_{ij} can be easily computed by using the following formula [9]: $E_{ij} = \frac{(j-1)(j-2)}{2} + i$.

In Table II, we use RI_2 to show the neighboring pairs. Then, we will convert RI_2 into a hash table in Step 3. We use the above rule to calculate the unique value of E_{ij} . Due to the input of the function $F(E_i, E_j)$, where E_j must be larger than E_i, we need to assign the smaller event to E_i, and the other one to E_j. Then, we record the event-pair and its instances into the corresponding unique value of the hash table. In our example, we know $ES = \{A, B, C, D, E\}$, and we get A=1, B=2, C=3, D=4 and E=5 by the rule. Then we calculate the unique value of ECA where C is larger than A, and we put the larger event C into Ej and the other one into Ei. The formula is derived as E_{ij}= $(j\text{-}1)(j\text{-}2)/2 + i$ = (3-1)(3-2)/2 +1 = 2. Therefore, we calculate all the corresponding unique value of two spatial events. From RI_2, we add the event-pair (C, A) and its instances pairs into the hash table as shown in Fig. 4.

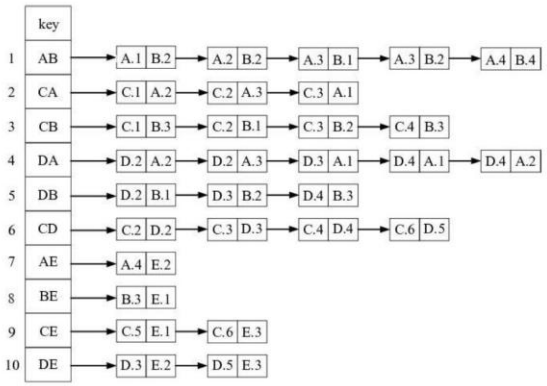

Fig. 4. Step 3: A hash table with size-2 instances

Step 4: (Generate *CountEP*)

In order to build *CountEP* (the count of event pairs with *size* = 2) from *size*-2 instances table in Step 3, we use the strategy below.

(a) Calculate the number of different instances of each event pairs in *size*-2 instance table, and record them in *CountEP*.

(b) The number of the second instance minus the number of the first instance in *CountEP* and save the result as *cmp* (compare). There are three cases for the value of *cmp* : (1) *cmp* = 0, (2) *cmp* > 0, (3) *cmp* < 0 (We marked the case with '*').

(c) If *cmp* is greater than or equal to 0, we update the number of first instance in the record of *size*-2 instances table with the value of the corresponding event in *IS*.

The purpose of computing *cmp* is to avoid the computation of the following step for getting the result of Pi, if *cmp* ≥ 0. In the cases of *cmp* ≥ 0, we let the number of instances of the first event in the pair be the number of instances of such an event in the whole database.

The objective of constructing *CountEP* is to show the relations of the events from *size*-2 instances table. In Step 4, we use Table III to show the process (4-(a), 4-(b)), and our goal is to get rid of the *size*-2 candidates whose participation

indexes are smaller than *Min_prev*. In Step 4-(c), if *cmp* ≥ 0 in *CountEP*, we update the count of first event with the value of event *D* in *IS*. Therefore, event pair (*C*, *D*) is changed to (*C*.6, *D*.4) in *CountEP*. The minimum of result (*C*.6, *D*.4) is 4, and it means that *C* and *D* appear together four times. Table IV shows those relations in *CountEP*.

