Utah State University

DigitalCommons@USU

All Graduate Theses and Dissertations

Graduate Studies

5-2009

Optimal Candidate Generation in Spatial Co-Location Mining

Zhongshan Lin

Utah State University

Follow this and additional works at: <https://digitalcommons.usu.edu/etd>

Part of the [Computer Sciences Commons](#)

Recommended Citation

Lin, Zhongshan, "Optimal Candidate Generation in Spatial Co-Location Mining" (2009). *All Graduate Theses and Dissertations*. 377.

<https://digitalcommons.usu.edu/etd/377>

This Thesis is brought to you for free and open access by the Graduate Studies at DigitalCommons@USU. It has been accepted for inclusion in All Graduate Theses and Dissertations by an authorized administrator of DigitalCommons@USU. For more information, please contact digitalcommons@usu.edu.

OPTIMAL CANDIDATE GENERATION IN SPATIAL CO-LOCATION MINING

by

Zhongshan Lin

A thesis submitted in partial fulfillment
of the requirements for the degree

of

MASTER OF SCIENCE

in

Computer Science

Approved:

SeungJin Lim
Major Professor

Stephen Allan
Committee Member

Curtis Dyreson
Committee Member

Byron R. Burnham
Dean of Graduate Studies

UTAH STATE UNIVERSITY
Logan, Utah

2009

Copyright © Zhongshan Lin 2009
All Rights Reserved

ABSTRACT

Optimal Candidate Generation in Spatial Co-location Mining

by

Zhongshan Lin, Master of Science

Utah State University, 2009

Major Professor: Dr. SeungJin Lim
Department: Computer Science

Existing spatial co-location algorithms based on levels suffer from generating extra, nonclique candidate instances. Thus, they require cliqueness checking at every level. In this thesis, a novel, spatial co-location mining algorithm that automatically generates co-located spatial features without generating any nonclique candidates at any level is proposed. Subsequently, this algorithm generates fewer candidates than other existing level-wise, co-location algorithms without losing any pertinent information. The benefits of this algorithm have been clearly observed at early stages in the mining process.

(77 pages)

ACKNOWLEDGMENTS

To my major advisor, Dr. Lim, and my committee members, Dr. Allan and Dr. Dyreson, who guided my study and research at Utah State University. To Ms. Cook, who checked and improved the grammar and formatting of this thesis. Most of all, to my wife, Jianjing, who has supported and continues to support my life and my work at Utah State.

Zhongshan Lin

CONTENTS

	Page
ABSTRACT.....	iii
ACKNOWLEDGMENTS	iv
LIST OF TABLES	vii
LIST OF FIGURES	viii
CHAPTER	
1 INTRODUCTION	1
1.1 Spatial Databases and Data Mining	1
1.2 Spatial Co-location Mining	2
1.2.1 Applications	3
1.2.2 Challenges	6
2 RELATED WORK	10
3 PRELIMINARIES	18
4 PREVIOUS ALGORITHMS	23
4.1 Spatial Association Rules Mining	23
4.2 Join-Based Co-location Mining	27
4.3 Partial Join-Based Co-location Mining	29
4.4 Joinless Co-location Mining	32
5 NEIGHBOR CLUSTER ALGORITHM	39
5.1 Neighbor Cluster	39
5.2 Level-wise Generation of Co-location Type Lists	42
5.3 The Algorithm	46
6 COMPLEXITY ANALYSIS	49
7 EXPERIMENTAL ANALYSIS	53
7.1 Synthetic Dataset	53
7.1.1 Space Requirement	55

7.1.2	Running Time	56
7.2	Real Dataset	58
7.2.1	Space Requirement	59
7.2.2	Running Time	59
7.3	Discussion	61
7.3.1	Effect of Level	61
7.3.2	Effect of Threshold	62
7.3.3	Effect of Dimensionality and Data Size.....	62
7.3.4	Effect of Number of Candidates on Running Time	62
8	CONCLUSION.....	65
	REFERENCES	67

LIST OF TABLES

	Page
1 Example Transaction Database Table.....	7
2 Neighbor Clusters (NCs) of the Spatial Feature Instances Shown in Figure 3.....	41
3 Level-wise Generation of Candidate Co-location Type List $\langle A, B, C, D \rangle$ from the Neighbor Clusters Shown in Table 2.	45
4 Fifteen Synthetic Datasets Used in the Experiment.....	54
5 Four Real Datasets Used in the Experiment	58

LIST OF FIGURES

	Page
1 Example spatial dataset.....	4
2 Example spatial dataset.....	8
3 Spatial types <i>A</i> , <i>B</i> , <i>C</i> , <i>D</i> and their instances.....	19
4 Grid partitioning of the spatial space presented in Figure 3	30
5 Another partitioning strategy for the spatial space presented in Figure 3	32
6 Comparison of numbers of candidates generated by both algorithms on the synthetic dataset with 40 feature types and 30,000 instances	56
7 Comparison of both algorithms on running times on three synthetic datasets	57
8 Comparison of both algorithms on running times on two synthetic datasets	57
9 Comparison of numbers of candidates generated by both algorithms on the Oregon dataset	59
10 Comparison of numbers of candidates generated by both algorithms on the Washington dataset	60
11 Comparison of numbers of candidates generated by both algorithms on the Indiana dataset	60
12 Comparison of numbers of candidates generated by both algorithms on the Kansas dataset.....	61
13 Comparison of both algorithms on running times on the Kansas and Indiana datasets	63
14 Comparison of both algorithms on running times on the Oregon and Washington datasets	64

CHAPTER 1

INTRODUCTION

1.1 Spatial Databases and Data Mining

A spatial database is a database capable of storing, indexing, and querying spatial data. Common spatial data include computer-aided design data and geographical data [19]. Spatial databases are usually implemented as extensions of traditional relational databases, by supporting complex spatial data types in addition to simple data types. Common spatial data types include point, line string, and polygon. The point data type is used to represent spatial objects whose shapes are not interesting to users. For example, a user may be interested in the location of a building but not its shape or dimensions. The line string data type typically represents spatial objects whose widths are negligible compared to their lengths and / or trends, for example, rivers. A spatial object of this type is usually represented as a set of points in the order of its trend. The polygon data type is used to represent spatial objects whose shapes are interesting to users. This data type can represent spatial objects of any shape. A spatial object of polygon data type is usually represented as a set of points in some order, where each point is a vertex of the polygon.

The support of complex spatial data types also introduces challenges. One of these challenges is the spatial join. In relational databases, table joins involve simple and cheap value comparisons, but in spatial databases, spatial joins are usually based on spatial relationships, such as overlapping, disjointing, touching, intersecting, etc. An example query involving spatial joins is given as below:

```

SELECT m.Name, r.Name
FROM Mountains m, Rivers r
WHERE ST_Intersects(m.geom, r.geom)

```

This query joins a mountain table and a river table, and returns a set of (mountain, river) name pairs where the mountains intersect the rivers. The “geom” fields are the spatial data fields. The checking of intersections between mountains and rivers is done using the *ST_Intersects* function provided by PostGIS [15]. Spatial relationship calculations are very complex and expensive compared with simple value comparisons in traditional table joins. This forces researchers and spatial database users to either find efficient ways to do the spatial joins, or avoid spatial joins as much as possible.

As the applications of spatial databases grow, spatial data mining has been developed to discover interesting, previously unknown patterns in spatial databases. The demand for processing massive spatial data is increasing rapidly, particularly in science (GIS, ecology, etc.), engineering (i.e., traffic control) and industry (i.e., GPS navigation and mobile/sensor network). Consequently, it is necessary to develop efficient spatial data mining techniques to help domain experts discover useful knowledge from the given databases.

1.2 Spatial Co-location Mining

Given a finite set of Boolean spatial features and their instances, spatial co-location mining seeks to discover a set of features whose instances are frequently co-located in close proximity. For the purposes of this thesis co-located instance means a set of spatial instances that form a clique. A clique is a set of spatial instances that are closely

related to each other. Further, we are interested in co-located instances with distinct spatial features only. That is, for any two instances in a clique, their feature types are different, and they are neighbors. A clique indicates strong coherence between its members. An example of spatial features and instances is presented in Figure 1. In this example, $\langle A_1, B_2, C_2, D_1 \rangle$ forms a clique, and $\langle A_2, B_3, C_1, D_2 \rangle$ forms another clique. Although the two cliques are connected through instance B_1 , the entire set of instances does not form a clique, and D_1 and C_1 are too far away from each other to have any relationship.

Depending on applications, examples of spatial features can vary from natural habitats of different animals or plants, to locations of high crime rates or pollution. The historical discovery of the water-borne nature of the Asiatic cholera epidemic that swept through London in 1854 is a well-known example of a co-located feature discovery. During this cholera epidemic, "When the government authorities turned off the water pump, the cholera began to subside" [16:186].

1.2.1 Applications

A typical application domain of co-location mining is in the field of ecology. Co-location mining is often used to discover symbiotic relationships between biological species in a certain area. Symbiosis is a phenomenon in which more than one biological species live closely to and have some significant interaction with each other. Symbiosis can be categorized as mutualism, commensalism, parasitism, competition, and neutralism. While these categories differ, they all share the same feature: two or more biological species live closely to and interact with each other.

Figure 1. Example spatial dataset.

A famous example of symbiosis is the symbiotic relationship between Ocellaris clownfish and sea anemones. Ocellaris clownfish live among sea anemones' stinging tentacles and are protected by them. On the other hand, Ocellaris clownfish are territorial and keep anemone-eating fish away from the sea anemones. Another interesting fact is that the Ocellaris clownfish produces special mucus to protect them from the stinging tentacles of the sea anemones.

To mine symbiotic relationships, a region, such as a continent, nation, or state, and a set of biological species are selected. Instances of those biological species are gathered in the region, and their habitats are annotated using their specie types. Distances between these habitats are calculated, and the neighborhood relationships between them are determined using a user-defined distance threshold. A co-location mining algorithm is then applied to the habitats, and the symbiotic relationships among the biological species are discovered.

Beyond symbiotic relationships, additional types of relationships also exist between biological species and other objects, such as rivers, lakes, deserts, etc. An example of this kind of relationship follows. A cactus has spine-like leaves and lives in

deserts, which are extremely dry and hot. The Turkish pine lives in dry and hot areas, too, and it has needle like leaves. A co-location rule for this example can be written as:

plants with small, needle like leaves → dry and hot areas.

As modern technologies develop, human interaction with nature increases. These interactions include increased pollution and the exploration of nature. Such interactions can affect animals and plants living in the polluted or explored areas, and change their behavior.

Another application domain of co-location mining is the area epidemiology and public health. Some diseases have a high correlation to the environments in which they occur. For example, people living close to polluted areas are often more likely to get certain cancers, and people infected with Avian influenza usually live or work close to poultry and fowl habitats. The water-borne nature of Asiatic cholera mentioned above is another good example. Co-location mining can be applied to this by selecting a set of features which can potentially affect human health, treating each disease as a feature and its occurrences as instances.

