Fast Mining of Spatial Collocations*

Xin Zhang, Nikos Mamoulis, David W. Cheung, Yutao Shou
Department of Computer Science
The University of Hong Kong
Pokfulam Road, Hong Kong
{xzhang,nikos,dcheung,ytshou}@cs.hku.hk

ABSTRACT

Spatial collocation patterns associate the co-existence of non-spatial features in a spatial neighborhood. An example of such a pattern can associate contaminated water reservoirs with certain deceases in their spatial neighborhood. Previous work on discovering collocation patterns converts neighborhoods of feature instances to itemsets and applies mining techniques for transactional data to discover the patterns. We propose a method that combines the discovery of spatial neighborhoods with the mining process. Our technique is an extension of a spatial join algorithm that operates on multiple inputs and counts long pattern instances. As demonstrated by experimentation, it yields significant performance improvements compared to previous approaches.

Categories and Subject Descriptors

H.2.8 [**Database Management**]: Database Applications—*Data Mining, Spatial Databases and GIS*

General Terms

Algorithms

Keywords

Collocation Pattern, Spatial Databases

1. INTRODUCTION

Spatial database management systems (SDBMSs) [17] manage large collections of multidimensional data which, apart from conventional features, include spatial attributes (e.g., location). They support efficient operations of basic retrieval tasks like spatial range queries, nearest neighbor search, spatial joins, etc. On the other hand, SDBMSs do not explicitly store patterns or rules that associate the spatial relationships between objects with some of their non-spatial features. As a result, there is an increasing interest in the automatic discovery of such information, which finds use in a variety of disciplines including environmental research, government services layout, and geo-marketing.

Classic spatial analysis tasks include spatial clustering [16, 14, 15], spatial characterization [8], spatial classification [6], and spatial trend detection [9]. In this paper, we focus on a recent mining problem; that of identifying groups of particular features that appear frequently close to each other in a geo-spatial map. This problem is also referred to as the *mining collocation patterns* problem [18, 4, 11]. As an application, consider an E-commerce company that provides different types of services, such as weather, timetabling and ticketing queries [10]. The requests for those services may be sent from different locations by (mobile or fixed-line) users. The company may be interested in discovering types of services that are requested by geographically neighboring users, in order to provide location-sensitive recommendations to them for alternative products. For example, having known that ticketing requests are frequently asked close to timetabling requests, the company may choose to advertise the ticketing service to all customers that ask for a timetabling service. As another example, a collocation pattern can associate contaminated water reservoirs with a certain decease in their spatial neighborhood.

As a more concrete definition of the problem, consider a number n of spatial datasets $R_1, R_2, \ldots, R_n$, such that each R_i contains objects that have a common non-spatial feature f_i. For instance, R_1 may store locations of weather requests, R_2 may store locations of timetabling requests, etc. Given a distance threshold ϵ, two objects on the map (independently of their feature labels) are *neighbors* iff their distance is at most ϵ. We can define a *collocation pattern* P by an undirected connected graph, where each node corresponds to a feature and each edge corresponds to a neighborhood relationship between the corresponding features. For example, consider a pattern with three nodes, labeled "timetabling", "weather", and "ticketing", and two edges connecting "timetabling" with "weather" and "timetabling" with "ticketing". An *instance* of a pattern P is a set of objects that satisfy the unary (feature) and binary (neighborhood) constraints specified by the pattern's graph. An instance of our example pattern is a set $\{o_1, o_2, o_3\}$, where $label(o_1)$="timetabling", $label(o_2)$="weather", $label(o_3)$ = "ticketing" (unary constraints) and $dist(o_1, o_2) \leq \epsilon$, $dist(o_1, o_3) \leq \epsilon$ (spatial binary constraints).

Interestingness measures for collocation patterns express the statistical significance of their instances. They can as-

*work supported by grant HKU 7149/03E from Hong Kong RGC.