Table III: Step 4-(a) and Step 4-(b): The count of event pairs (*CountEP*)

CountEP		
(C:4, D:4, *cmp*:0)	(D:3, A:3, *cmp*:0)	*(A:4, B:3, *cmp*:-1)
(C:3, A:3, *cmp*:0)	(D:3, B:3, *cmp*:0)	(A:1, E:1, *cmp*:0)
*(C:4, B:3, *cmp*:-1)	(D:2, E:2, *cmp*:0)	(B:1, E:1, *cmp*:0)
(C:2, E:2, *cmp*:0)		

Table IV: Step 4-(c): Updating the count of event pairs (*CountEP*)

CountEP		
(C:6, D:4, *cmp*:0)	(D:5, A:3, *cmp*:0)	*(A:4, B:3, *cmp*:-1)
(C:6, A:3, *cmp*:0)	(D:5, B:3, *cmp*:0)	(A:4, E:1, *cmp*:0)
*(C:4, B:3, *cmp*:-1)	(D:5, E:2, *cmp*:0)	(B:4, E:1, *cmp*:0)
(C:6, E:2, *cmp*:0)		

Step 5: (Generate *RatioEP*)

(a) If the pair is marked with '*' in *CountEP* which means that its *cmp* is less than 0, we calculate $min(\frac{\text{the count of the first event}}{\text{the count of the first event in } IS}, \frac{\text{the count of the second event}}{\text{the count of the second event in } IS})$; otherwise, we calculate $\frac{\text{the count of the second event - cmp}}{\text{the count of the first event (in } IS)}$. Then, we store the result into *RatioEP* (the ratio of event pairs). Note that in Step 4-(c), we record the count of the first event with the count of corresponding event in *IS*. Moreover, (*the count of the second event*) – *cmp* = (*the count of the second event*) – (*the count of the second event* - *the count of the first event*) = *the count of the first event*. That is, for the case of *cmp* ≥ 0, we care *the count of the first event* which will be the numerator of *Pi*. Furthermore, the denominator of Pi will be the count of the first event in *IS*, which is recorded in the updated *EC* in Step 4-(c). The key point is that we list the instance pairs in the descending order of counts of events. In the previous step, we have sorted the event pair according to *Event Order*, so the count of the first event is greater than or equal to the second event (*x* , *y*).

(b) We compare all the values in *RatioEP* with *Min_prev*. Then, we remove the pair if the value is smaller than *Min_prev*, and also delete the entire data of the event pair in the *size*-2 instance table.

In Step 5, our goal is to calculate the participation index(*Pi*) and delete those candidates whose participation index are smaller than *Min_prev*. We can derive participation index from the information stored in *CountEP*. In most cases, due to the same numerator, we can compare the fraction values by only comparing their denominators. Which means, if the denominator is larger, then the value would be smaller. The minimum of the participation ratio is produced by the largest event count placed in the denominator.

Additionally, we propose a method to calculate the result. In order to speed up the process of pruning, we sort the event types by the count (the number of instances for each event type). We induce two formula to calculate the participation indices by using the information of *CountEP*. For the case of *cmp* < 0, we still must compare the participation ratio of the event pair and decide which one is the minimum value and such a result is still larger than the threshold. For the cases of *cmp* ≥ 0, we only have to care of the participation ratio of the first event in the event pair and the value is same as $\frac{\text{the count of the second event - cmp}}{\text{the count of the first event in the whole database (i.e., } IS)}$.

Note that the numerator, the count of the second event - *cmp*, is equal to the count of first event participating in the event pair. The reason of such a reduced computation step could be explained as follows.

In our example, we set *Min_prev* = 0.3. Candidates (*A, E*) and (*B, E*) are pruned, because the index of (*A, E*) and (*B, E*) are both (1-0)/4 = 1/4, which is less than our defined threshold 0.3.

The result of Step 5-(b) before the pruning process is shown in Table V. After the pruning process, candidates (*A, E*) and (*B, E*) are pruned from the hash table.

