Discovering Prevalent Co-location Patterns Without Collecting Co-location Instances

Vanha Tran[1](✉), Caodai Pham[2], Thanhcong Do[1], and Hoangnam Pham[1]

[1] FPT University, Hanoi 155514, Vietnam
hatv14@fe.edu.vn, {congdthe150385,namphhe160714}@fpt.edu.vn
[2] Le Quy Don Technical University, Hanoi 11355, Vietnam
daipc.isi@lqdtu.edu.vn

Abstract. Discovering prevalent co-location patterns (PCPs) is a process of finding a set of spatial features in which their instances frequently occur in close geographic proximity to each other. Most of the existing algorithms collect co-location instances to evaluate the prevalence of spatial co-location patterns, that is if the participation index (a prevalence measure) of a pattern is not smaller than a minimum prevalence threshold, the pattern is a PCP. However, collecting co-location instances is the most expensive step in these algorithms. In addition, if users change the minimum prevalence threshold, they have to re-collect all co-location instances for obtaining new results. In this paper, we propose a new prevalent co-location pattern mining framework that does not need to collect co-location instances of patterns. First, under a distance threshold, all cliques of an input dataset are enumerated. Then, a co-location hashmap structure is designed to compact all these cliques. Finally, participation indexes of patterns are efficiently calculated by the co-location hashmap structure. To demonstrate the performance of the proposed framework, a set of comparisons with the previous algorithm which is based on collecting co-location instances on both synthetic and real datasets is made. The comparison results indicate that the proposed framework shows better performance.

Keywords: Prevalent co-location pattern · Co-location instance · Clique · Hashmap

1 Introduction

The unknown and valuable knowledge mined from spatial data sets can be applied to many domains, thus spatial data mining has received more and more attention recently. Prevalent co-location pattern (PCP) mining, which discovers a set of spatial features whose instances frequently appear together in proximity space, is an important branch of spatial data mining. For example, shopping centers and restaurants are co-located commonly in cities, thus {Shopping center, Restaurant} is called a prevalent co-location pattern. The information about the pattern can be provided to businessmen where they should set up new shops or restaurants to get the best benefit. The PCP mining technology has been applied widely in location-based services [1], environment [2], public safety [3], socio-economics [4], ecology [5], urban transportation [6], and so on.

N. T. Nguyen et al. (Eds.): ACIIDS 2023, LNAI 13995, pp. 408–420, 2023.
https://doi.org/10.1007/978-981-99-5834-4_33

Different from association rule mining [7], objects in spatial data carry spatial locations and are distributed in a continuous space. They have complex neighbor relationships. Hence, discovering PCPs from spatial data sets is nontrivial.

1.1 Related Work

Many mining algorithms have been proposed and they can be roughly divided into three categories. The first type can be called the conventional mining algorithm which effectively mines all correct prevalent patterns. These algorithms belonging to this type use the time-consuming join operator to generate co-location instances [8, 9], check neighborhoods [10], and construct a prefix tree structure to find co-location instances [11, 12]. However, these algorithms are hard to deal with big data sets, thus based-MapReduce [13], Hadoop [14], and GPU [15–17] parallel mining algorithms have been proposed. The second type can be named the special spatial data type mining algorithm which discoveries co-location patterns from special spatial data such as spatiotemporal data [18], rare event data [19], uncertain data [20], fuzzy data [21], data with network constraints [1] or data with considering density-weighted distance [22], and so on. The third type can be labeled as the compression PCP mining algorithm which represents concisely the mining results by top-*k* closed patterns [23], maximal patterns [24], and redundant patterns from the mining results [25, 26].

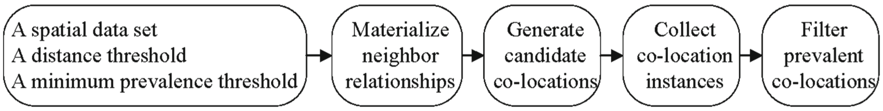

A spatial data set
A distance threshold
A minimum prevalence threshold
Materialize neighbor relationships
Generate candidate co-locations
Collect co-location instances
Filter prevalent co-locations

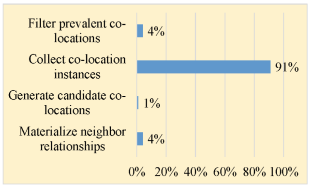
Fig. 1. The common framework of prevalent co-location pattern mining.