Another key area to which co-location mining can be applied is marketing. An example application is automobile marketing. Assume that a car dealer is selling three kinds of automobiles: new trucks, new sedan cars, and used cars. The dealer can collect previous buyers' home locations from sales records and annotate them as three spatial feature types: "new truck purchased," "new sedan purchased," and "used car purchased." This information, together with other geographical spatial features (like "mountainous area," "river," "urban area," and "college campus") and related spatial instances, is then

used as inputs and fed to the co-location mining algorithms. Patterns between these spatial feature types are mined and that could be potentially interesting to the car dealer.

Example patterns may include: *mountainous area* $\rightarrow$ *new truck purchased*

(indicating that a person living close to mountainous areas is likely to buy a truck),

urban area $\rightarrow$ *new sedan purchased* (indicating that a person living close to an urban

area is likely to buy a sedan car), and *college campus* $\rightarrow$ *used car purchased*

(indicating that a person living close to college campuses is likely to buy a used car).

These kinds of patterns help dealers do location sensitive marketing, like advertising trucks in mountainous areas and sedan in urban areas.

1.2.2 Challenges

The spatial co-location mining problem has its root in the traditional association rules mining problem, which has been extensively studied. However, traditional association rules mining is based on the assumption that the data are given in a transactional dataset in which transactions are disjointed sets of items. An example transactional dataset is shown in Table 1. In this dataset, the three transactions are completely disjointed and independent. The only relationship between them is that they are ordered according to their timestamps, which is irrelevant to data mining. The notion of proximity between data objects is also absent in traditional association rules mining. In co-location mining applications, the natural transactions are absent. A simple co-location dataset example is presented in Figure 2. In this figure, each node represents a spatial instance, and each edge between two nodes represents the neighborhood relationship between the two instances. We can clearly identify a group of instances $\langle A_1, B_4, D_3 \rangle$.

They strongly cohere because they are neighbors to each other, and they do not have any relationships with instances outside the group. Similarly, we can see that $\langle A_2, B_2, C_2, D_1 \rangle$ and $\langle A_3, B_3, C_1, D_2 \rangle$ are also two strongly cohered groups of instances. However, $\langle A_2, B_2, C_2, D_1 \rangle$ and $\langle A_3, B_3, C_1, D_2 \rangle$ are not independent groups like $\langle A_1, B_4, D_3 \rangle$, because there is a weak relationship between them, through instance B_1 . If we treat $\langle A_2, B_2, C_2, D_1 \rangle$ and $\langle A_3, B_3, C_1, D_2 \rangle$ as disjoint transactions, we lose the relationships between B_1 and C_2 , and B_1 and D_2 ; if we combine them as a single transaction, we are assuming that they strongly cohere, which does not actually hold. Hence, it is not a trivial job to transactionize the spatial datasets. Consequently, traditional association rules mining algorithms are not suitable for spatial co-location mining. This motivates researchers to find new ways to model the co-location mining problem.

Another challenge of co-location mining is the huge number of needed calculations. A dataset with n spatial instances has as many as $2^n - n - 1$ potential co-location instances. In real applications, the number n itself would be very big. Thus, a quick and cheap co-location instance discovery approach is highly desirable. Existing co-location mining algorithms [8, 12, 17, 23, 24] largely have their foundation in the

Table 1. Example Transaction Database Table.

ID	Time	Transaction
1	07/15/2008 09:20:08 PM	Egg, Cake, Candle, Soy Milk
2	07/15/2008 09:34:21 PM	Hat, Glove, Coat
3	07/15/2008 10:02:17 PM	Beef, Chicken, Rice

Figure 2. Example spatial dataset.

generate-and-test strategy of an Apriori algorithm [1]: that is, the algorithms traverse the spatial instances set one level at a time, generate candidates, and then test if each candidate forms a clique. These existing co-location mining approaches are likely to generate and test a huge number of candidates, which creates a bottleneck in the algorithm's performance.

In this thesis, a novel, level-wise, spatial co-location algorithm called the neighbor cluster algorithm (NCA) based on a new data structure called the neighbor cluster (NC) is proposed. This algorithm is equipped with a linear set intersection operation. The neighbor cluster data structure materializes the neighborhood relationships between spatial instances and enables quick identification of cliques in the co-location mining process. The contributions of this work are two-fold:

1. Optimal candidate generation: The NC data structure guarantees any candidate co-location instance generated at any level to be a clique, and hence the proposed algorithm generates an optimal number of candidates without losing information at each

level. This helps us incrementally mine co-location patterns while maximizing efficiency in the mining process.

2. Performance gain: NCA optimizes co-location mining by generating less candidate co-location instances at each level than other level-wise algorithms. The subsequent savings from the generation of a lower number of candidates and the avoidance of cliqueness checking helps improve the performance of the mining process. As will be shown later, the proposed algorithm improves the overall performance without violating the completeness and the correctness of the solution to the problem.

The remainder of the thesis is organized as follows. In Chapter 2, previous research efforts and achievements related to co-location mining are introduced, followed by relevant basic concepts in Chapter 3. In Chapter 4, four models and algorithms highly related to this work are introduced and compared. The proposed co-location mining algorithm is presented in Chapter 5. The analytical and empirical reviews of the proposed algorithm are given in Chapters 6 and 7, respectively. Finally, conclusions are drawn in Chapter 8.

CHAPTER 2

RELATED WORK

Spatial data mining is an active research topic because of the increasing demand on processing huge spatial data gathered from such areas as GIS applications and scientific research. Some generalized frameworks for spatio-temporal patterns mining have been proposed [22].

An offshoot of spatial association rules mining, co-location mining is an important member of the spatial data mining field. The problem of spatial association rules mining was first presented in [10], in which the authors proposed a reference feature centric model [18] to discover spatial association rules. In this model, the user first specifies a reference feature (like "park"). Spatial instances of other spatial features close to instances of the reference feature are retrieved. Each set of spatial instances that are neighbors to an instance of the reference feature is considered as a transaction, and traditional association rules mining algorithms are applied to discover association rules related to this reference feature. The basic idea behind this approach is converting the spatial datasets to nonspatial transaction datasets, and applying traditional data mining techniques. This idea was popular in the early stages of spatial data mining research. However, this approach has a major drawback in that not every application has a clear reference feature. For example, in the general field of ecology, scientists might be interested in the co-location patterns between biological species without special interest in a certain species. Using this approach, scientists would need to use each species as the reference feature and do multiple mining. Such a strategy would result in too much

human involvement and would be likely to miss some interesting patterns. In Chapter 4, this model will be evaluated in further detail.

In [12], the author proposes an Apriori-like algorithm for co-location mining. The definition of co-location patterns in this paper, however, is different from the one we are interested in. This approach uses the number of instances as the prevalence measure of the co-location patterns. However, this prevalence measure is not necessarily anti-monotone because a single spatial instance might participate in multiple instances of a co-location pattern. To make the prevalence measure anti-monotone such that the pruning works, this approach adds a restriction that each spatial instance can participate in only one instance of a co-location pattern. This restriction leads to another problem: the different ordering of spatial features and the steps of mining process itself leads to different co-location instances and thus the prevalence of a co-location pattern calculated might differ, too. Another problem with this approach is that each co-location instance is represented using its centroid. For a certain co-location instance, this algorithm will only join it with the nearest instance of a new feature type, according to the distance between the centroid and instances of new feature types. So, the co-location instances discovered do not necessarily form a clique.

In [17], an event-centric model is proposed to solve the co-location mining problem. This model defines clique instances as co-location instances and does not require any reference features. The authors adopt an Apriori-like generate-and-test approach to co-location mining. The generation of co-location instances is achieved using spatial join operations [2, 7, 14] or instance join operations. Compared with the approach

proposed in [12], the event centric model adopts a prevalence measure (called participation ratio) that is naturally anti-monotone, and the mining result is independent of the ordering of the spatial features and the steps of the mining process. However, the algorithm requires a large amount of spatial join or instance join operations, which are expensive.

To reduce the number of instance join operations or spatial join operations, a partial-join algorithm was proposed in [23]. This approach is based on plane partitioning. It partitions a plane such that the maximum distance between any two spatial instances in a partition is smaller than or equal to the distance threshold that defines the neighborhood relationships between spatial instances. After partitioning, spatial instances within the same partition automatically form a clique. Instance join or spatial join operations are only needed for neighborhood relationships crossing partition boundaries. This approach reduces the amount of instance join and spatial join operations, and it achieves a better performance. However, it often still requires a large amount of spatial join or instance join operations.

To totally avoid the joins, in [24], the authors propose two data structures to materialize spatial data. Based on these two data structures, a joinless algorithm is created. Although spatial data could be better organized using the data structures, the co-location instances generated by the joinless algorithm are not necessarily cliques. Subsequently, this model requires cliqueness checking for each candidate co-location instance to make sure that all patterns discovered are correct. In Chapter 4, the join-based algorithm, the partial-join algorithm and the joinless algorithm will be evaluated in greater detail.

In [26], the authors adopt a multi-way join and plane partitioning-based approach, which allows mining co-location patterns while simultaneously discovering neighborhood relationships between spatial instances.

Clustering-based approaches have also been considered. In [6], the authors proposed a vertical-view approach and a horizontal-view approach. The vertical-view approach partitions the spatial space to an $m \times m$ grid, and the algorithm finds features appearing in each cell on the grid. By doing this, each cell on the grid is converted to a transaction, and a traditional association rules mining algorithm is applied to discover frequently co-located features. However, the result of this approach depends on the granularity of the plane partitioning. The horizontal-view approach performs spatial clustering on instances of each spatial feature and finds co-location patterns according to the intersections of spatial features' clusters.

In [9], a similar clustering-based approach was adopted in such a way that if instances of feature A are more dense in feature B 's neighborhood than its global density, A and B are assumed to be co-located. These clustering-based approaches assume that spatial instances with the same feature are likely to be located in close proximity, which may not be true in some applications.

It is worthwhile to mention that in [13], the authors introduce more complex spatial co-location relationships between different features and instances of the same feature, such as self-co-location, self-exclusion, positive and exclusive relationships between different features, and so forth. These relationships may be of particular interest depending on the application.

Frequent subgraph mining [11, 21] is also related to co-location mining. Both frequent subgraph mining and co-location mining are intended to discover frequent substructures among spatial objects, but co-location mining is more interested in discovering co-location relationship between spatial features while frequent subgraph mining emphasizes the structural characteristics (i.e., isomorphism) of graphs.

Co-location mining has been extended in several different directions. In [8], the authors point out that the current co-location mining model might miss some important patterns when some features have far less instances than others. For example, for co-location pattern $\langle \textit{Smoking}, \textit{Lung Cancer} \rangle$, the number of instances of *Smoking* might be far smaller than the number of instances of *Lung Cancer*, because lung cancer might be caused by many reasons, like air pollution, etc., and smoking is just one of them. So, the overall coherence between *Smoking* and *Lung Cancer* might be weak. However, *Smoking* might be highly related to *Lung Cancer* in the sense that smoking will likely lead to lung cancer. To resolve this issue, the authors adopted a new prevalence measure considering the maximum participation ratios of co-location patterns. A weak monotonic property of the new prevalence measure was discovered. Another mining algorithm was later proposed based on this property [8].