Table V: Step 5-(a): The ratio of event pairs (*RatioEP*)

RatioEP		
(CD: $\frac{4}{6}$)	(DA: $\frac{3}{5}$)	(AB: $min(\frac{4}{4},\frac{3}{4})$)*
(CA: $\frac{3}{6}$)	(DB: $\frac{3}{5}$)	(AE: $\frac{1}{4}$)
(CB: $min(\frac{4}{6},\frac{3}{4})$)*	(DE: $\frac{2}{5}$)	(BE: $\frac{1}{4}$)
(CE: $\frac{2}{6}$)		

Step 6: (Construct the graph)

Using the event types of the candidate set in the hash table to construct the graph. If two vertices event *i* and event *j* are related, then there will be an edge e_n between them. We calculate *n* by the formula used in *9D-SPA representation* in Step 3. Our goal is to find the maximal cliques.

By using the prevalent *size*-2 co-location, we can get the graph as shown in Fig. 5 and learn that *(A,B,C,D)* is one of the maximal co-location candidates.

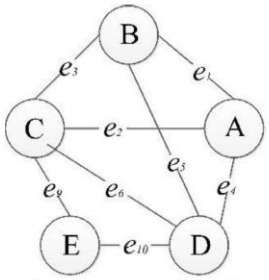

Fig. 5. Step 6: The graph of size-2 co-location

Step 7: (Build the *Count-Ordered Instrances-Tree*)

(a) According to the size of each neighbor relation in the hash table, we find out the least two event pairs.

(b) We use the least two event pairs to change the *Event Order* and rearrange a new order to build the *Count-Ordered Instances-Tree* with the root "*COIT*".

(c) Based on the new order, we can use the hash table to get the information of instances pairs and generate the nodes level by level.

(d) We make sure that each stored node has a relationship with its ancestors.

The rules of the *Count-Ordered Instrances-Tree* are as follows. (1) The depth of *COITree* is equal to the length of

MCan. (2) The types of instances at each level i are the same as $Mcan(i)$.

In Step 6, we have found out the candidates of the maximal co-location patterns, and now we will build the *Count-Ordered Instances-Tree* to represent the relations between instances. First, we know that $\{A,B,C,D\}$ is one of the candidates maximal co-location patterns. Then, we get the information of the two event pairs $\{C,A\}$ and $\{D,B\}$ which appear the least times from size-2 hash table *Ins_HT*2. We exchange the *Event Order* from *CDAB* to *CADB*. For each instance pair of the event types of $\{C,A\}$, first, we store "*COIT*" as the root. Then, determine whether the current instance exists in the first layer. If it exists, we add the related nodes as the child of the current instance. Otherwise, we add the instance pair as a new branch of the root. For example, we get the instance pairs ($C.1$, $A.2$), ($C.2$, $A.3$) and ($C.3$, $A.1$) of event pair $\{C,A\}$ from *Ins_HT*2 and add them as the children of the root "*COIT*". Then, we determine $A.2$ has relationships with $D.2$ and $D.4$. We confirm whether $D.2$ and $D.4$ have relationship with their ancestor $C.1$, respectively. If not, we will not add them into the tree. On the other hand, $A.3$ has a relationship with $D.2$, and $D.2$ has a relationship with its ancestor $C.2$. Therefore, we add $D.2$ as the child of current instance $A.3$. We show Step 7 in Fig. 6, Fig. 7, Fig. 8. (Note that in Fig. 7, we also check whether the instances between events C and D exist. We find that only instance pairs ($C.2$, $D.2$) and ($C.3$, $D.3$) exist. Moreover, in Fig. 8, we also check whether the instances between events A, C and B exist. We find that instance pairs ($B.1$, $A.3$), ($B.1$, C.2), ($B.2$, $A.1$) and ($B.2$, $C.3$) exist.)

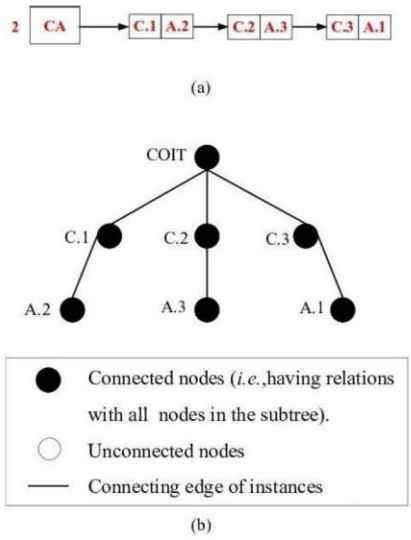

Fig. 6. Step 7: Building the two-level *Count-Ordered Instrances-Tree*: (a) the related hash table (C,A); (b) two-level tree containing instances of events C and A.