KDD'04 August 22–25, 2004, Seattle, Washington, USA.

sist the derivation of useful rules that associate the instances of the features. We adopt the measures of [18, 4] (to be reviewed in Section 2), which can be used to solve several variants of the mining problem. Based on the interestingness measures, we propose a method that combines the discovery of spatial neighborhoods with the mining process. Note that the discovery of object neighborhoods (or else pattern instances) is in fact a (multi-way) spatial join problem. We extend a hash-based spatial join algorithm to operate on multiple feature sets in order to identify such neighborhoods. The algorithm divides the map and partitions the feature sets using a regular grid. While identifying object neighborhoods in each partition, at the same time, the algorithm attempts to discover prevalent and/or confident such patterns by counting their occurrences at production time. If the memory is not sufficient to discover the frequent patterns in one pass over all partitions, we employ the partitioning mining paradigm to solve the problem. As demonstrated by experimentation, our technique yields significant performance improvements compared to previous methods that do not integrate the mining process with the spatial query processing part.

The rest of the paper is organized as follows. Section 2 provides backgound in spatial data mining. Section 3 describes our method for discovering collocation patterns and related association rules. Sections 4 and 5 present our experimental results and conclude the paper.

2. BACKGROUND

Past research on mining spatial associations is based on two models; the reference feature model and the collocation patterns model. In this section we briefly review these models and discuss their pros and cons.

2.1 Reference feature collocations

The problem of mining association rules based on spatial relationships (e.g., adjacency, proximity, etc.) of events or objects was first discussed in [5]. The spatial data are converted to transactional according to a *centric reference feature* model. Consider a number n of spatial datasets $R_1, R_2, \ldots, R_n$, such that each R_i contains all objects that have a particular non-spatial feature f_i. Given a feature f_i, we can define a transactional database as follows. For each object o_i in R_i a spatial query is issued to derive a set of features $I = \{f_j : f_j \neq f_i \wedge \exists o_j \in R_j(dist(o_i, o_j) \leq \epsilon)\}$.[1] The collection of all feature sets I for each object in R_i defines a transactional table T_i. T_i is then mined using some itemsets mining method (e.g., [1, 19]). The *frequent* feature sets I in this table, according to a minimum support value, can be used to define rules of the form:

$$label(o) = f_i \Rightarrow o \text{ close to some } o_j \in R_j, \forall f_j \in I$$

The support of a feature set I defines the confidence of the corresponding rule. For example, consider the three object-sets shown in Figure 1. The lines indicate object pairs within distance ϵ from each other. The shapes indicate different features. Assume that we want to extract rules having feature a on their left-hand side. In other words, we want to find features that occur frequently close to feature a. For each instance of a, we generate an itemset; a_1 generates $\{b, c\}$ because there is at least one instance of b (e.g., b_1 and b_2) and one instance of c (e.g., c_1) close to a_1. Similarly, a_2 generates itemset $\{b\}$ (due to b_2). Let 75% be the minimum confidence. We first discover frequent itemsets (with minimum support 75%) in $T_a = \langle\{b, c\}, \{b\}\rangle$, which gives us a sole itemset $\{b\}$. In turn, we can generate the rule

$$label(o) = a \Rightarrow o \text{ close to } o_j \text{ with } label(o_j) = b,$$

with confidence 100%. For simplicity, in the rest of the discussion, we will use $f_i \Rightarrow I$ to denote rules that associate instances of feature f_i with instances of feature sets I, $f_i \notin I$, within its proximity. For example, the rule above can be expressed by $a \Rightarrow \{b\}$. The mining process for feature a can be repeated for the other features (e.g., b and c) to discover rules having them on their left side (e.g., we can discover rule $b \Rightarrow \{a, c\}$ with conf. 100%). Note that the features on the right hand side of the rules are not required to be close to each other. For example, rule $b \Rightarrow \{a, c\}$ does not imply that for each b the nearby instances of a and c are close to each other. In Figure 1, observe that although b_2 is close to instances a_1 and a_2 of a and instance c_2 of c, c_2 is neither close to a_1, nor to a_2. Methods for discovering rules requiring the right-hand side features to form a neighborhood will be reviewed in the next subsection.