Instead of considering all spatial instances as points in previous co-location mining models, in [20], the authors propose a buffer-based model to mine co-location patterns while preserving the spatial instances' shapes. The buffer-based model extends spatial objects with a length d buffer, where d is a user-defined buffer size. The buffer of a spatial object represents the area affected by the spatial object. The buffering operation

is usually supported by the spatial databases. A co-location instance is then a set of spatial instances whose buffered areas intersect each other, and the size of the intersection area indicates the strength of the coherence of the co-located instances. Compared with the event-centric model, the buffer-based model has two advantages. First, it preserves the shapes of spatial instances. Second, it is spatially more meaningful. However, this model also involves a lot of complex and expensive spatial operations that can greatly affect the performance.

The above co-location mining models and algorithms are designed to mine co-location patterns in a global space. However, given the same global space, different applications might be interested in different parts of the global space, such as a specific country or state. In such instances, it would be unnecessarily time consuming for each application to mine co-location patterns in a specific zone from scratch. In [4], the authors propose an algorithm to support efficient zonal co-location mining. This algorithm indexes different parts of the global space using a data structure called *clQuad-tree*. Co-location patterns and instances within each part are discovered and stored on *clQuad-tree*, as well. When a zone is specified by the user, parts of the global space that intersect the zone are retrieved from *clQuad-tree* together with their co-location patterns and instances. The co-location patterns can then be discovered at a lower cost.

Another kind of extension is integrating co-location mining with temporal data. In this kind of application, the input is a spatial space and several snapshots of the spatial space at different times. Each snapshot is a map of the spatial space with spatial instances on it at a certain moment. As time goes by, the spatial instances may move or disappear,

and new spatial instances may emerge. Based on this kind of input, in [25], the authors propose co-evolving spatial events mining. Such mining allows users to specify the evolvement of co-location prevalence as a sequence of prevalence, and discovers co-location patterns that evolve similarly with the specified evolvement. A function calculating the similarity between two evolvements is used. A naïve mining approach is mining the complete set of co-location patterns in each snapshot, discovering the evolvement for each co-location pattern, and filtering out the co-location patterns whose evolvements satisfy the user's requirements. However, the authors discovered some anti-monotone properties of the co-location patterns and their evolvements. They proposed an algorithm that avoids unnecessary co-location patterns mining and thus achieves a better performance.

Given a set of snapshots of the spatial space, a new time prevalence measure representing how often a co-location pattern appears in these snapshots is introduced in [5]. A high time prevalence indicates that the co-location pattern does not appear coincidentally, but consistently. The time prevalence is measured as the fraction of the number of snapshots where the co-location pattern is found prevalent to the total number of snapshots. This kind of co-location patterns is called mixed-drove spatio-temporal co-occurrence patterns. As in the co-evolve pattern mining discussed previously, there also exists a naïve but unnecessarily expensive approach for mixed-drove co-occurrence patterns mining. The authors propose a more efficient algorithm based on the observation that if a co-location pattern is not time prevalent, its supersets will not be time prevalent [5].

Although these extensions have their own distinct applications and requirements, they are all based on traditional co-location mining and require quick clique recognition. Therefore, the improvement of the performance of traditional co-location mining will also improve the performance of these new extensions.

CHAPTER 3

PRELIMINARIES

A spatial dataset has the following three properties:

1. A set of spatial feature types $F = \{F_1, \dots, F_n\}$, e.g., “park,” “school,” “zoo” and “library.”
2. A set of feature instances $I = \{I_1, \dots, I_m\}$ s such that $I_k \in I$ ($1 \leq k \leq m$) is an instance of type $F_l \in F$, ($1 \leq l \leq n$). For example, “Zion National Park” is an instance of the type “park,” and “Logan High” is an instance of the type “school.”
3. A set of neighborhood relationships R whose elements are an unordered pair of instances in I of different types, such that the two instances are neighbors to each other. For example, (“Logan City Library,” “Logan High”) denotes that the two instances are neighbors. The neighborhood relationships are usually defined according to Euclidean distances. However, since the neighborhood relationships are given as input, the user can define other kinds of neighborhood relationships according to the specific application, thus providing more flexibility to users.

As an example, consider Figure 3 wherein each node represents an instance, and each edge denotes the neighborhood relationship between the two corresponding instances at both ends. Hence, we identify a set of types $F = \{A, B, C, D\}$, a set of instances $I = \{A_1, A_2, A_3, A_4, A_5, B_1, B_2, B_3, B_4, C_1, C_2, C_3, C_4, D_1, D_2\}$ in the figure.

Figure 3. Spatial types A , B , C , D and their instances.

Next, are the definitions of the concepts used in this thesis, which commonly appear in the spatial co-location literature. Let $f(I_k)$ be a function that returns type of instance I_k .

Definition 1 (Co-location Instance List). Given a set of types F , a set of instances I defined over F , and a set of neighborhood relationships R over I , a list of instances $\langle I_1, \dots, I_m \rangle$ such that $I_k \in I$, ($1 \leq k \leq m$ and $m \geq 1$) is called co-location instance list if

1. Any pair of instances in $\langle I_1, \dots, I_m \rangle$ is an element of R , i.e., $\langle I_1, \dots, I_m \rangle$ is a clique, and
2. $f(I_i) < f(I_j)$ holds for every pair of instances I_i, I_j where $1 \leq i < j \leq m$. (in other words, instances are lexicographically ordered according to their type.)

□

Note that any sublist l of a co-location instance list is also a co-location instance list as long as $|l| \geq 2$.

Example 1. Consider the instances and the neighborhood relationships in Figure 3. $\langle A_4, B_2, C_3 \rangle$ and $\langle A_4, B_2, C_1 \rangle$ are co-location instance lists of types $\langle A, B, C \rangle$, where $\langle A_5, B_4, C_3 \rangle$ of the same types is not a co-location instance list since B_4 and C_3 are not neighbors. In other words, $\langle A_4, B_2, C_3 \rangle$ and $\langle A_4, B_2, C_1 \rangle$ are cliques, whereas $\langle A_5, B_4, C_3 \rangle$ is not. $\square$

A set of co-location instance lists is called a co-location list set when every instance list in the set has the same types. Such a group of types is called co-location type list. For instance, $\langle A_4, B_2, C_3 \rangle$ and $\langle A_4, B_2, C_1 \rangle$ in Example 1 form a set of co-location instance lists $\{\langle A_4, B_2, C_3 \rangle, \langle A_4, B_2, C_1 \rangle\}$ whose co-location type list is $\langle A, B, C \rangle$.

Notice that in co-location mining, we are not interested in the co-location patterns involving the same spatial types. For example, we are not interested in a co-location pattern such as “schools are frequently co-located with schools.” However, if so desired, we can easily change the “school” feature type into several more detailed feature types, like “elementary school,” “middle school,” “high school,” and “university,” Co-location patterns can then be mined between these new feature types.

Definition 2. (Participation Ratio). Given a type set F and a co-location type list $\mathcal{C}$ defined over F , let CL denote a co-location list set complying with $\mathcal{C}$. Then, the participation ratio of $\mathcal{C}$ conditioned on type $F_i \in \mathcal{C}$ is defined as

$$\Pr(\mathcal{C}|F_i) = \frac{\text{number of distinct instances of } F_i \text{ in } CL}{\text{number of instances of } F_i \text{ globally}} \square$$

The participation ratio of a co-location type list $\mathcal{C}$, $\Pr(\mathcal{C})$ is defined as the minimal conditional participation ratio of $\mathcal{C}$, i.e., $\Pr(\mathcal{C}) = \min (\Pr(\mathcal{C}|F_i))$.

Example 2. Consider a co-location instance list set $\{\langle A_4, B_2, C_3 \rangle, \langle A_4, B_2, C_1 \rangle\}$ and its corresponding co-location type list $\mathcal{C} = \langle A, B, C \rangle$ in Example 1. From this, we obtain $\Pr(\mathcal{C}|A) = \frac{1}{5} = 0.2$, $\Pr(\mathcal{C}|B) = \frac{1}{4} = 0.25$, $\Pr(\mathcal{C}|C) = \frac{2}{4} = 0.5$ which yields 0.2 as the participation ratio of $\mathcal{C}$. $\square$

The participation ratio of a co-location type list conditioned on a spatial feature type indicates how strongly the spatial feature type is related to the co-location type list. For example, if there exists a co-location type list $\mathcal{C} = \langle F_1, F_2, F_3 \rangle$ and $\Pr(\mathcal{C}|F_2) = 90\%$, 90% of F_2 's instances will participate in the co-location instance lists of $\mathcal{C}$, which also means that if we find an instance I_2 of F_2 , then the possibility that we will find instances of F_1 and instances of F_3 in I_2 's neighborhood is 90%. The participation ratio of a co-location type list indicates how strongly the spatial feature types within the co-location type list cohere together.

Using the definition of participation ratio, we say that a co-location type list $\mathcal{C}$ is frequent if $\Pr(\mathcal{C})$ is greater than or equal to the given threshold. If the threshold is 0.2, the co-location type list $\langle A, B, C \rangle$ in Example 2 is frequent.

Lemma 1 (Anti-Monotone). Given two co-location type lists $C_1 = \langle F_1, \dots, F_m \rangle$ and $C_2 = \langle F_1, \dots, F_n \rangle$, if $C_1 \subseteq C_2$, then $\Pr(C_1) \geq \Pr(C_2)$.

Proof. Assuming that $F_i \in C_1$ and $F_i \in C_2$, then $\Pr(C_1|F_i) \geq \Pr(C_2|F_i)$, because for any instance I_i of F_i , if I_i participates C_2 , then it should also participate C_1 . However, if an instance I_i of F_i participates C_1 , it may or may not participate C_2 . So, $\Pr(C_1|F_i) \geq \Pr(C_2|F_i)$.

Assuming that $Pr(C_1) = Pr(C_1|F_i)$ and $Pr(C_2) = Pr(C_2|F_j)$, then $Pr(C_1) \geq Pr(C_2)$, because $Pr(C_1|F_i) \geq Pr(C_2|F_i) \geq Pr(C_2|F_j)$ according to the definition of the participation ratio. $\square$

Lemma 1 indicates that the participation ratio is monotonically non-increasing according to the increasing sizes of co-location type lists. If a co-location type list C_1 's participation ratio is smaller than the threshold given by the user, then for any co-location type list C_2 , where $C_1 \subset C_2$, C_2 's participation ratio is smaller than the threshold, too. Using this feature, the pruning of co-location type lists can be applied to the co-location mining process.

Definition 3 (Co-location Rule). Given a type set F , and a set of co-location type lists $\mathcal{CT}$, defined over F , a co-location rule is a rule of the form $C_1 \rightarrow C_2$ such that $C_1, C_2 \in \mathcal{CT}$ and $C_1 \cap C_2 = \emptyset$. $\square$

For example, $\langle A, C \rangle \rightarrow \langle D \rangle$ is a co-location rule shown in Figure 3.