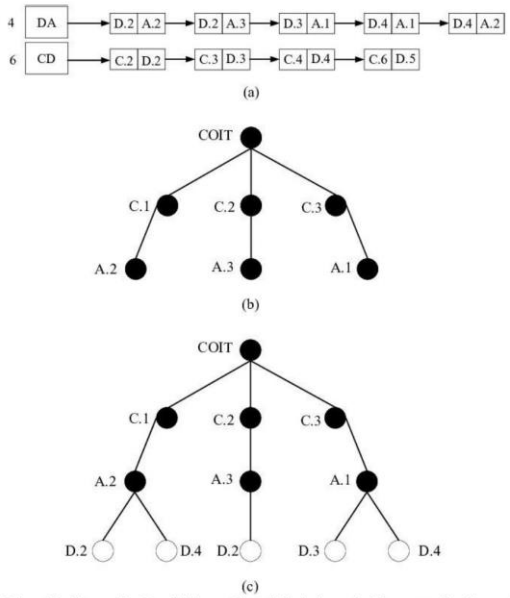

Fig. 7. Step 7: Building the third-level *Count-Ordered Instrances-Tree*: (a) the related hash table (D,A) and (C,D); (b) two-level tree containing instances of event (C,A); (c) searching the neighbor instances of type D connected with type A in level 2.

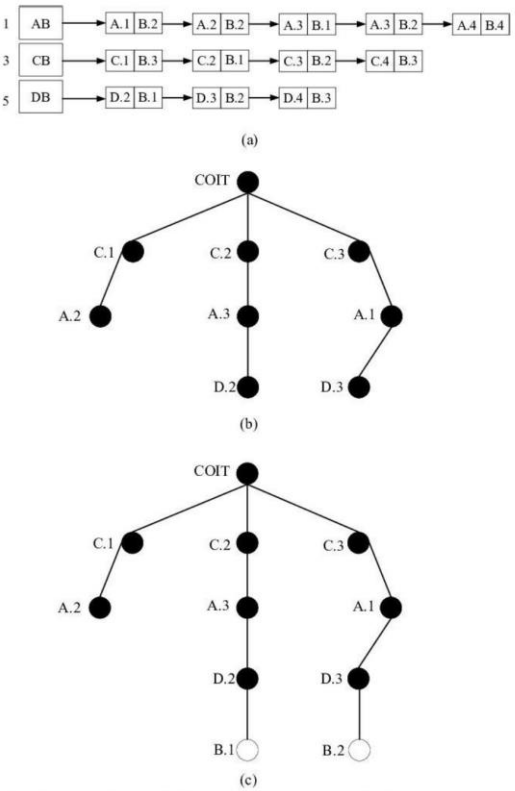

Fig. 8. Step 7: Building the fourth-level *Count-Ordered Instrances-Tree*: (a) the related hash table (A,B), (C,B) and (D,B); (b) third-level tree; (c) searching the neighbor instances of type B connected with type D in level 3.