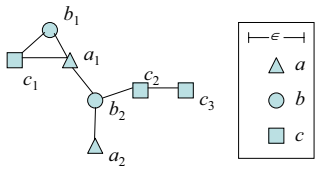


Figure 1: Mining example

2.2 Collocation patterns

Recently [10, 18, 4, 11], the research interest shifted towards mining *collocation patterns*, which are feature sets with instances that are located in the same neighborhood. A pattern P of length k is described by a set of features $\{f_1, f_2, ..., f_k\}$. A valid instance of P is a set of objects $\{o_1, o_2, ..., o_k\} : (\forall 1 \leq i \leq k,\ o_i \in R_i) \wedge (\forall 1 \leq i < j \leq k,\ dist(o_i, o_j) \leq \epsilon)$. In other words, all pairs of objects in a valid pattern instance should be close to each other, i.e., the closeness relationships between the objects should form a *clique graph*. Consider again Figure 1 and the pattern $P = \{a, b, c\}$. $\{a_1, b_1, c_1\}$ is an instance of P, but $\{a_1, b_2, c_2\}$ is not.

[18, 4] define some useful measures that characterize the interestingness of collocation patterns. The first is the *participation ratio* $pr(f_i, P)$ of a feature f_i in pattern P, which is defined by the following equation:

$$pr(f_i, P) = \frac{\#\text{ instances of } f_i \text{ in any instance of } P}{\#\text{ instances of } f_i} \quad (1)$$

Using this measure, we can define *collocation rules* that associate features with the existences of other features in their neighborhood. In other words, we can define rules of the form $label(o) = f_i \Rightarrow o$ participates in an instance of P with confidence $pr(f_i, P)$. These rules are similar to the ones defined in [5] (see previous subsection); the difference

[1] Note that we used a distance relationship ($dist(o_i, o_j) \leq \epsilon$) in this definition; in general, any spatial relationship (i.e., topological, distance, directional) between R_i and R_j could be prescribed by the data analyst.

here is that there should be neighborhood relationships between all pairs of features on the right hand side of the rule. For example, $pr(b, \{a, b, c\}) = 0.5$ implies that 50% of the instances of b (i.e., only b_1) participate in some instance of pattern $\{a, b, c\}$ (i.e., $\{a_1, b_1, c_1\}$).

The *prevalence* $prev(P)$ of a pattern P is defined by the following equation:

$$prev(P) = \min\{pr(f_i, P), f_i \in P\} \quad (2)$$

For example, $prev(\{b, c\}) = 2/3$, since $pr(b, \{b, c\}) = 1$ and $pr(c, \{b, c\}) = 2/3$. The prevalence captures the minimum probability that whenever an instance of some $f_i \in P$ appears on the map, then it will participate in an instance of P. Thus, it can be used to characterize the strength of the pattern in implying collocations of features. In addition, prevalence is monotonic; if $P \subseteq P'$ then $prev(P) \geq prev(P')$. For example, since $prev(\{b, c\}) = 2/3$, we know that $prev(\{a, b, c\}) \leq 2/3$. This implies that the a priori property holds for the prevalence of patterns and algorithms like Apriori [1] can be used to mine them in a level-wise manner [18].