The confidence level of a co-location rule $R = C_1 \rightarrow C_2$ is defined in a way similar to the traditional confidence, i.e.,

$$Pr(R) = \frac{\text{number of instances of } C_1 \in (C_1 \cup C_2)}{\text{number of instances of } C_1}$$

CHAPTER 4

PREVIOUS ALGORITHMS

4.1 Spatial Association Rules Mining

The problem of spatial association rules mining was first discussed in [10], and the authors propose a reference feature centric model [18] to discover spatial association rules.

In this model, a spatial association rule is a rule of the form:

$$C_1 \wedge \dots \wedge C_m \rightarrow D_1 \wedge \dots \wedge D_n (s\%, c\%)$$

where C_i ($1 \leq i \leq m$) and D_i ($1 \leq i \leq n$) are predicates and at least one of these predicates should be spatial predicates. Spatial predicates can involve any spatial relationship supported by the spatial databases. The support of the rule, denoted as $s\%$, is defined as $\frac{\text{the number of objects who satisfy } C_1 \wedge \dots \wedge C_m \wedge D_1 \wedge \dots \wedge D_n}{\text{the number of all objects in the dataset}}$. The confidence of rule, denoted as $c\%$, is defined as $\frac{\text{the number of objects which satisfy } C_1 \wedge \dots \wedge C_m \wedge D_1 \wedge \dots \wedge D_n}{\text{the number of objects which satisfy } C_1 \wedge \dots \wedge C_m}$. A high support indicates that the objects occur frequently, and a high confidence indicates strong implication. In spatial association rules mining, the user provides both a threshold of the support and a threshold of the confidence. A rule with support and confidence higher than the corresponding thresholds is considered as a strong rule, and only strong rules are returned to the user.

An example of spatial association rules is:

$$is_house(X) \wedge close_to(X, Oswego Lake) \rightarrow is_expensive(X)$$

This example rule can be interpreted as: if a house is close to Oswego Lake, then it is expensive. In this example, the two predicates $is_house(X)$ and $is_expensive(X)$ are non-spatial predicates, while $close_to(X, Oswego Lake)$ is a spatial predicate. Moreover, this example implies a reference feature, which is house.

In [10], an SQL-like spatial data mining query language is adopted. Suppose that a user wants to discover the association rules among houses in the state of Oregon and their physical proximity to roads, rivers, mountains, parks, and byways. Such a query can be written as:

Discover spatial association rules
inside Oregon
from road R, river L, mountain M, park P, byway B
in relevance to house H
where close_to(H.geom, X.geom) and $X \in \{R, L, M, P, B\}$

Here, $close_to(X, Y)$ is a function taking two spatial objects as arguments and returning *TRUE* if the two spatial objects are close to each other. *geom* is a field storing detailed geometrical data of spatial objects.

The mining of spatial association rules requires five inputs: 1) a spatial database, 2) a reference class (“house” in the above example), 3) a set of classes whose relationships with the reference class are interesting to users (“road,” “river,” “mountain,” “park,” and “byway” in the above example), 4) a spatial relationship type (“close_to” relationship between the reference class and other classes in the above example), 5) a minimum support threshold and a minimum confidence threshold.

The mining of spatial association rules takes the following steps:

1. Extract spatial objects from the spatial database.
2. For each object of the reference class, discover its specified spatial relationship with spatial objects of other classes.
3. Mine spatial association rules.

Using the above example, at Step 1, all houses, roads, rivers, mountains, parks, and byways within Oregon are extracted. This can be done using queries on the spatial database.

At Step 2, for each house extracted in the previous step, find all roads, rivers, mountains, parks and byways that are close to it. This step can create a bottle-neck in performance, because the calculation of spatial relationships between spatial objects with irregular shapes is very expensive. Since only predicates whose supports are higher than the minimum support threshold are needed in the next step, a filter-and-refine approach is adopted to achieve the best performance. This approach takes three steps. On the first step, each spatial object is represented using its minimum bounding rectangle, and the spatial relationships between spatial objects are coarsely calculated. On the second step, the support of each spatial predicate is calculated. If the support is smaller than the minimum support threshold, it is pruned. On the last step, a refined spatial relationship calculation is performed on each remaining predicate, and predicates that do not satisfy the refined spatial relationship requirement are pruned. The supports of predicates are recalculated, and predicates whose supports are smaller than the threshold are pruned again. The reason for adopting the filter-and-refine approach is that the cost of calculating

spatial relationships between spatial objects with rectangular shapes is far less expensive than the cost of directly calculating spatial relationships between spatial objects with irregular shapes. If the distance between two spatial objects' bounding boxes is greater than the distance threshold, they are definitely not neighbors, and the expensive exact calculation can be avoided. The coarse level pruning using supports of predicates further reduces the number of exact calculations.

A new database table is generated after the steps discussed above. In this table, each row represents a house and contains a cell that is a set of spatial objects close to the house. Each row of the table can be considered as a transaction. Further, traditional association rules mining algorithms, which have been well developed, can be applied to discover the association rules. The only thing we need to take additional care of is that each transaction implies a house. So, for each association rule $C \rightarrow D$ mined, we need to add an additional $is_house(X)$ predicate and get $is_house(X) \wedge C \rightarrow D$.

In the real world, spatial classes usually have refined subclasses. For example, the “road” class can be refined to “highway,” “interstate,” “street,” etc. Spatial relationships may also be refined. For example, a “close_to” relationship can be refined into “intersects,” “touches,” “within,” etc. Given concept hierarchies on spatial classes and/or spatial relationships, multiple-level association rules mining can also be applied.

As discussed in Chapter 2, the reference feature centric model has a major limitation in that users are required to provide a reference feature, which would be difficult in some applications. For this reason, event centric co-location mining was proposed.

4.2 Join-Based Co-location Mining

To overcome the drawbacks of spatial association rules mining as mentioned previously, an event centric co-location mining model was proposed in [17]. The problem definition of this model is the same as defined in Chapter 3.

Since the event centric model does not require a reference class, spatial classes in a spatial dataset are equal. Using this model, we are not able to transactionize spatial datasets in the same way as the reference centric model allows. Actually, because the spatial spaces, wherein spatial objects are distributed, are continuous and there is no natural way to partition them, transactionizing the spatial datasets and applying the association rules mining algorithms are not trivial matters.

In [17], the authors adopt an Apriori-like join-and-test approach. This approach starts with length-1 spatial co-location type lists (which are spatial features) and their instances, and the mining moves iteratively from length- k ($k \geq 1$) to length- $(k + 1)$. On each iteration, the mining involves following four steps:

1. Generate candidate length- $(k + 1)$ co-location type lists from frequent length- k co-location type lists. This is achieved by joining any two frequent length- k co-location type lists that share the same first $k - 1$ spatial features but have different k^{th} spatial features. Any length- k sublist of the generated candidate co-location type list is checked if it exists in the frequent co-location type lists set because according to Lemma 1, if a length- k co-location type list is not frequent, then all its supersets will not be frequent.

2. For each candidate length- $(k + 1)$ co-location type lists generated in previous step, generate all co-location instance lists.
3. Calculate the participation ratios for each candidate co-location type list, and prune the candidate co-location type list if its participation ratio is smaller than a user-defined participation ratio threshold.
4. For each remaining co-location type list, generate co-location rules, and prune co-location rules whose confidences are smaller than a user-defined confidence threshold.

Step 1, Step 3, and Step 4 are straightforward and do not involve any spatial operations. However, Step 2 is critical and usually would be the bottle-neck of the performance, because the number of spatial features might be small in real applications, but for each spatial feature, the number of instances is usually big.

The authors of [17] adopt a join-based approach to generate co-location instance lists. Assuming that for any co-location instance list s , $s.I_i$ denotes the i^{th} spatial instance of s . Assuming that there are two prevalent length- k co-location type lists c_1 and c_2 , and we can get a length- $(k + 1)$ candidate co-location type list c by joining c_1 and c_2 , the co-location instance lists set of c_1 is S_1 and the co-location instance lists set of c_2 is S_2 . The co-location instance list set of c is generated by joining any s_1 and s_2 , where $s_1 \in S_1$, $s_2 \in S_2$, $s_1.I_1 = s_2.I_1$, $s_1.I_2 = s_2.I_2$, ..., $s_1.I_{k-1} = s_2.I_{k-1}$, $s_1.I_k$ and $s_2.I_k$ are neighbors. As we can see, to generate co-location instance lists of c , we need to test $|S_1| \times |S_2|$ co-location instance lists joins in the worst case scenario.

4.3 Partial-Join-Based Co-location Mining

To reduce the number of joins required during the co-location mining process, a partial-join based algorithm is proposed in [23]. This algorithm specifically assumes that neighborhood relationships are based on Euclidean distances. The basic idea of the algorithm is that partitioning the spatial space in a way that for any two spatial instances within the same partition, their Euclidean distances are smaller than or equal to the maximum distance threshold, which defines the neighborhood relationship, all spatial instances within the same partition are automatically a clique. The generation of co-location instance lists for a co-location type list using spatial instances within the same partition is to simply discover the correct subsets of the instances, and save the efforts on joins. This kind of co-location instance list is called an intraX co-location instance list.

Figure 4 shows an example of space partitioning that is generated by partitioning the original space, as shown in Figure 3, into 15 squares. The boundaries of the partitions are represented using dashed lines. Assuming that the maximum distance threshold which determines neighborhood relationships is d , then each partition in Figure 4 is a $\frac{d}{\sqrt{2}} \times \frac{d}{\sqrt{2}}$ square, and the maximum distance between any two spatial instances within this partition is d . This guarantees that all spatial instances within the same partition are neighbors to each other, thus forming a clique.

Partitioning into squares is not the only way to partition a spatial space. We can also partition the spatial space into circles with $\frac{d}{2}$ radius. The only requirement of partitioning is that the maximum distance between instances within the same partition is not greater than the distance threshold.

Figure 4. Grid partitioning of the spatial space presented in Figure 3.

Although the basic idea of this algorithm is easy to understand, two problems remain: 1) There are different ways to partition a spatial space, which way will optimize the overall performance? 2) There might be some co-location instance lists involving spatial instances from two or more partitions. How can we generate these co-location instance lists?

The second problem is discussed first. Let's assume that the spatial space has already been partitioned into a set of disjoint partitions, and each spatial instance belongs to one and only one partition.

The co-location instance lists involving spatial instances from two or more partitions must include at least one pair of instances who are neighbors to each other and reside in different partitions. The neighborhood relationship of these two instances cuts the boundary of two partitions. This kind of neighborhood relationship is called a cut neighborhood. For a co-location instance list $CL = \langle I_1, I_2, \dots, I_n \rangle$, whose instances reside in partitions $P_1, \dots, P_m$, any spatial instance $I_i \in CL$, I_i must have cut neighborhoods with

some instances in CL , but reside in partitions other than where I_i resides. A co-location instance list, whose instances all have at least one cut neighborhood, is called an interX co-location instance list.

When we are mining length- k ($k \geq 2$) co-location type lists, each length- k co-location type list maintains a set of interX co-location instance lists. The set of interX co-location instance lists is usually smaller than the complete set of co-location instance lists. When we move to length- $(k + 1)$ co-location type lists mining, joins only need to be applied among length- k interX co-location instance lists. This approach saves a lot of time, compared to the joins applied between complete sets of co-location instance lists.