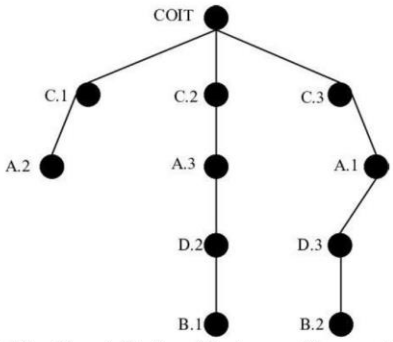

Fig. 9. The *Count-Ordered Instances-Tree* containing only 3 paths (10 nodes) to find the instances of maximal co-location candidate *A,B,C,D*

For the comparison part, we make a comparison to discuss the difference between our proposed Count- Ordered Instances-Tree and the *SGCT* algorithm [8]. In our example, the candidates of maximal co-location are [*CADB*] and [*CDE*]. Fig. 9. illustrates the process of generating the instances-tree of the maximal co-location patterns in our approach. It contains 3 paths and 10 nodes. Then, Fig. 10 shows the process of condensed-instances tree in the *SGCT* algorithm containing 5 paths and 13 nodes. Therefore, when we generate the instances-tree of maximal co-location patterns, our approach would be more efficient than the *SGCT* algorithm.

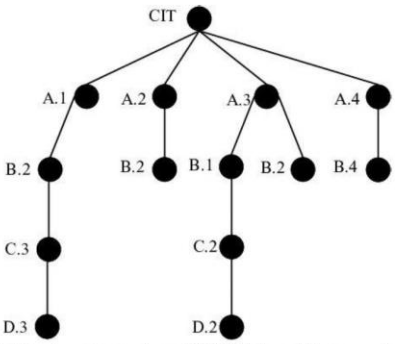

Fig. 10. The tree based on *SGCT* algorithm containing 5 paths (13 nodes) to find the instances of maximal co-location candidate *A,B,C,D*

IV. PERFORMANCE

In this section, we will compare the performance between our approach and the *SGCT* algorithm.

In this performance study, we generate the objects and their corresponding locations in a two-dimensional coordinate as the input data. An object contains an event and an instance. The parameters used in the process of generating synthetic data are described as follows. Parameter *dis_thr* means the distance threshold of the neighbor relation. We use the parameter *dis_thr* to obtain relations. If the distance of two objects is smaller than *dis_thr* which is given by the user, it represents the two objects as neighbor relation. Parameter *Min_prev* means the prevalence threshold which is defined by the user. The range of *Min_prev* is generated between 0 and 1. Parameter $|D|$ is a flag bit and represents the density of the spatial dataset. When the spatial data is sparse, Parameter $|D|$ is 0. Otherwise, Parameter $|D|$ is 1, when the spatial data is dense. We use two parameters which are defined by the user: parameter $|E_N|$ to represent the total number of events and the parameter $|I_N|$ to represent the total number of instances in the spatial dataset. Parameter $|R_N|$ means the number of relation pairs and is affected by parameter *dis_thr* and parameter $|D|$.

The synthetic datasets are generated using a spatial data generator similar to [3]. Moreover, the *SGCT* approach also uses such kind of input (*i.e.*, a 2-D coordinate). Furthermore, we will give an example of synthetic data as follows. First, we initialize the parameters as follows. We set *dis_thr* = 15, *Min_prev* = 0.2, $|E_N|$ = 5, and $|I_N|$ =24. In this dataset, it includes 5 events which are A, B, C, D and E and contains 24 instances which are randomly distributed to each event. We assume that event A, B, C, D and E have 4, 7, 6, 5, and 2 instances, respectively. Second, we assume that the candidate contains events A, B, C and E. In the co-location pattern, event B has the largest number of instances in those events. We use the instances of event B to calculate the count which is ⌈7 * 0.2⌉ = 2. The co-location patterns can be larger than the *Min_prev*, so it means that the relations of the co-location patterns have to appear two times.

Here, we will show the experimental results which compare the performance between our approach and the *SGCT* approach. We have two datasets, and we will compare the processing time and the number of nodes of the synthetic database. The first dataset is dense with 25 spatial events and 1k instances. Those instances are distributed in a map with size 100*100, resulting the density=0.1. The second dataset is sparse with 25 spatial events and 5k instances. Those instances are distributed in a map with size 500*500, resulting the density=0.02. We first present the performance experiment of comparing algorithms in dense datasets.