All of the above algorithms are developed based on a common framework shown in Fig. 1 [10] with four phases. The first phase materializes neighbor relationships of instances of the spatial data set. A set of candidate co-location patterns is generated in the second phase. The third phase collects all co-location instances of each candidate. After that, the participation index of each candidate pattern is calculated based on the co-location instances and PCPs are filtered in the fourth phase. However, the framework has two drawbacks. First, it employs a time-consuming generate-test candidate model. If the number of features is large or the data set is big/dense, the number of candidates becomes very huge and the mining process will take a lot of execution time [14].

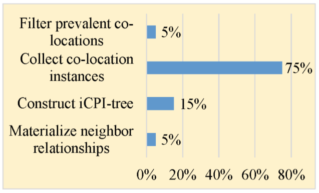
For example, Fig. 2 shows the execution time of each phase [27] in joinless [10] and iCPI-tree [11]. It can be seen that most of the execution time is devoted to collecting co-location instances. Second, if users change the minimum prevalence threshold, this framework has to re-collect the co-location instances of all candidates. Thus, this mining framework is poorly flexible.

1.2 Contributions

This paper proposes a new prevalent co-location pattern mining framework that tackles the two drawbacks. The key contributions are as follows.



(a) The joinless algorithm



(b) The iCPI-tree algorithm

Fig. 2. The execution time in each phase of the mining framework shown in Fig. 1.

(1) We design a fast enumerating clique approach. The neighbor relationships of instances are represented by using cliques.
(2) A co-location hashmap structure, which is constructed from the cliques, is designed to store compactly neighbor relationships of instances. All information about co-location instances of patterns is located under the hashmap structure.
(3) Our algorithm is no longer using the time-consuming generate-test model. Based on the co-location hashmap structure, the participation indexes of patterns are efficiently and easily calculated. When the minimum prevalence threshold is changed, the proposed method can quickly and adaptively give new results.

The rest of the paper is organized as follows. The basic concept of PCP mining is described briefly in Sect. 2. Section 3 represents the proposed mining framework in detail. A set of experiments is designed to demonstrate the advantage of our method in Sect. 4. Section 5 concludes the paper and provides directions for future work.

2 The Basic Concept

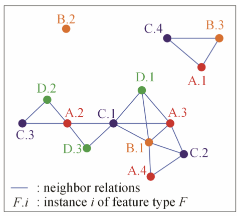
$F = \{f_1,\ldots,f_n\}$ is a set of spatial features and S is a set of their instances. Each instance in S is a vector $<$ feature type, ID, location $>$. A co-location pattern $c = \{f_1,\ldots,f_k\}$ is a subset of F whose instances have neighbor relationships to each other. The number of features in c, k, is called the size of c. If the distance between instances is smaller than a distance threshold d, the two instances have a neighbor relationship, e.g., $\overline{\text{A.1B.3}}$. A co-location instance of c, I, is a set of instances, $I \subseteq S$, which includes the instances of all features in c and forms a clique (all instances have neighbor relationships with each other). A set of all co-location instances is called the table instance of c, denoted $T(c)$.

The participation ratio (PR) of feature f_i in a pattern c is denoted as $PR(f_i, c) = \frac{\text{Number of instances of} f_i \text{in} T(c)}{\text{Total number of instances of} f_i \text{in} S}$. The participation index (PI) of c is defined as $PI(c) = \min\{PR(f_i, c)\}, f_i \in c$. Users give a minimum prevalence threshold, *min_prev*, if the participation index of c is larger than *min_prev*, c is called a prevalent co-location pattern.

Lemma 1: The participation ratio and the participation index are monotonically non-increasing with the size of the co-location pattern.

Proof: Please refer to [8] in detail.

Lemma 1 shows that if pattern c' is a super pattern of c, $c \subseteq c'$, we have a relationship $PI(c) \geq PI(c')$. If c is not prevalent, c' is also not a prevalent co-location. Lemma 1 is employed to quickly discover PCPs in our algorithm.