Finally, the *confidence* $conf(P)$ of a pattern P is defined by the following equation:

$$conf(P) = \max\{pr(f_i, P), f_i \in P\} \quad (3)$$

For example, $conf(\{b, c\}) = 1$, since $pr(b, \{b, c\}) = 1$ and $pr(c, \{b, c\}) = 2/3$. The confidence captures the ability of the pattern to derive collocation rules using the participation ratio. If P is confident with respect to a minimum confidence threshold, then it can derive at least one collocation rule (for the attribute f_i with $pr(f_i, P) = conf(P)$). In Figure 1, $conf(\{b, c\}) = 1$ implies that we can find one feature in $\{b, c\}$ (i.e., b) every instance of which participates in an instance of $\{b, c\}$. Given a collection of spatial objects characterized by different features, a minimum prevalence threshold min_prev, and a minimum confidence threshold min_conf, a data analyst could be interested in discovering prevalent and/or confident patterns, and the collocation rules derived by them. The confidence of a collocation rule between two patterns, $P_1 \rightarrow P_2, P_1 \cap P_2 = \varnothing$, can be defined by the conditional probability that an instance of P_1 participates in some instance of $P_1 \cup P_2$ (given that $P_1 \cup P_2$ is prevalent with respect to min_prev) [18].

Previous methods on mining (prevalent or confident) collocation patterns [10, 18, 4, 11] separate the part that evaluates the spatial relationships from the mining part. The typical approach is to initially perform a spatial (distance) join [17] to retrieve object pairs within distance ϵ to each other to generate the instances of 2-length patterns. The prevalence of these patterns is then evaluated and only prevalent ones (and their instances) are kept. From two k-length patterns P_1 and P_2, a candidate pattern P_3 of length $k + 1$ is generated, by Apriori-based candidate generation [1] (the first $k - 1$ features of P_1 and P_2 should be common and the last ones are the additional features in P_3). Each candidate pattern (whose subsets are all prevalent) is then validated by joining the instances of P_1 and P_2 where the first $k - 1$ feature instances are common. The distance of the last two feature instances in P_3 is then checked to validate whether they are close to each other. After finding all valid instances of a candidate pattern P_3, we need to validate whether they are prevalent by counting the participation ratio of each feature in them.

Consider again the example of Figure 1. After spatially joining the object-set pairs (a, b), (b, c), and (a, c), we retrieve the instances of the corresponding patterns. For example, the instances of $P_1 = \{a, b\}$ are $\{(a_1, b_1), (a_1, b_2), (a_2, b_2)\}$. Moreover, their prevalences are computed (100% for P_1). Note that in order to compute the prevalences, we need a bitmap for each set, that marks the objects that are found in each pattern. For example we use a bitmap for a to mark its instances that participate in P_1 and after counting the number of 1's in it, we can find that $pr(a, \{a, b\}) = 100\%$. If $min\_prev = 30\%$, observe that also $P_2 = \{b, c\}$ and $P_3 = \{a, c\}$ are prevalent. From P_1 and P_3, we can generate $P_4 = \{a, b, c\}$ (since its subset P_2 is also prevalent). The instances of P_4 are generated by joining the instances of P_1 with those of P_3 on their agreement on the instance of a. Since the only instance of P_3 is (a_1, c_1), the only potential instance of P_4 is (a_1, b_1, c_1), which is committed after verifying that b_1 is close to c_1. The prevalence of P_4 (=1/3) is then verified.

This technique is adapted in [18, 4, 11] to mine prevalent and confident patterns. For confident patterns [4], the method is slightly changed since confidence is not monotonic, but has a *weak* monotonicity property; given a k-length pattern P at most one $(k-1)$-length sub-pattern P' of P may have lower confidence than P. In [10], a similar approach is followed. However, a particular instance of a feature f_i is not allowed to participate into multiple instances of a pattern P that includes f_i. This affects the quality of the results, since the actual number of pattern instances is underestimated. Also, depending on which pattern instance a particular feature is assigned to, we can have different mining results.

The techniques discussed so far suffer from certain efficiency problems. First, they require a potentially large number of $(k-1)$-length pattern instances to be computed before k-length patterns can be discovered. As discussed in [11], these instances can be too many to fit in memory. Also the computational cost of processing them and computing participation ratios from them is very high. In addition, spatial joins are only used to find the instances of 2-length patterns only, and afterwards no special techniques are used to prune the space using the distribution of the feature instances in space. In this paper, we build on the work of [18, 4] and propose an efficient technique that integrates the computation of spatial neighbor relationships with the mining algorithm.