In Fig. 11 and Fig. 12, we set the upper bound of the neighbor distance=15 (*dis_thr*=15, the dense dataset). From Fig. 11, we show that as the number of relations increases in the spatial dataset, our approach still generates less number of nodes than the *SGCT* algorithm. From Fig. 12, we show that the processing time of our approach is shorter than the *SGCT* algorithm. The number of cliques under the change of *Min_prev* is shown in Table VI.

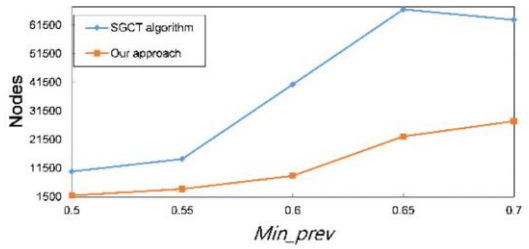

Fig. 11. A comparison of the number of nodes of the dense dataset under different *Min_prev*

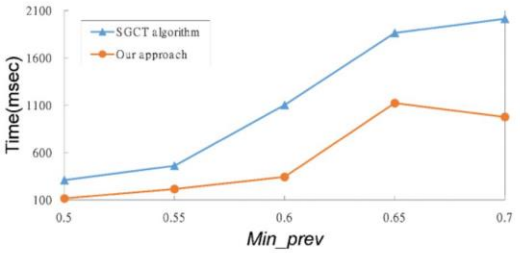

Fig. 12. A comparison of the processing time of the dense dataset under different *Min_prev*

Table VI: The number of cliques of the dense dataset under different *Min_prev*

Min_prev	0.5	0.55	0.6	0.65	0.7
The number of cliques	15	19	36	51	51

In Fig. 13 and Fig. 14, we set the upper bound of the neighbor distance=17 (*dis_thr*=17, the sparse case). From Fig.13, we show that our approach is still generating less number of nodes than the *SGCT* algorithm. As the *Min_prev* increases, the number of nodes constructed in both approaches changes. The number of nodes increases when the *Min_prev* is changed from 0.25 to 0.4, and the number of nodes decreases when the *Min_prev* is 0.45. However, our approach always generates less number of nodes than the *SGCT* algorithm. Note that when the *Min_prev* is too high, many candidates of size-2 could be pruned in both algorithms, resulting in decrease of constructed trees in both algorithms. The main reason for less number of needed nodes for mining in our approach than the *SGCT* algorithm is that we sort the tree by the number of relations. That is, we do the sorting step in the preprocess of constructing the Count-Ordered Instances-tree for mining.

In Fig. 14, we present the comparison of processing time of the *SGCT* algorithm and our approach with the sparse dataset under the change of minimum prevalence threshold. From Fig. 14, we show that the processing time of our approach is faster than that of the *SGCT* algorithm. The processing time of the *SGCT* algorithm and our approach first increases and then decreases because the processing time is related to the *Min_prev*. For example, when the *Min_prev* is changed from 0.25 to 0.35, the processing time of both algorithms increases. However, when the *Min_prev* is changed from 0.35 to 0.45, the processing time of both algorithms decreases. When the *Min_prev* is too high, many candidates of size-2 are pruned in both algorithms, resulting in the decrease of the size of the constructed trees in both algorithms to prune many candidates of *size*-2. So, the number of cliques decreases. The number of cliques under the change of *Min_prev* is shown in Table VII.