A	B
A.1	B.3
A.3	B.1
A.4	B.1
3/4	2/3
2/3	

A	C
A.1	C.4
A.2	C.1
A.2	C.3
A.3	C.2
A.4	C.2
4/4	4/4
4/4	

A	D
A.2	D.2
A.2	D.3
A.3	D.1
2/4	3/3
2/4	

B	C
B.1	C.1
B.1	C.2
B.3	C.4
2/3	3/4
2/3	

B	D
B.1	D.1
1/3	1/3
1/3	

C	D
C.1	D.1
C.1	D.3
C.3	D.2
2/4	3/3
2/4	

A	B	C
A.1	B.3	C.4
A.3	B.1	C.1
A.3	B.1	C.2
A.4	B.1	C.2
3/4	2/3	3/4
	2/3	

A	B	D
A.3	B.1	D.1
1/4	1/3	1/3
	1/4	

A	C	D
A.2	C.1	D.3
A.2	C.3	D.2
A.3	C.1	D.1
2/4	2/4	3/3
	2/4	

B	C	D
B.1	C.1	D.1
1/3	1/4	1/3
	1/4	

A	B	C	D
A.3	B.1	C.1	D.1
1/4	1/3	1/4	1/3
	1/4		

Fig. 3. An example of prevalent co-location pattern mining.

Definition 1 (Participating instance, ParI): The distinguish instance set of each feature in $T(c)$ is denoted as the participating instances of the feature in c.

For example, for candidate $c = \{A, B, C\}$, all co-location instances of it are $T(c) = \{\{A.1, B.3, C.4\}, \{A.3, B.1, C.1\}, \{A.3, B.1, C.2\}, \{A.4, B.1, C.2\}\}$. We obtain $ParI$(A, c) = {A.1, A.3, A.4}, $ParI$(B, c) = {B.1, B.3}, and $ParI$(C, c) = {C.1, C.2, C.4}. Thus, $PR(\text{A,c}) = \frac{|\{\text{A.1,A.3, A.4}\}|}{|\{\text{A.1,A.2,A.3, A.4}\}|} = \frac{|\text{ParI(A,c)}|}{4} = \frac{3}{4} = 0.75$, $PR(\text{B}, c) = 0.67$, and $PR(\text{C}, c) = 0.75$. Finally, $PI(c) = \min\{0.75, 0.67, 0.75\} = 0.67$. Assuming a user sets $min\_prev = 0.5$, since $PI(c) = 0.67 > min\_prev = 0.5$, thus {A, B, C} is a PCP.

It can be seen that if the participating instances of each feature in a pattern are obtained first, there is no need to collect all co-location instances of the pattern.

3 The New Prevalent Co-location Pattern Mining Framework

3.1 Enumerating Cliques

After materializing neighbor relationships of instances and converting to a star neighborhood structure which is developed by Yoo et al. [10], we design a finding clique strategy to quickly enumerate all cliques of an input data set. Table 1 shows the star neighborhoods of each center instance constructed from Fig. 3. Note that the neighbors of each center instance in the star neighborhood are sorted in descending order. Based on Fig. 3 and Table 1, we observe three conclusions: (1) The information of neighbor relationships of all sub-cliques is included in the maximal clique. For example, clique {A.1, B.3, C.4} includes all neighbor information of sub-cliques {A.1, B.3}, {A.1, C.4}, and {B.3, C.4}; (2) Each center instance combines with its star neighbors to construct several maximal cliques. For example, center instance A.2 and its neighbors {C.1, C.3, D.2, D.3} can generate two maximal cliques {A.2, C.1, D.3} and {A.2, C.3, D.2}. Note that the maximal notion here is local, only relative to one center instance. One maximal clique, that is constructed by one center instance, must be a clique (and maybe a maximal clique) in global. For example, {B.1, C.1, D.1} is a maximal clique constructed by center instance B.1, however, it is a clique in global since it has a super-clique {A.3, B.1, C.1, D.1}; (3) All neighbor relationships of instances of an input dataset are partitioned into cliques and it does not miss any neighbor relationships. For example, in Table 1, the spatial dataset in Fig. 3 is partitioned into 10 true cliques.

Based on the above observation, we design an enumerating clique strategy. Our main idea is, in each center instance and its neighbors, to generate a set of candidate maximal cliques, and then verify which of the candidates are true maximal cliques. To do this, we iterate verifying the directed sub-cliques of candidates. If any directed sub-clique is not a true clique, the current candidate clique is deleted and turned to the others.