In [24], the authors add a coarse filtering step to the partial-join algorithm. The following is based on the assumption that we are generating co-location instance lists for a length- $(k + 1)$ co-location type list $CL = \langle F_1, \dots, F_k, F_{k+1} \rangle$. After generating intraX co-location instance lists, spatial instances participating in the interX co-location instance lists of length- k sublist of CL are gathered. We assume that these spatial instances also participate in the co-location instance lists of CL and calculate the coarse participation ratio of CL using the intraX co-location instance lists and these spatial instances. The coarse participation ratio is greater than or equal to the real participation ratio, and if it is smaller than the participation ratio threshold, CL is pruned without generating its interX co-location instance lists.

Now, let us consider the partitioning strategies. From previous discussion, we know that the partial-join algorithm achieves better performance by reducing the join operations between co-location instance lists. The ideal partitioning strategy clusters

spatial instances perfectly and minimizes the number of cut neighborhoods between spatial instances. Figure 4 adopts a simple partitioning strategy. However, from the figure, we can see that this strategy partitions the instances of the clique $\langle A_4, B_2, C_3, D_1 \rangle$ into four different partitions, and results in six cut neighborhoods between them. A better partitioning strategy is presented in Figure 5.

Unfortunately, there is no single superior partitioning strategy available. The best strategy for one dataset might be the worst for another dataset. In fact, [23] does not propose any partitioning strategy, but rather, suggests a few strategies, including grids [16] and maximal cliques [3], etc. Moreover, join operations are unavoidable in most cases. These drawbacks motivate researchers to find other solutions.

4.4 Joinless Co-location Mining

To further improve the performance of co-location instance lists generation, in [24], the authors propose a co-location mining algorithm that does not rely on join

Figure 5. Another partitioning strategy for the spatial space presented in Figure 3.

operations at all.

The joinless co-location mining algorithm is built on top of a novel data structure called star neighborhood, which is used to materialize neighborhood relationships and replace the original neighborhood relationships set R . In this algorithm, each spatial instance is associated with a star neighborhood. The star neighborhood of a spatial instance I_i is defined as:

$$SN(I_i) = \{I_j | i = j \vee (R(I_i, I_j) \wedge f(I_j) > f(I_i))\}$$

Here, R represents the set of neighborhood relationships, and $R(I_i, I_j)$ means I_i and I_j are neighbors. $f(I_k)$ is a function that returns a spatial instance, I_k 's feature type. The joinless algorithm lexicographically orders spatial feature types, and the star neighborhood of a spatial instance I_i includes I_i and all I_i 's neighbors whose spatial feature types are lexicographically greater than I_i 's. The star neighborhood is then a star-like data structure with a center instance and its neighbors surrounding it. The star neighborhood of I_i does not include I_i 's neighbors whose spatial feature types are lexicographically smaller than I_i 's, to avoid duplications. [24] proves that this process does not lose any neighborhood relationships.

The definition of the star neighborhood implies that if there is a co-location type list $\mathcal{C} = \langle F_1, F_2, \dots, F_k \rangle$ (assuming that spatial feature types in $\mathcal{C}$ are lexicographically ordered) and one of its co-location instance lists $\langle I_1, I_2, \dots, I_k \rangle$, then $\langle I_1, I_2, \dots, I_k \rangle \subseteq SN(I_1)$.

Based on the star neighborhood, another data structure, called star instance, is proposed. For a co-location type list $\mathcal{C} = \langle F_1, F_2, \dots, F_k \rangle$, a list of spatial instances

$SI = \langle I_1, I_2, \dots, I_k \rangle$ is a star instance of $\mathcal{C}$, if $f(I_1) = F_1$, $SI \subseteq SN(I_1)$, and for any $I_i \in SI$, $f(I_i) \in \mathcal{C}$ holds. Given a co-location type list $\mathcal{C} = \langle F_1, F_2, \dots, F_k \rangle$, its star instances are generated by going through the instances of F_1 , and finding correct subsets of their star neighborhoods.

Notice that a star instance is not guaranteed to be a co-location instance list, because it is not guaranteed to be a clique. For a co-location type list, the set of its star instances is a superset or an equal set of the set of its co-location instance lists. After generating a star instance, the cliqueness of this start instance is checked and is pruned if it is not a clique.

The cliqueness checking of a star instance is done using an instance lookup approach. For a co-location type list $\mathcal{C} = \langle F_1, F_2, \dots, F_k \rangle$ and one of its star instances $SI = \langle I_1, I_2, \dots, I_k \rangle$, if $CI = \langle I_2, \dots, I_k \rangle$ is a co-location instance list of $\langle F_2, \dots, F_k \rangle$, then SI is a clique. The proof of this statement follows. For any $I_i \in SI$, $I_j \in SI$, and $I_i \neq I_j$, if $I_i \in CI$ and $I_j \in CI$, then I_i and I_j are neighbors according to the definition of co-location instance list; otherwise, either $I_i = I_1$ or $I_j = I_1$; thus, I_i and I_j are neighbors because $SI \subseteq SN(I_1)$ and any instance in $SN(I_1)$ is a neighbor of I_1 according to the definition of star neighborhood. Based on the above discussion, the cliqueness checking of SI can be done by checking the existence of CI in the co-location instance lists set of $\langle F_2, \dots, F_k \rangle$.

Based on the star neighborhood and star instance, the mining of length- $(k + 1)$ co-location type lists from frequent length- k type lists takes four steps:

1. Generate candidate length- $(k + 1)$ co-location type lists from frequent length- k co-location type lists. This is the same as the first step of the join-based algorithm.
2. For each candidate length- $(k + 1)$ co-location type list, generate star instances in the way discussed above.
3. For each star instance generated, check its cliqueness, and prune it if it is not a clique. This step is not required when mining length-2 co-location type lists, because the star instances generated at this iteration are elements of the neighborhood relationship set R , which are guaranteed to be cliques.
4. Calculate the participation ratio for each candidate co-location type list using the remaining star instances, and prune the candidate co-location type lists whose participation ratios are smaller than the participation ratio threshold, along with their star instances.

Since the set of star instances of a co-location type list is the superset or equal set of its co-location instance lists set, if the participation ratio of the co-location type list calculated using the star instances is smaller than the participation ratio threshold, then the real participation ratio calculated from its co-location instance lists set are smaller than the participation ratio threshold, too. Based on this observation, a coarse level pruning step is added after the generation of star instances. This coarse level pruning step assumes that all star instances are cliques and calculates the participation ratios of candidate co-location type lists using the star instances. If a co-location type list's participation ratio calculated using its star instances is smaller than the participation ratio

threshold, then this co-location type list, along with its star instances, are pruned, and the computation on the cliqueness checking of its star instances is saved; otherwise, the cliqueness checking is performed on each of its star instances, and a refined participation ratio is calculated. The co-location type list is pruned if its refined participation ratio is smaller than the participation ratio threshold.

The complete joinless algorithm is shown below:

Inputs:

F : a set of types;

I : a set of instances defined over F ;

R : a set of neighborhood relationships between the instances in I ;

p : minimum participation ratio threshold;

c : minimum confidence threshold;

Output:

CR : a set of co-location rules whose confidence is larger than or equal to c .

Variables:

k : length of the current co-location type lists;

T_k : set of co-location type lists of length- k ;

L_k : set of co-location instance lists of length- k ;

SI_k : set of star instances of length- k ;

SN : set of star neighborhoods;

Algorithm:

$SN := generate_star_neighborhoods(F, I, R);$

```

 $T_1 := F;$ 

 $L_1 := I;$ 

 $k := 2;$ 

while  $k \leq |F|$  and  $T_{k-1} \neq \emptyset$ 

     $T_k := \text{generate\_candidate\_co - location\_type\_lists}(T_{k-1});$ 

    for each  $t \in T_k$ 

         $SI_k \cup = \text{generate\_star\_instances}(t, F, I, SN);$ 

    end for

    if  $k \neq 2$ 

         $T_k, SI_k := \text{filter\_infrequent\_co - location\_}$ 
            $\text{type\_lists}(T_k, SI_k, p);$ 

         $L_k := \text{cliqueness\_checking}(SI_k, L_{k-1});$ 

    end if

     $T_k, L_k := \text{filter\_infrequent\_co - location\_}$ 
        $\text{type\_lists}(T_k, L_k, p);$ 

end while

 $CR := \text{generate\_co - location\_rules}(c, F \cup T_2 \cup \dots, I \cup L_2 \cup \dots);$ 

```

At the end of [24], the authors analyze and compare the computation complexities of the join-based algorithm, the partial-join based algorithm, and the joinless algorithm. The authors demonstrate the joinless algorithm's advantages in performance. The authors also conducted experiments comparing the performance of three algorithms on synthetic data sets and real data sets. The results show that the joinless algorithm performed better

than the other two algorithms in most cases. However, in data sets where instances were likely to be clustered, the partial-join algorithm performed slightly better than the joinless algorithm, while in sparse data sets, the join-based algorithm proved to be more appropriate than the joinless algorithm.

CHAPTER 5

NEIGHBOR CLUSTER ALGORITHM

The design goal of the proposed co-location mining algorithm is better efficiency than the existing algorithms by avoiding the generation of nonclique instances and join operations, and subsequently not requiring extra cliqueness checking for every instance at every iteration level. This goal is achieved by introducing 1) the novel neighbor cluster (NC) data structure, and 2) a linear intersection operation over NC.

5.1 Neighbor Cluster

The neighbor cluster data structure is to organize neighborhood relationships between spatial feature instances while preserving cliqueness among them. In this data structure, instances are ordered according to the lexicographical order of their feature types, whereas the input neighborhood relationships are unordered.

Given a set of feature instances I and neighborhood relationship set R defined over I , the neighbor cluster (NC) of an instance $I_k \in I$ whose feature type is $f(I_k)$ is defined as:

$$NC(I_k) = \{I_l | I_l \in I \wedge (I_k, I_l) \in R \wedge f(I_k) < f(I_l)\}$$

In other words, the neighbor cluster of an instance I_k is excluded from encoding the neighborhood relationship between I_k and any instance whose feature type is lexicographically smaller than $f(I_k)$. However, the same neighborhood relationship is encoded in the NC of any instance I_l whose feature type is lexicographically smaller than

$f(I_k)$, which satisfies the symmetry property of neighborhood relationships discussed in Chapter 3. Hence, we can see that NC is lossless with respect to R .

Notice that the definition of a neighbor cluster of instance I_k looks similar to the definition of the star neighborhood of I_k . However, the star neighborhood of I_k includes the instance I_k itself, which the neighbor cluster of I_k does not.

For the instances presented in Figure 3 and the neighborhood relationships depicted, the NC for each instance is built and shown in Table 2.