3. EFFICIENT MINING OF SPATIAL COLLOCATIONS

Before we describe our methodology, let us briefly discuss the requirements for an efficient spatial collocations mining tool that operates on a SDBMS:

- *It should be able to operate on large datasets.* Modern SDBMS can potentially store huge amounts of information. Clearly, mining tools that operate on main memory data only are insufficient.

- *It should perform mining fast.* Computational techniques for deriving the patterns and their essential interestingness measures should be optimized.

- *It should take advantage by spatial reasoning and spatial query processing techniques to optimize search as*

much as possible. Techniques from Spatial Databases and Computational Geometry should be utilized wherever appropriate to improve mining efficiency.

- *It should be able to operate on data that are not indexed*. The input of a mining tool is a number of object-sets with ad-hoc features. As discussed in [5], we may apply some operations on the database first to extract the features to be mined. Thus the object-sets may not always be supported by spatial indexes (e.g., R-trees [3]). For example, if we want to mine relationships between expensive houses and luxurious beaches, we may not expect that there is an exclusive index for expensive houses (even though there could be an index for houses of any type). As another example, consider raw data (e.g., mobile user requests) that are produced on-the-fly and there are no pre-existing indexes for them. As argued in [12], it might be more beneficial to operate on raw spatial data, than building indexes especially for the operations.

- *It should be able to operate on one or multiple (spatial) relations*. Going back to our previous examples, we may want to mine data from more than one spatial relations (e.g., one storing houses and one storing beaches), or a single relation (e.g., one storing requests), after classifying its contents into multiple sets; one for each feature.

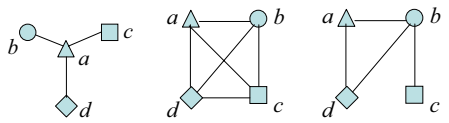
Our proposed mining tool is designed to meet the above requirements. In addition, it is appropriate for mining collocation patterns based on prevalence and/or confidence and deriving the rules from them. Last but not least, it can discover patterns and rules based on both reference feature and collocation models, discussed in Sections 2.1 and 2.2. The next paragraphs describe our approach in detail.

3.1 Representing patterns as graphs

The model described and used in [18, 4, 11] considers only collocation patterns, in each valid instance of which *all pairs* of objects should satisfy some spatial relationship (e.g., they should be close to each other). The same requirement should hold for the patterns mined in [10]. On the other hand, the data analyst may be interested in patterns with arbitrary or no relationships between some feature pairs. In general, we can represent a pattern by a graph, where each node represents a *variable* labeled by a feature and each edge represents the spatial relationship that should hold between the corresponding variables in valid pattern instances.

Figure 2 shows examples of a *star* pattern, a *clique* pattern and a generic one. A variable labeled with feature f_i is only allowed to take instances of that feature as values. Variable pairs that should satisfy a spatial relationship (i.e., constraint) in a valid pattern instance are linked by an edge. In the representations of Figure 2, we assume that there is a single constraint type (e.g., close to), however in the general case, any spatial relationship could label each edge. Moreover, in the general case, a feature can label more than two variables. Patterns with more than one variables of the same label can be used to describe *spatial autocorrelations* on a map.