Table 1. Enumerating true cliques from star neighborhoods

Instance	Star neighbors	Candidate maximum cliques	True cliques
A.1	B.3, C.4	{A.1, B.3, C.4}	{A.1, B.3, C.4}
A.2	C.1, C.3, D.2, D.3	{A.2, C.1, D.2}, {A.2, C.1, D.3} {A.2, C.3, D.2}, {A.2, C.3, D.3}	{A.2, C.1, D.3} {A.2, C.3, D.2}
A.3	B.1, C.1, C.2, D.1	{A.3, B.1, C.1, D.1} {A.3, B.1, C.2, D.1}	{A.3, B.1, C.1, D.1}
A.4	B.1, C.2	{A.4, B.1, C.2}	{A.4, B.1, C.2}
B.1	C.1, C.2, D.1	{B.1, C.1, D.1}, {B.1, C.2, D.1}	{B.1, C.1, D.1}
B.2	-	-	-
B.3	C.4	{B.3, C.4}	{B.3, C.4}
C.1	D.1, D.3	{C.1, D.1}, {C.1, D.3}	{C.1, D.1}, {C.1, D.3}
C.2	-	-	-
C.3	D.2	{C.3, D.2}	{C.3, D.2}
C.4	-	-	-
D.1	-	-	-
D.2	-	-	-
D.3	-	-	-

-: having no neighbors or cliques

For example, Table 2 lists the iterator steps when enumerating maximal cliques of B.1. The star neighbors of B.1 are {C.1, C.2, D.1} and they generate two candidate maximal cliques {B.1, C.1, D.1} and {B.1, C.2, D.1}. First, we obtain the directed sub-clique of the candidate, {C.1, D.1} and {C.2, D.1}. Next, getting the star neighborhood of the first instance in the directed sub-clique, for C.1 be {D.1, D.3} and for C.2 be ∅. After that, finding the intersection of the directed sub-clique with the star neighbors of C.1, {D.1} ∩ {D.1, D.3} = {D.1} and C.2, ∅ ∩ {D.1} = ∅. If the size of the intersecting result (denoted as *Flag*) is equal to the size of the directed sub-clique subtracting 1, it means all instances in the directed sub-clique may have neighbor relationships to each other and the current candidate maximal clique may be a true clique. If not, the current candidate is not a true clique and is deleted immediately.

Table 2. Enumerating maximal cliques of B.1

Candidate maximal cliques	Iterator step				True clique
	Directed sub-clique	Star neighborhoods of the first instance	Intersection	*Flag*	
{B.1, C.1, D.1}	{C.1, D.1}	{D.1, D.3}	{D.1}	1	Yes
{B.1, C.2, D.1}	{C.2, D.1}	Ø	Ø	0	No

The pseudo-code of enumerating cliques is plotted in Algorithm 1. The first phase generates star neighborhoods of a given dataset under a distance threshold (Step 1). The second phase iterates each item (instance) and generates a set of candidate maximal cliques of the current item (Steps 2–3). The third phase checks which candidate is a true clique. To do this, directed sub-cliques of the candidate are obtained, *subCand* (Step 9). Next, the star neighbors of the first element in directed sub-cliques are also acquired, *starNei* (Step 10). After that, the intersection of *subCand* and *starNei* is found (Step 11). If the size of the intersection is not equal to the size of the directed sub-clique, it means the current candidate maximal clique is not a true clique (Step 12). Then all directed sub-cliques of the current candidate are generated and added to the candidate maximal clique set (Step 14). Else the current candidate may be a true clique and the process go to the next iteration (Step 18). The final phase returns a set of true cliques (Step 26).

Algorithm 1. Discovering clique algorithm

```
Inputs: a spatial dataset S, and a neighbor distance threshold d.
Output: a set of all cliques SoTC.
Variables: SN: a hashmap structure; SoCC: a set of candidate cliques; s: the size of SoCC;
flag: marking if a candidate clique is a true/false clique.
1: SN ← generate_star_neighbors(S, d)
2: for item ∈ SN do
3:     SoCC ← generate_maximal_candidate_cliques(item)
4:     while SoCC not empty do
5:         cand ← SoCC.PopFront(); // get a candidate
6:         s ← cand.size()
7:         flag ← True //assuming the current candidate clique is a true clique
8:         while (s > 2) do
9:             subCand ← get_directed_sub_clique(cand)
10:            starNei ← SN.find(subCand.first())
11:            interSet ← subCand ∩ starNei
12:            if (interSet.size() != subCand.size()-1) do // not a true clique
13:                flag ← False
14:                SoCC.add(generate_direct_sub_candidate_cliques(cand))
15:                SoCC.remove(cand)
16:                break
17:            else // the current sub-candidate clique may be is a true clique, next iteration
18:                s--
19:            end if
20:        end while
21:        if (flag == True) do
22:            SoTC.add(cand)
23:        end if
24:    end while
25: end for
26: return SoTC
```

3.2 Constructing a Co-location Hashmap Structure

As can be seen in Sect. 3.1, all neighbor relationships of instances are partitioned into a set of cliques. Participating instances of all patterns can be discovered from these cliques. We design a co-location hashmap structure to compact these cliques so that information about participating instances of all patterns can be quickly obtained from the structure.
Definition 2 (Co-location hashmap structure): A co-location hashmap structure is a two-level nested hashmap structure whose key and value are denoted as:

(1) The key is a set of feature types of instances in the cliques.
(2) The value is a hashmap structure whose key and value are the feature type and the instance ID of instances in the cliques, respectively.