Next, let's extend the notion of NC to be applicable to co-location instance lists, in addition to merely instances. The neighbor cluster of a co-location instance list is defined as follows:

Definition 4 (Neighbor Cluster). Given a set of neighbor clusters $\{NC(I_1), NC(I_2), \dots\}$ defined over a set of instances $I = \{I_1, I_2, \dots\}$, let $\langle I_j, I_{j+1}, \dots, I_k \rangle$ be a co-location instance list. Then, the neighbor cluster (NC) of the co-location instance list is recursively defined as:

$$NC(\langle I_j, I_{j+1} \rangle) = NC(I_j) \cap NC(I_{j+1})$$

$$NC(\langle I_j, \dots, I_k \rangle) = NC(\langle I_j, \dots, I_{k-1} \rangle) \cap NC(I_k) \quad \square$$

In other words, $NC(\langle I_j, \dots, I_k \rangle) = NC(I_j) \cap NC(I_{j+1}) \cap \dots \cap NC(I_k)$. Let us take an example of NC generation.

Example 3. Consider the set of the NCs of feature instances shown in Table 2, and suppose that we are generating $NC(\langle A_1, C_4 \rangle)$. This is done by $NC(A_1) \cap NC(C_4)$ which yields $\{C_4, D_2\} \cap \{D_2\} = \{D_2\}$. It is easy to see that $NC(\langle A_1, D_2 \rangle) = NC(A_1) \cap$

$NC(D_2) = \{C_4, D_2\} \cap \emptyset = \emptyset$. Furthermore, $NC(\langle A_1, C_4, D_2 \rangle)$ is computed by

$NC(\langle A_1, C_4 \rangle) \cap NC(D_2)$ which yields $\emptyset$. $\square$

Table 2. Neighbor Clusters (NCs) of the Spatial Feature Instances Shown in Figure 3.

Type	Instance (I_k)	$NC(I_k)$
A	A_1	$\{C_4, D_2\}$
	A_2	$\emptyset$
	A_3	$\{B_3, B_4\}$
	A_4	$\{B_2, C_1, C_3, D_1\}$
	A_5	$\{B_4, C_3\}$
B	B_1	$\emptyset$
	B_2	$\{C_1, C_3, D_1\}$
	B_3	$\emptyset$
	B_4	$\emptyset$
C	C_1	$\emptyset$
	C_2	$\emptyset$
	C_3	$\{D_1\}$
	C_4	$\{D_2\}$
D	D_1	$\emptyset$
	D_2	$\emptyset$

Lemma 2 (Completeness). Given a co-location instance list C_k and $NC(C_k)$, if an instance I_j , such that $I_j \notin C_k$, can be appended to C_k to yield a new co-location instance list C_{k+1} , then $I_j \in NC(C_k)$ holds. $\square$

Proof. If I_j can be integrated into co-location instance list C_k to form a length- $(k+1)$ co-location instance list $C_{k+1} = \langle I_1, \dots, I_k, I_j \rangle$, then I_j has the property that $\forall I_i \in C_k$, $f(I_i) < f(I_j)$, and $(I_i, I_j) \in R$ by the definition of co-location instance list in Definition 1. Subsequently, $I_j \in NC(I_i)$ by the definition of neighbor cluster. Since $I_j \in NC(I_1), \dots, I_j \in NC(I_k)$, then $I_j \in NC(I_1) \cap \dots \cap NC(I_k)$ must hold. In fact, $NC(I_1) \cap \dots \cap NC(I_k) = NC(C_k)$ by Definition 4. In other words, for any instance I_j which is appendable to C_k , $I_j \in NC(C_k)$. $\square$

Lemma 2 implies that we will not miss any qualified co-location instance list by using NC.

5.2 Level-wise Generation of Co-location Type Lists

We now discuss how co-location type lists are generated during the generation of neighbor clusters in the NCA algorithm.

In general, given a set of instances $I = \{I_1, \dots, I_m\}$ and a set of types $F = \{F_1, \dots, F_n\}$, the general steps of length- $(k+1)$ ($k \geq 1$) co-location type lists generation are:

1. *Generation of candidate type lists.* Generate all candidate length- $(k+1)$ type lists from frequent length- k co-location type lists. This is done by joining any two length- k frequent co-location type lists with the same length- $(k-1)$

prefix and different k^{th} type. Check if any of its length- k sublists is frequent or not. This step is the same as the first step of the join-based algorithm.

2. *Generation of co-location instance lists and neighbor clusters.* For any length- $(k + 1)$ candidate co-location type list $\langle F_1, \dots, F_k, F_{k+1} \rangle$, select each co-location instance list $\langle I_1, \dots, I_k \rangle$ of $\langle F_1, \dots, F_k \rangle$ one at a time.
 - a. *Co-location instance list.* Generate a new co-location instance list $\langle I_1, \dots, I_k, e \rangle$ by appending each element e in $NC(\langle I_1, \dots, I_k \rangle)$ to $\langle I_1, \dots, I_k \rangle$ one at a time, where e is of the type F_{k+1} .
 - b. *Neighbor clusters.* Generate $NC(\langle I_1, \dots, I_k, e \rangle)$ using the equation in Definition 4.
3. *Generation of frequent co-location type lists.* For each candidate co-location type list of length- $(k + 1)$, determine if it is frequent by examining its participation ratio. Any candidate co-location type list whose participation ratio is less than the given threshold is pruned together with its associated co-location instance lists.

Notice that, when $k=1$, the set of candidate length-1 co-location type lists is the set of feature types F , and the set of length-1 co-location instance lists is the set of the instances given in I .

Upon the completion of co-location type lists and co-location instance lists generation, the participation ratio of each co-location type list is calculated prior to the next iteration. Any candidate co-location type list whose participation ratio is less than the given threshold is pruned together with its associated co-location instance lists.

Example 4. For the example in Figure 3, a complete list of mining result is presented in Table 3, assuming that the participation ratio threshold is 0.0. Below, let's see how $\langle A, B, C, D \rangle$ and its instance $\langle A_4, B_2, C_3, D_1 \rangle$ are generated. Let's begin with the feature A from which candidate co-location type lists $\langle A, B \rangle$, $\langle A, C \rangle$ and $\langle A, D \rangle$ are generated. Only instance A_4 is considered since none of A_1, A_2, A_3 or A_5 contributes to the co-location instance list of $\langle A, B, C, D \rangle$. A_4 is integrated with each element in $NC(A_4)$ one at a time to obtain new co-location instance lists $\langle A_4, B_2 \rangle$ for $\langle A, B \rangle$, $\langle A_4, C_1 \rangle$ and $\langle A_4, C_3 \rangle$ for $\langle A, C \rangle$, and $\langle A_4, D_1 \rangle$ for $\langle A, D \rangle$. Their NCs are also generated and shown in Table 2. Afterwards, the frequentness of each candidate co-location type list is determined. Then, let's move on to the next level to generate candidate co-location type lists of length-3, e.g., $\langle A, B, C \rangle$ by joining $\langle A, B \rangle$ and $\langle A, C \rangle$. In Table 3, note that co-location instance lists are ordered according to their types for more readability and do not reflect the actual sequence of generation at each level.

Next, the first two rows of Level 3 in Table 3 are generated from $NC(\langle A_4, B_2 \rangle) = \{C_1, C_3, D_1\}$. In the last iteration, $\langle A, B, C \rangle$ and $\langle A, B, D \rangle$ are joined, and the candidate co-location type list $\langle A, B, C, D \rangle$ is generated. The co-location instance list $\langle A_4, B_2, C_3, D_1 \rangle$ of $\langle A, B, C, D \rangle$ is extended from $\langle A_4, B_2, C_3 \rangle$ in a similar way. $\square$

In the following lemma, another property of the NC data structure is presented. This property is the key to the scalability benefits of the NCA algorithm.

Lemma 3 (Cliquesness preserving). Given a co-location instance list C_k and its neighbor cluster $NC(C_k)$, any C_{k+1} generated by appending $I_j \in NC(C_k)$ to C_k guarantees to be a co-location instance list. $\square$

Table 3. Level-wise Generation of Candidate Co-location Type List $\langle A, B, C, D \rangle$ from the Neighbor Clusters Shown in Table 2.

Level	Candidate co-location type list	Co-location instance list	Neighbor Clusters	Participation ratio
2	$\langle A, B \rangle$	$\langle A_3, B_3 \rangle$	$\emptyset$	$\frac{3}{5}$
		$\langle A_3, B_4 \rangle$	$\emptyset$	
		$\langle A_4, B_2 \rangle$	$\{C_1, C_3, D_1\}$	
		$\langle A_5, B_4 \rangle$	$\emptyset$	
	$\langle A, C \rangle$	$\langle A_1, C_4 \rangle$	$\{D_2\}$	$\frac{3}{5}$
		$\langle A_4, C_1 \rangle$	$\emptyset$	
		$\langle A_4, C_3 \rangle$	$\{D_1\}$	
		$\langle A_5, C_3 \rangle$	$\emptyset$	
	$\langle A, D \rangle$	$\langle A_1, D_2 \rangle$	$\emptyset$	$\frac{2}{5}$
		$\langle A_4, D_1 \rangle$	$\emptyset$	
	$\langle B, C \rangle$	$\langle B_2, C_1 \rangle$	$\emptyset$	$\frac{1}{4}$
		$\langle B_2, C_3 \rangle$	$\{D_1\}$	
	$\langle B, D \rangle$	$\langle B_2, D_1 \rangle$	$\emptyset$	$\frac{1}{4}$
	$\langle C, D \rangle$	$\langle C_3, D_1 \rangle$	$\emptyset$	$\frac{2}{4}$
		$\langle C_4, D_2 \rangle$	$\emptyset$	
3	$\langle A, B, C \rangle$	$\langle A_4, B_2, C_1 \rangle$	$\emptyset$	$\frac{1}{5}$

		$\langle A_4, B_2, C_3 \rangle$	$\emptyset$	
	$\langle A, B, D \rangle$	$\langle A_4, B_2, D_1 \rangle$	$\emptyset$	$\frac{1}{5}$
	$\langle A, C, D \rangle$	$\langle A_1, C_4, D_2 \rangle$	$\emptyset$	$\frac{2}{5}$
		$\langle A_4, C_3, D_1 \rangle$	$\emptyset$	
	$\langle B, C, D \rangle$	$\langle B_2, C_3, D_1 \rangle$	$\emptyset$	$\frac{1}{4}$
4	$\langle A, B, C, D \rangle$	$\langle A_4, B_2, C_3, D_1 \rangle$	$\emptyset$	$\frac{1}{5}$

Proof. Given a co-location instance list $C_k = \langle I_1, \dots, I_k \rangle$, $NC(C_k) = NC(I_1) \cap \dots \cap NC(I_k)$ by Definition 4. $\forall I_j \in NC(C_k)$, $I_j \in NC(I_1) \cap \dots \cap NC(I_k)$; in other words, $I_j \in NC(I_1)$, ..., $I_j \in NC(I_k)$, which implies that $(I_1, I_j) \in R$, ..., $(I_k, I_j) \in R$. Since C_k is a clique (by Definition 1) and I_j is a neighbor to every instance of C_k , $C_{k+1} = \langle I_1, \dots, I_k, I_j \rangle$ is guaranteed to be a clique. $\square$

Lemma 3 shows that a valid co-location instance list generated in the NCA algorithm preserves the cliqueness property all the time, and NC is the enabling mechanism. Because of this characteristic of NC, the NCA algorithm does not need cliqueness checking for every co-location instance list.