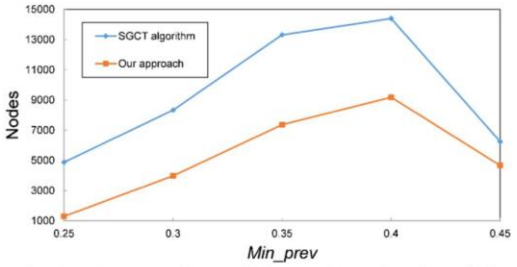

Fig. 13. A comparison of the number of nodes of the sparse dataset under different *Min_prev*

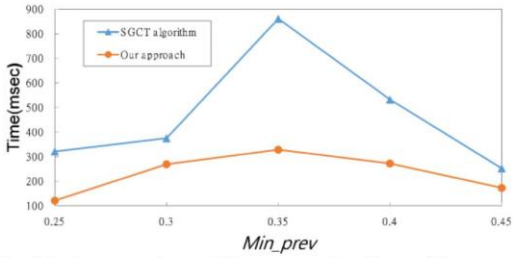

Fig. 14. A comparison of the processing time of the sparse dataset under different Min_prev

Table VII: The number of cliques of the sparse dataset under different *Min_prev*

Min_prev	0.25	0.3	0.35	0.4	0.45
The number of cliques	10	21	30	31	12

V. CONCLUSION

In this paper, we have proposed an approach which uses the data structure Count-Ordered Instances-Tree for generating the instances of the maximal co-location patterns efficiently. In our approach, our Count-Ordered Instances-Tree needs less number of nodes than the structure of the *SGCT* algorithm and can get the same instances cliques. Because the order of our generating tree is based on the number of relations in the database. The experimental results have shown that our approach is better than the *SGCT* algorithm. Data increment may change the found maximal co-location patterns; therefore, how to find the maximal co-location patterns incrementally is the possible future research direction.

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] 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.

[2] L. Wang, K. Xie, T. Chen, and X. Ma, "Efficient discovery of multilevel spatial association rules using partitions," *Information Software Technology*, Vol. 47, No. 13, pp. 829–840, Oct. 2005.

[3] 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, No. 12, pp. 1472–1485, Dec. 2004.

[4] J. S. Yoo, S. Shekhar, J. Smith, and J. P. Kumquat, "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.

[5] J. S. Yoo and S. Shekhar, "A joinless approach for mining spatial colocation patterns," *IEEE Trans. on Knowledge and Data Engineering*, Vol. 18, No. 10, pp. 1323–1337, Oct. 2006.
[6] L. Wang, L. Zhou, J. Lu, and J. Yip, "An order-clique-based approach for mining maximal co-locations," *Information Sciences*, Vol. 179, No. 19, pp. 3370–3382, Sept. 2009.
[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] X. Yao, L. Peng, L. Yang, and T. Chi, "A fast space-saving algorithm for maximal co-location pattern mining," *Expert Systems with Applications*, Vol. 63, pp. 310–323, Nov. 2016.
[9] P. W. Huang and C. H. Lee, "Image database design based on 9D-SPA representation for spatial relations," *IEEE Transactions on Knowledge and Data Engineering*, Vol. 16, No. 12, pp. 1486–1496, Dec. 2004.

Ye-In Chang received her B.S. degree in Computer Science and Information Engineering from National Taiwan University, Taipei, Taiwan, in 1986. She received her M.S. and Ph.D. degrees in Computer and Information Science from The Ohio State University, Columbus, Ohio, in 1987 and 1991, respectively. From August 1991 to July 1999, she joined the faculty of Department of Applied Mathematics at National Sun Yat-Sen University, Kaohsiung, Taiwan. From August 1997, she has been a Professor in Department of Applied Mathematics at National Sun Yat-Sen University, Kaohsiung, Taiwan. Since August 1999, she has been a Professor in Department of Computer Science and Engineering at National Sun Yat-Sen University, Kaohsiung, Taiwan. Her research interests include database systems, distributed systems, multimedia information systems, mobile information systems and data mining.

W. H. Chung received M.S. degrees in Computer Science and Engineering from National Sun Yat-Sen University in 2017. She is currently a system designer in Taiwan.

K. C. Lin received B.S. degree from Feng Chia University in 2017. He is currently a M.S. student in Department of Computer Science and Engineering at National Sun Yat-Sen University. His research interests includes data mining and distributed computing.