Figure 4 shows the co-location hashmap structure which is constructed from the cliques enumerated in Table 1. It can be seen that all cliques are compressed compactly in the co-location hashmap structure.

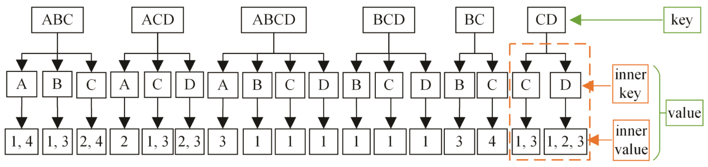


Fig. 4. The co-location hashmap structure of the data set in Fig. 3.

Algorithm 2. Building a co-location hashmap structure

```
Inputs: a set of cliques $SoTC$.
Output: a co-location hashmap structure $CoLHM$.
Variables: $key$, $value$.
1: for $clique \in SoTC$ do
2:     $key \leftarrow$ get_all_features($clique$)
3:     $value \leftarrow$ construct_inner_hashmap($clique$)
4:     $CoLHM \leftarrow$ update($key$, $value$, $CoLHM$)
5: end for
6: return $CoLHM$
```

Algorithm 2 shows the pseudo-code of building the co-location hashmap structure. The first phase creates the key and the value based on each clique (Steps 2–3). The second phase updates the co-location hashmap structure by the created key and value (Step 4). If the key has already existed in *CoLHM*, update the value; else directly add the key and the value into *CoLHM*. Finally, Algorithm 2 returns a co-location hashmap structure and it will be used to calculate the participation indexes of all patterns in Sect. 3.3.

3.3 Calculating Participation Indexes and Filtering PCPs

Based on the co-location hashmap structure, any patterns can be extracted from the keys and the information about participating instances of a pattern is obtained by the values. The participating instances of a pattern are embedded into two parts: one is the pattern itself and the other is the super-patterns of the pattern.

To describe simply, we use $<$ feature type: {instance ID} $>$ to represent instances of a feature. For example, the participating instances of pattern {B, C} can be acquired form key BC $<$ B: {3}, C: {4} $>$ and its super keys, BCD $<$ B: {1}, C: {1} $>$, ABCD $<$ B: {1}, C: {1} $>$ and ABC $<$ B: {1, 3}, C: {2, 4} $>$. Hence, the participating instances of {B, C} are $<$ B: {1, 3}, C: {1, 2, 4} $>$. This result is exactly the same as Fig. 3.

To make full use of Lemma 1, we first generate all possible patterns from the key set, and then start mining from the size 2 patterns. If a pattern is not prevalent, all its supersets can be deleted directly.

Algorithm 3 is designed to quickly calculate the participation indexes of patterns based on the co-location hashmap structure. The first phase takes all keys in the co-location hashmap structure and generates the power sets of these keys (Steps 1–3). All possible patterns, *candPatts,* are generated and sorted by their size (Step 4). The second phase gets each pattern, *patt* and finds its participating instances, *ParI*, by gathering the values of all keys that are supersets of *patt* (Steps 6–7). The third phase calculates PI of *patt* based on its participating instances (Step 8). If the PI of *patt* is larger than *min_prev*, it is prevalent and added into *PSCs* (Steps 9–11). Else all possible patterns that are supersets of the pattern are removed from *candPatts* (Step 12). Our algorithm makes full use of Lemma 1 to prune unnecessary possible patterns in advance.