5.3 The Algorithm

When frequent co-location type lists are available, interesting co-location rules are generated from them. An interesting co-location rule is a rule whose confidence is

greater than or equal to the given confidence threshold. The confidence of a co-location rule is calculated using the confidence formula defined in Chapter 3.

Inputs:

F : a set of feature types;

I : a set of instances defined over F ;

R : a set of neighborhood relationships between the instances in I ;

p : minimum participation ratio threshold;

c : minimum confidence threshold;

Output:

CR : a set of co-location rules whose confidence is larger than or equal to c .

Variables:

k : length of the current co-location type lists;

L_k : set of co-location instance lists of length- k ;

NC_k : set of neighbor clusters of co-location instance lists of length- k ;

T_k : set of co-location type lists of length- k ;

Algorithm:

$NC_1 := generate_length_1_neighbor_clusters(F, I, R);$

$L_2, NC_2, T_2 := length_2_mining(F, I, NC_1);$

$L_2, NC_2, T_2 := filter_infrequent_co - location_ \\ type_lists(L_2, T_2, NC_2, p);$

$k := 3;$

while $k \leq |F|$ and $T_{k-1} \neq \emptyset$


```

 $T_k := \text{generate\_candidate\_co - location\_type\_lists}(T_{k-1});$ 
for each  $t$  in  $T_k$ 
     $L_k, NC_k \cup = \text{generate\_co - location\_}$ 
         $\text{instance\_lists}(L_{k-1}, NC_{k-1}, t);$ 
end for
 $L_k, NC_k, T_k := \text{filter\_infrequent\_co - location\_}$ 
     $\text{type\_lists}(L_k, T_k, NC_k, p);$ 
end while
 $CR := \text{generate\_co - location\_rules}(c, F \cup T_2 \cup \dots, I \cup L_2 \cup \dots);$ 

```

CHAPTER 6

COMPLEXITY ANALYSIS

In this chapter, the time complexity of the NCA algorithm is analyzed. In addition, the complexity of NCA in comparison with the joinless algorithm [24] is presented to contrast the benefit of NCA. The joinless algorithm was chosen because it is the most notable state-of-the-art algorithm.

Both algorithms can be divided into three phases:

1. Phase 1: Generate a neighbor cluster in NCA or a star neighborhood in the joinless algorithm for each feature instance from the given neighborhood relationships set R . Since both neighbor clusters and star neighborhoods are generated by inspecting each element of R , the time complexity of this step is proportional to $|R|$.
2. Phase 2: Mine length-2 co-location patterns.
3. Phase 3: Iterate the process to mine co-location patterns that are longer than 2.

For Phase 2, the joinless algorithm and NCA generate length-2 co-location instance lists and co-location type lists from star neighborhoods and neighbor clusters respectively, and prune infrequent co-location type lists.

Compared to the joinless algorithm, however, NCA requires an extra set intersection operation between two neighbor clusters to generate the NC of the length-2 co-location instance list of the two corresponding instances. Since co-location instance lists are generated from $NC(I_k)$ by concatenating each element e of $NC(I_k)$ to I_k one at a time followed by performing the set intersection operation between $NC(I_k)$ and $NC(e)$,

the total cost for this step is $O(|I| \times \overline{NC} \times \tau)$ where $\overline{NC}$ is the average length of the neighbor clusters and τ is the cost of the set intersection operation (which is linear to $\overline{NC}$ as the instances in a neighbor cluster are sorted), which in return yields $O(|I| \times \overline{NC}^2)$. In contrast, the complexity of the joinless algorithm is $O(|I| \times \overline{SN})$ where $\overline{SN}$ is the average length of the star neighborhoods.

In fact, the difference in the real world complexities of the two algorithms at this phase is usually very small since spatial datasets are often sparse, and as a result, $\overline{NC}$ is much smaller than $|I|$ and NCs are even $\emptyset$ in many cases.

Since both algorithms require the same number of iterations to mine co-location patterns of length greater than 2 in the third phase, let us consider just one iteration to compare the two algorithms.

In the joinless algorithm, five steps are involved as follows:

1. Generate all possible candidate co-location type lists of length- $(k + 1)$ from the co-location type lists of length- k . This is done by joining co-location type lists of length- k with the same length- $(k - 1)$ prefix. Assume that the number of frequent length- k co-location type lists is N_k , then the time complexity at this stage is $O(k^3 \times N_k^3)$.
2. Assuming that the number of candidate length- $(k + 1)$ co-location type lists is N_{k+1} , for each candidate co-location type list, generate all possible star instances from the corresponding star neighborhoods: This is done by performing a pattern matching of each candidate co-location type list of

length- $(k + 1)$ in the star neighborhood set, which is $O(N_{k+1} \times |I|/|F| \times p)$ where p is the pattern matching cost.

3. Prune infrequent candidate co-location type lists using the star instances. This is done by counting the number of star instances for every type in each co-location type list, which is $O(M_{k+1} \times (k + 1))$, where M_{k+1} is the number of the star instances of length- $(k + 1)$.
4. Prune each star instance that is not a clique. This is done by performing a pattern matching for each star instance of length- $(k + 1)$ to the set of star instances of length- k , which costs $O(M_k \times M_{k+1} \times p)$, where M_k is the number of co-location instance lists mined in iteration k .
5. Prune infrequent candidate co-location type lists again using the remaining star instances (Step 3 above).

In contrast, the NCA algorithm involves three steps as follows:

1. Generate all possible candidate length- $(k + 1)$ co-location type lists from the co-location type lists of length- k . This is identical to Step 1 of the joinless algorithm.
2. For each candidate length- $(k + 1)$ co-location type list T_{k+1} , generate all the possible candidate length- $(k + 1)$ co-location instance lists from length- k co-location instance lists of co-location type list T_k , where T_k is the length- k prefix of T_{k+1} , along with neighbor clusters as well. This is done by concatenating each element e of type F_{k+1} in $NC(L_k)$ to the length- k co-location instance list L_k one at a time for every L_k of type T_k , followed by

performing the set intersection operation between $NC(L_k)$ and $NC(e)$, the total cost for this step is $O(M_k \times NC_k \times \tau)$ where M_k is defined similarly to Step 3 of the joinless algorithm, and NC_k is the average length of the neighbor clusters at the k^{th} iteration, and τ is the cost of the set intersection operation.

3. Prune infrequent co-location type lists. Step 3 of the joinless algorithm.

As we can see, the time complexity of Step 1 in both algorithms is the same.

Based on the assumption that $p \approx \tau$, the complexity of Step 4 of the joinless algorithm is greater than or equal to that of the Step 2 of NCA because the star instance set of the joinless algorithm might include non-clique instance lists. For the same reason, the time complexity of NCA is also less than or equal to that of the joinless algorithm at Step 3. Moreover, the joinless algorithm requires an extra Step 2 and Step 5. The benefit of NCA on time complexity is repeated for every other k , which is more significant than the onetime extra cost incurred in NCA at Phase 2.

CHAPTER 7

EXPERIMENTAL ANALYSIS

For the empirical evaluation of NCA, the same strategy used in the complexity analysis is taken: that is, the performance of NCA was evaluated in contrast to the aforementioned the joinless algorithm. To this end, both algorithms were implemented in Java using JDK 1.5 and tested on a Windows XP computer with 2GB of RAM. They were tested on running time and space requirement over a number of synthetic and real GIS datasets using various testing parameters (details are shown below). Space requirement was measured by the number of candidate co-location instance lists generated by the two algorithms.

The performance gain of NCA over the joinless algorithm is consistent both with the synthetic and real datasets as presented in the following sections. Here, candidate co-location instance list means any candidate co-location instance list generated by two algorithms, including nonclique candidate co-location instance lists (in case of the joinless algorithm), at any stage of the mining process.

7.1 Synthetic Dataset

15 synthetic spatial datasets were generated, each of which ranged from 10,000 to 30,000 spatial feature instances, and 20 to 60 feature types, as shown in Table 4 with other properties. In these datasets, $|I|$ = number of instances, $|F|$ = number of types, NT = number of different frequentness threshold values used, MT = maximum threshold value used. Each dataset was generated by first randomly generating instances on a $1,000 \times 1,000$ spatial grid, and then randomly assigning feature types on these instances.

In this experiment, the neighborhood radius was set to 10 grid units. Here, $|I|$ is the number of instances, $|F|$ is the number of types, NT is the number of different frequentness threshold values used, MT is the maximum threshold value used.

Table 4. Fifteen Synthetic Datasets Used in the Experiment.

Dataset	$ I $	$ F $	NT	MT
Dataset 1	10,000	20	20	0.20
Dataset 2	10,000	30	20	0.15
Dataset 3	10,000	40	20	0.15
Dataset 4	10,000	50	20	0.15
Dataset 5	10,000	60	20	0.15
Dataset 6	20,000	20	20	0.35
Dataset 7	20,000	30	20	0.25
Dataset 8	20,000	40	20	0.20
Dataset 9	20,000	50	20	0.20
Dataset 10	20,000	60	20	0.15
Dataset 11	30,000	20	20	0.40
Dataset 12	30,000	30	20	0.30
Dataset 13	30,000	40	20	0.25
Dataset 14	30,000	50	20	0.25
Dataset 15	30,000	60	20	0.20

Notice that NT 's value is always 20, which means the performance of both algorithms was evaluated under 20 different frequentness thresholds on each dataset. If the length of the longest patterns discovered in the dataset is 2, the testing would not continue with a higher threshold because, as the threshold value increases, the length of the longest pattern that can be found tends to decrease. In fact, mining length 2 spatial co-location patterns can be done by calculating the participation ratios directly from the original neighborhood relationships set without applying any particular co-location mining algorithm.

7.1.1 Space Requirement. Numbers of candidate co-location instance lists generated by the two algorithms on the 15 datasets were compared. Regardless of the differences in the number of feature instances and types, NCA never generated more candidates than the joinless algorithm did, and usually it generated fewer candidates, especially when frequentness thresholds are small. In addition, the lower the threshold was, the larger the space requirement gain of NCA over the joinless algorithm became in any combinations of number of instances and number of types.

The comparison of numbers of candidates generated by both algorithms on one of the synthetic datasets is shown in Figure 6. This dataset has 40 feature types and 30,000 instances. From this figure, we can see that, at level 2, both algorithms generated the same number of candidates because all the candidates of length 2 automatically satisfy cliqueness. However, at levels greater than 2, NCA always generated a lower number of candidates than the joinless algorithm because NCA guarantees to generate clique candidates only at every level whereas the joinless algorithm generates additional non-

Figure 6. Comparison of numbers of candidates generated by both algorithms on the synthetic dataset with 40 feature types and 30,000 instances.

clique candidates at any level greater than 2.

The observed advantage of NCA with respect to the space requirement was consistent in all the test cases using the 15 datasets. The comparisons using other synthetic datasets are not presented here, but they all had similar patterns. Certainly, the overall number of candidates generated by an algorithm has a negative impact on the running time of the algorithm as discussed next.

7.1.2 Running Time. The running time of the two algorithms were examined using the same 15 datasets with various frequentness threshold values. NCA outperformed the joinless algorithm in these tests. The results of some of the synthetic datasets are shown in Figure 7 and Figure 8. The performance gain pattern of NCA was consistent with

Figure 7. Comparison of both algorithms on running times on three synthetic datasets.