Algorithm 3. Calculating PIs and filtering PCPs

```
Inputs: a co-location hashmap structure CoLHM and a prevalence threshold min_prev.
Output: all prevalent spatial co-location patterns PSCs.
Variables: key, value.
1: for each key in CoLHM.keys() // get all keys in the co-location hashmap structure
2:     candPatts ← gen_power_sets(key)
3: end for
4: candPatts ← sort_by_size(candPatts)
5: while candPatts is not empty do
6:     patt ← candPatts.PopFront()
7:     ParI ← (patt, CoLHM)
8:     PI ← calculate_PI(ParI)
9:     if (PI ≥ min_prev) do
10:        PSCs.add(patt)
11:    else
12:        candPatts.delete_all_superset(patt)
13:    end if
14: end while
15: return PSCs
```

Figure 5 shows the proposed mining framework. Different from the framework shown in Fig. 1, our framework has no time-consuming collecting co-location instance phase.

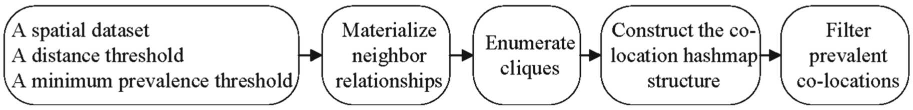

A spatial dataset
A distance threshold
A minimum prevalence threshold
Materialize neighbor relationships
Enumerate cliques
Construct the co-location hashmap structure
Filter prevalent co-locations

Fig. 5. The proposed prevalent co-location pattern mining framework.

3.4 The Time Complexity Analyses

As shown in Fig. 5, our mining algorithm has four phases. The first phase materializes neighbor relationships and converts to the star neighborhood structure and the computational complexity of this phase is about $O(n^2 \times \frac{d^2}{A})$ [25] where d is the distance threshold, n is the number of instances and A is the area of the input dataset space.

Assuming s_{avg} is the average size of neighbors of each instance. The computational complexity of enumerating cliques is about $O\left(n \times s_{avg}^2\right)$. The third phase constructs the co-location hashmap structure and the computational complexity of this phase is about $O(L)$ where L is the number of cliques. The final phase calculates PIs and filters prevalent co-location patterns and its computational complexity is about $O(l \times m)$ where l and m are the numbers of items in the co-location hashmap structure and numbers of all possible

Table 3. Parameters of the synthetic dataset

Parameters	Values
Number of features	15
Number of instances	20000
Frame size (D × D)	1000, 10000
d	15
min_prev	0.2
clumpy	1
across	0
overlap	0

* *clumpy*, *across,* and *overlap* refer to [10] in detail

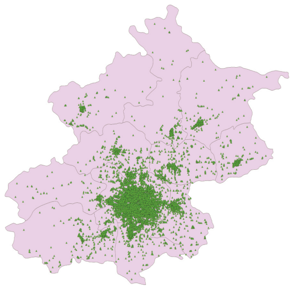


Fig. 6. The distribution of the real dataset.

patterns. Therefore, the total computational complexity of our mining framework is about $O\left(n^2 \times \frac{d^2}{A}\right) + O\left(n \times s_{avg}^2\right) + O(L) + O(l \times m)$.

4 Experiment Evaluations

We design a set of experiments to demonstrate the proposed mining framework is efficient. We chose the joinless algorithm [10] which is known as a correct and efficient algorithm for finding PCPs based on the framework in Fig. 1. Our experiments are coded by C++ and performed on a PC machine with 16 GB main memory.

4.1 Experimental Datasets

Both synthetic and real datasets are used in our experiments. Table 3 shows a summary of the synthetic dataset which is generated by a generator developed by Shekhar et al. [8]. We use the real data set that is a facility point data set from Beijing, China, that contains 25,276 items (instances) belonging to 12 feature types such as residential area, company, and restaurant. The distribution of the real data set is shown in Fig. 6.

4.2 Mining Comparisons

Comparison of the Costs of Computation Complexity Factors: We show the execution time in each phase of our algorithm and the joinless algorithm in Table 4. Since the two algorithms use star neighborhoods, Table 4 only lists the execution time of the last three phases in each framework. In the sparse dataset, since the number of co-location instances is small, PIs of patterns are also small and many candidates are pruned, thus the execution time of joinless is a little faster than our algorithm. However, in the dense dataset, our algorithm is faster than joinless. It can be seen that most of the execution time of joinless is devoted to collecting co-location instances. Our algorithm without collecting co-location instances shows less execution time.

Table 4. Effects of the decision by the two mining frameworks

Execution time (s)	Joinless		Execution time (s)	Our algorithm	
	Sparse	Dense		Sparse	Dense
Generate candidates	0.005	0.051	Enumerate cliques	0.044	22.006
Collect co-location instances	4.017	47.172	Construct hashmap	0.374	0.95
Filter prevalent patterns	0.026	0.294	Filter prevalent patterns	6.332	6.03
Total time	4.048	47.517	Total time	6.75	28.986

Effect of the Minimum Prevalence Threshold: Figure 7 shows the effect of the proposed algorithm when changing minimum prevalence thresholds on the real data set with the distance threshold set to 300m. It is clear to see that the execution time of joinless is very expensive when *min_prev* is small. In this case, many candidates become PCPs and it must be collect co-location instances of all these patterns. With the increase of *min_prev*, the execution time in joinless decreases. While our algorithm is robust because it is designed to avoid the effect of changing *min_prev*. When changing *min_prev*, our algorithm only needs to calculate the PIs of patterns from the co-location hashmap structure without other redundant operations. Thus, it can quickly give new mining results to users.

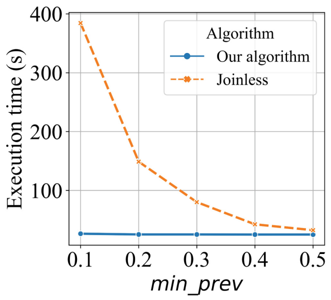


Fig. 7. The effect of the *min_prev* threshold.

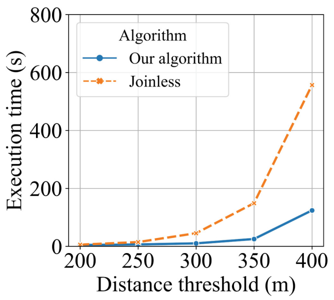


Fig. 8. The effect of the distance threshold.

Effect of the Distance Threshold: The effect of distance thresholds also is evaluated in the real dataset with *min_prev* is set to 0.2. As shown in Fig. 8, with the increase of the distance threshold, the execution time of the two algorithms also increase. However, our algorithm shows better performance.

Effect of Numbers of Instances: The final experiment evaluates the effect of the number of instances of the compared algorithms. We set the synthetic data sets with a frame size is 5000 × 5000, the number of features is 15, $d = 30$, $min\_prev = 0.2$, and the number of instances is changed. As shown in Fig. 9, with the increase in the number of instances (data sets become larger), the execution time of the two algorithms also increases. However, the performance of Joinless degenerates quickly in the situation of large data sets, while the proposed algorithm shows better scalability.

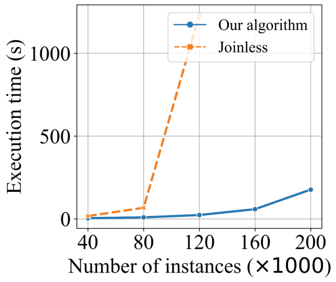


Fig. 9. The effect of numbers of instances.

5 Conclusion

A new PCP mining framework is proposed in this paper. The proposed framework partitions neighbor relationships of instances into cliques and constructs a co-location hashmap structure based on these cliques that compact the neighboring instances. By using the structure, the participation indexes of patterns can be calculated quickly without collecting their co-location instances. If users change the minimum prevalence threshold, our algorithm only needs to re-calculate the participation indexes and directly give new mining results. Thus, the proposed algorithm is robust with the minimum prevalence threshold. The experimental result indicates that the performance of our algorithm is better than collecting co-location instance-based mining algorithms.

It can be seen that a heavy job in our algorithm is enumerating cliques. However, this step is quite easy to implement in parallel computing. Thus, our future work focuses on transforming the proposed algorithm into an efficient parallel mining method.

References

1. Yu, W.: Spatial co-location pattern mining for location-based services in road networks. Expert Syst. Appl. **46**, 324–335 (2016)
2. Akbari, M., Samadzadegan, F., Weibel, R.: A generic regional spatio-temporal co-occurrence pattern mining model: a case study for air pollution. J Geogr Syst. **17**, 249–274 (2015)
3. Mohan, P., Shekhar, S., Shine, J.: A neighborhood graph based approach to regional co-location pattern discovery: a summary of results. In: 19th ACM SIGSPATIAL, pp. 122–132. ACM, NY (2011)
4. Cai, J., Deng, M., Liu, Q.: Nonparametric significance test for discovery of network-constrained spatial co-location patterns. Geogr. Anal. **51**, 3–22 (2019)
5. Deng, M., He, Z., Liu, Q.: Multi-scale approach to mining significant spatial co-location patterns. Trans. GIS **21**, 1023–1039 (2017)
6. Wang, S., Huang, Y., Wang, X.: Regional co-locations of arbitrary shapes. In: Advances in Spatial and Temporal Databases, pp. 19–37. Springer, Berlin (2013). https://doi.org/10.1007/978-3-642-40235-7_2
7. Kishor, P., Porika, S.: An efficient approach for mining positive and negative association rules from large transactional databases. In: ICICT, pp. 1–5. IEEE, India (2016)
8. Shekhar, S., Huang, Y.: Discovering spatial co-location patterns: a summary of results. In: Advances in Spatial and Temporal Databases, pp. 236–256. Springer, Berlin (2001). https://doi.org/10.1007/3-540-47724-1_13