Figure 8. Comparison of both algorithms on running times on two synthetic datasets.

synthetic datasets of other parameter settings.

7.2 Real Dataset

To test the two algorithms with real datasets, the Oregon, Washington, Indiana, and Kansas USGS geoname datasets made available by the U.S. Board on Geographic Names were used. Before the experiment, noise records were cleaned from all the datasets by removing instances whose longitude or latitude is 0 which implies an unknown value according to the U.S. Board on Geographic Names. After cleaning, the number of instances ranged from 17,000 to 52,000 and the number of feature types ranges from 46 to 61 as shown in Table 5. In these datasets, $|I|$ is the number of instances, $|F|$ is the number of types.

The neighborhood diameter used with the Oregon and Washington datasets was set to approximately half mile, whereas the neighborhood diameter used with the Indiana and Kansas datasets was set to approximately one mile. The frequentness threshold values used for the Oregon and Washington datasets ranged from 0.035 to 0.7, from 0.032 to 0.64 for the Indiana dataset experiment, and from 0.025 to 0.5 for the Kansas

Table 5. Four Real Datasets Used in the Experiment.

Dataset	$ I $	$ F $
Indiana	25,233	53
Kansas	17,842	46
Oregon	52,359	61
Washington	31,796	59

dataset experiment.

7.2.1 Space Requirement. Numbers of candidate co-location instance lists generated by the two algorithms were compared. The patterns of numbers of candidates generated from the four datasets, as shown in Figures 9, 10, 11, and 12, are very similar to what we have observed from the synthetic datasets as presented in the previous sections. NCA clearly generates less candidates overall than the joinless algorithm.

7.2.2 Running Time. The performance of both algorithms in terms of running time using the four real datasets was compared. The results are shown in Figures 13 and 14. NCA outperforms the joinless algorithm in all datasets, similar to what we have observed in the running time test using the synthetic datasets, presented in the previous section.

Figure 9. Comparison of numbers of candidates generated by both algorithms on the Oregon dataset.

Figure 10. Comparison of numbers of candidates generated by both algorithms on the Washington dataset.

Figure 11. Comparison of numbers of candidates generated by both algorithms on the Indiana dataset.

Figure 12. Comparison of numbers of candidates generated by both algorithms on the Kansas dataset.

7.3 Discussion

As we can see, NCA never generates more candidate co-location instance lists than the joinless algorithm at any level of co-location mining at any threshold value regardless of different number of feature instances and types. Since NCA does not generate any nonclique candidates whereas the joinless algorithm does, it always generates fewer candidates than the joinless algorithm in every mining session.

7.3.1 Effect of Level. It is to be noted that there is no difference at level 1 among Apriori-like level-wise algorithms: i.e., the occurrence count of each single item is obtained through a database scan. As the mining process proceeds toward higher levels in the item set lattice, the candidate population decreases dramatically due to pruning, thus

leaving only a small number of surviving candidates, in which the performance difference among comparable level-wise algorithms becomes insignificant. Hence, the advantage of NCA over the joinless algorithm with respect to the number of candidates generated is anticipated at level 2 and a few higher levels, as shown in all the figures in this chapter.

7.3.2 Effect of Threshold. Generally, the performance difference among comparable level-wise algorithms is negligible when the given threshold is high because most candidates are pruned anyway. Hence, the real performance comparison makes sense only when the threshold is relatively low, i.e., when a large number of database scans and combinatorial pattern matching are in demand. As shown in the figures in this chapter, the benefit of NCA over the joinless algorithm in terms of the number of candidates generated with relatively low threshold values is obvious.

7.3.3 Effect of Dimensionality and Data Size. As the dimension of the dataset increases, all the level-wise algorithms including NCA and the joinless algorithm suffer from the curse of dimensionality. It, however, has no impact on the performance comparison between NCA and the joinless algorithm. As the data size becomes larger, more candidates are likely generated. In such a case, NCA's advantage over the joinless algorithm increases, as well. For example, the performance difference between the two algorithms becomes wider with 52,359 instances than with 31,796 instances, as shown in Figure 14. This phenomenon is also observed in Figure 7 and Figure 13.

7.3.4 Effect of Number of Candidates on Running Time. Due to the smaller number of candidates generated in NCA as compared to the joinless algorithm, the

overall running time of NCA is generally less than that of the joinless algorithm.

However, when both algorithms terminate at level 2 due to a high threshold or small data size, the joinless algorithm runs slightly faster than NCA. This is because at level 2, the joinless algorithm does not perform explicit cliqueness checking. However, NCA still performs NC generation at level 2. Overall, NCA runs faster than the joinless algorithm when a substantial number of candidates of length 3 or longer are generated to offset the cost of set intersection operations at level 2 over the cost of cliqueness checking.

Figure 13. Comparison of both algorithms on running times on the Kansas and Indiana datasets.

Figure 14. Comparison of both algorithms on running times on the Oregon and Washington datasets.

CHAPTER 8

CONCLUSION

In this research, a spatial co-location mining problem was investigated and a novel co-location mining algorithm was proposed, which does not require cliqueness checking for each co-location candidate at each iteration.

This thesis shows that this goal was accomplished through introducing a novel data structure called neighbor cluster (NC). This thesis also analytically demonstrates the benefit of the NC data structure over the star neighborhood data structure with respect to space requirement and running time, which was repeatedly and consistently validated through a series of testing on selected real GIS and synthetic datasets, in addition to the results of analytical scrutiny.

Future work for this project should include the following:

1. Investigate a way to avoid NC generation at level 2 when the mining process terminates after level 2, thus making NCA more compelling.
2. In the current work, the notion of neighborhood is uniform in size and shape throughout the entire search space. Future research should consider different options to relax this constraint to model real world scenarios.
3. The proposed algorithm is based on the assumption that the entire dataset can fit into the main memory. However, in real applications, the spatial datasets are usually very big and cannot be handled within the main memory. A disk-resident algorithm needs to be proposed.

4. The co-location mining problem solved in this project does not involve temporal data. It assumes that the locations of spatial instances will not change. However, this may not be true in many applications. For example, in the field of ecology, biological species are constantly migrating over time. The discovery of consistent co-location patterns as time goes by would be desirable in these applications.

REFERENCES

- [1] Agrawal, R. and Srikant, R. Fast Algorithms for mining association rules. In *Proceedings of 20th International Conference on Very Large Databases*, 1995.
- [2] Arge, L., Procopiuc, O., Ramaswamy, S., Suel, T., and Vitter, J. S. Scalable sweeping-based spatial join. In *Proceedings of 24 International Conference on Very Large Databases*, 1998.
- [3] Berge, C. *Graphs and Hypergraphs*. Elsevier, 1976.
- [4] Celik, M., Kang, J. M., and Shekhar, S. Zonal co-location pattern discovery with dynamic parameters. In *Proceedings of 7th IEEE International Conference on Data Mining*, 2007.
- [5] Celik, M., Shekhar, S., Rogers, J. P., Shine, J. A., and Yoo, J. S. Mixed-drove Spatio-temporal co-occurrence pattern mining: A summary of results. In *Proceedings 6th IEEE International Conference on Data Mining*, 2006.
- [6] Estivill-Castro, V. and Lee, I. Data mining techniques for autonomous exploration of large volumes of geo-referenced crime data. In *Proceedings of 6th International Conference on Geocomputation*, 2001.
- [7] Graefe, G. Sort-merge-join: An idea whose time has(h) passed? In *Proceedings of the 10th International Conference on Data Engineering*, 1994.
- [8] Huang, Y., Pei, J. and Xiong, H. Mining Co-Location Patterns with Rare Events from Spatial Data Sets. *GeoInformatica* 10, 3 (2006), 239-260.
- [9] Huang, Y. and Zhang, P. On the relationships between clustering and spatial co-location pattern mining. In *Proceedings of the 18th IEEE International Conference on Tools with Artificial Intelligence*, 2006.
- [10] Koperski, K. and Han, J. Discovery of spatial association rules in geographic information databases. *Lecture Notes in Computer Science* 951 (1995), 47-66.
- [11] Kuramochi, M. and Karypis, G. GREW-A scalable frequent subgraph discovery algorithm. In *Proceedings of the 4th IEEE International Conference on Data Mining*, 2004.
- [12] Morimoto, Y. Mining frequent neighboring class sets in spatial databases. In *Proceedings of the 7th ACM Conference on Knowledge Discovery and Data Mining*, 2001.

- [13] Munro, R., Chawla, S., and Sun, P. Complex Spatial Relationships. In *Proceedings of the IEEE Bioinformatics Conference*, 2003.
- [14] Patel, J. M. and DeWitt, D. J. Partition based spatial-merge join. In *ACM Conference on Management of Data*, 1996.
- [15] PostGIS: Documentation. 2008. <http://postgis.refractory.net/documentation/>, Nov. 2008.
- [16] Shekhar, S. and Chawla, S. *Spatial Databases: A Tour*. Prentice Hall, 2003.
- [17] Shekhar, S. and Huang, Y. Discovering spatial co-location patterns: A summary of results. In *Proceedings of 7th International Symposium on Advances in Spatial and Temporal Databases*, 2001.
- [18] Shekhar, S., Huang, Y., Wu, W., Lu C. T., and Chawla, S. What's spatial about spatial data mining: Three Case Studies. In *Data Mining for Scientific and Engineering Applications*, R.L. Grossman, Ed. Kluwer Academic Publishing, 2001, 487-514.
- [19] Silberschatz, A., Korth, H. F., and Sudarshan, S. *Database System Concepts*, 5th ed. McGraw-Hill, 2005.
- [20] Xiong, H., Shekhar, S., Huang, Y., Kumar, V., Ma, X., and Yoo, J. S. A framework for discovering co-location patterns in data sets with extended spatial objects. In *Proceedings of 2004 SIAM International Conference on Data Mining*, 2004.
- [21] Yan, X. and Han, J. gSpan: Graph-based substructure pattern mining. In *Proceedings of 2002 IEEE International Conference on Data Mining*, 2002.
- [22] Yang, H., Parthasarathy, S., and Mehta, S. A generalized framework for mining spatio-temporal patterns in scientific data. In *Proceedings of the 11th ACM International Conference on Knowledge Discovery in Data Mining*, 2005.
- [23] Yoo, J. S. and Shekhar, S. A partial join approach for mining co-location patterns. In *Proceedings of the 12th ACM International Symposium on Advances in Geographic Information Systems*, 2004.
- [24] Yoo, J. S. and Shekhar, S. A joinless approach for mining spatial co-location patterns. *IEEE Trans. Knowledge and Data Engineering* 18, 10 (2006), 1323-1337.

- [25] Yoo, J. S., Shekhar, S., Kim, S., and Celik, M. Discovery of co-evolving spatial event sets. In *Proceedings of the SIAM International Conference on Data Mining*, 2006.
- [26] Zhang, X., Mamoulis, N., Cheung, D. W., and Shou, Y. Fast mining of spatial collocations. In *Proceedings of the 10th ACM International Conference on Knowledge Discovery and Data Mining*, 2004.