9. Yoo, J., Shekhar, S., Smith, J., Kumquat, J.: A partial join approach for mining co-location patterns. In: 12th Annual ACM International Workshop on Geographic Information Systems, pp. 241–249. ACM, New York (2004)
10. Yoo, J., Shekhar, S.: A joinless approach for mining spatial colocation patterns. IEEE Trans. Knowl. Data Eng. **18**, 1323–1337 (2006)
11. Wang, L., Bao, Y., Lu, J., Yip, J.: A new join-less approach for co-location pattern mining. In: 8th IEEE International Conference on Computer and Information Technology, pp. 197–202. Sydney (2008)
12. Wang, L., Bao, Y., Lu, Z.: Efficient discovery of spatial colocation patterns using the iCPI-tree. The Open Inf. Syst. J. **3**, 69–80 (2009)
13. Yoo, J., Boulware, D., Kimmey, D.: A parallel spatial co-location mining algorithm based on mapreduce. In: International Congress on Big Data, pp. 25–31 (2014)
14. Yoo, J., Boulware, D., Kimmey, D.: Parallel co-location mining with MapReduce and NoSQL systems. Knowl Inf. Syst. (2019)
15. Andrzejewski, W., Boinski, P.: Efficient spatial co-location pattern mining on multiple GPUs. Expert Syst. Appl. **93**, 465–483 (2018)
16. Sainju, A., Aghajarian, D., Jiang, Z., Prasad, S.: Parallel grid-based colocation mining algorithms on GPUs for big spatial event data. IEEE Trans Big Data, pp. 1–1 (2018)
17. Andrzejewski, W., Boinski, P: Parallel approach to incremental co-location pattern mining. Information Sci. **496**, 485–505 (2019)
18. Leibovici, D., Claramunt, C., Guyader, D., Brosset, D.: Local and global spatio-temporal entropy indices based on distance-ratios and co-occurrences distributions. Int. J. Geogr. Inf. Sci. **28**, 1061–1084 (2014)
19. Huang, Y., Pei, J., Xiong, H.: Mining co-location patterns with rare events from spatial data sets. GeoInformatica **10**, 239–260 (2006)
20. Wang, L., Wu, P., Chen, H.: Finding probabilistic prevalent colocations in spatially uncertain data sets. IEEE Trans. Knowl. Data Eng. **25**, 790–804 (2013)
21. Ouyang, Z., Wang, L., Wu, P.: Spatial co-location pattern discovery from fuzzy objects. Int. J. Artif Intell. Tools **26**, 1750003 (2016). https://doi.org/10.1142/S0218213017500038
22. Yao, X., Chen, L., Peng, L., Chi, T.: A co-location pattern-mining algorithm with a density-weighted distance thresholding consideration. Inf. Sci. **396**, 144–161 (2017)
23. Yoo, J., Bow, M.: Mining top-k closed co-location patterns. In: International Conference on Spatial Data Mining and Geographical Knowledge Service, pp. 100–105. IEEE, Fuzhou (2011)
24. Wang, L., Zhou, L., Lu, J., Yip, J.: An order-clique-based approach for mining maximal co-locations. Inf. Sci. **179**, 3370–3382 (2009)
25. Wang, L., Bao, X., Zhou, L.: Redundancy reduction for prevalent co-location patterns. IEEE Trans. Knowl. Data Eng. **30**, 142–155 (2018)
26. Wang, L., Bao, X., Chen, H., Cao, L.: Effective lossless condensed representation and discovery of spatial co-location patterns. Inf. Sci. **436–437**, 197–213 (2018)
27. Boinski, P., Zakrzewicz, M.: Collocation pattern mining in a limited memory environment using materialized iCPI-tree. In: Data Warehousing and Knowledge Discovery, pp. 279–290. Springer, Berlin (2012). https://doi.org/10.1007/978-3-642-32584-7_23