By definition, the instances of a k-length pattern are the results of a *multi-way* spatial join that combines k spatial datasets (indicated by the labels of the variables) using the



(a) star (b) clique (c) generic

Figure 2: Three pattern representations

spatial relationships described by the graph edges [7]. For example, the instances of the pattern in Figure 2a can be described by the following extended SQL statement, assuming that the instances of feature f_i are stored in relation R_{f_i}:

SELECT R_a.id, R_b.id, R_c.id, R_d.id
FROM R_a, R_b, R_c, R_d
WHERE R_a.location close_to R_b.location
AND R_a.location close_to R_c.location
AND R_a.location close_to R_d.location

Note that this generic definition for spatial patterns, does not affect the definitions of interestingness measures. For example, the participation ratio of a *variable* in a pattern P is the percentage of its instances that participate in instances of P. The participation ratios of pattern variables can be used to derive useful rules, associating the label (i.e., feature) of the variable with the existence of instances of other variables that qualify the specified constraints of the pattern graph. The participation ratio of the variable labeled c in the pattern of Figure 2c captures the probability that the existence of c implies an instance of b in its neighborhood, which forms a clique with an instance of a and an instance of d.

We will initially confine our discussion in patterns that form a star or clique graph and no label constrains more than one variable, because of their high relevance to previous work. Later, we will discuss how our methods can be used to mine more generic patterns.

3.2 Reference feature collocation patterns

We will first focus on methods for mining star-like patterns (like the one in Figure 2a), which have been defined by Koperski and Han [5] (see Section 2.1). In other words we want to find rules that associate the existence of a feature with the existence of other feature instances near it. As an example, consider the rule: "given a pub, there is a restaurant and a snack bar within 100 meters from it with confidence 60%".

Without loss of generality, we assume that the input is n datasets $R_1, R_2, ..., R_n$, such that for each i, R_i stores instances of feature f_i. A dataset R_i may already exist as a spatial relation in the SDBMS (e.g., a relation with cities) or it can be constructed on-the-fly by the analyst (e.g., a relation with large cities, not explicitly stored). Our method imposes a regular grid over the space to mine and *hashes* the objects into partitions using this grid. Given a distance threshold ϵ, each object is extended by ϵ to form a disk and hashed into the partitions intersected by this disk. Figure 3 shows an example. The space is partitioned into 3×3 cells. Object a_1 (which belongs to dataset R_a, corresponding to feature a) is hashed to exactly one partition (corresponding to the central cell C_5). Object b_1 is hashed to two partitions (C_2 and C_5). Finally, object c_1 is hashed into four partitions (C_4,C_5,C_7, and C_8). The reason for extending the objects

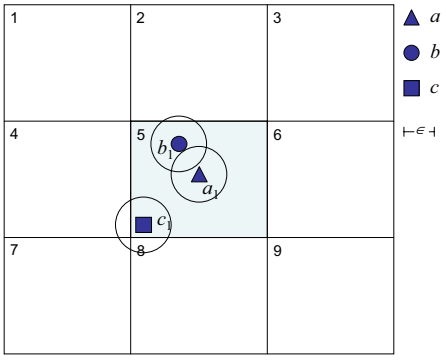


Figure 3: A regular grid and some objects

and hashing them to potentially multiple partitions will be explained shortly. The size of the grid is chosen in a way such that (i) the total number of feature instances (for all features) that are assigned to a grid cell are expected to fit in memory and (ii) the side of a cell is at least 2ϵ long. Due to (i), we can find the patterns using an efficient main memory algorithm for each cell. Due to (ii), no feature instance is assigned to more than four cells (replication is controlled).

Thus the mining algorithm operates in two phases; the hashing phase and the mining phase. During the hashing phase, each dataset R_i is read and the instances of the corresponding feature are partitioned into a number of buckets (as many as the cells of the grid). The mining phase employs a main memory algorithm to efficiently find the association rules in each cell. This method is in fact a multi-way main memory spatial join algorithm based on plane sweep [13, 2, 7]. The sketch of the algorithm is given in Figure 4. The **synch_sweep** procedure extends the plane sweep technique used for pairwise joins to (i) apply for multiple inputs and (ii) find for each instance of one input if there is at least one instance from other inputs close to it.

synch_sweep takes as input a feature f_i (also denoted by the index i) and a set of partitions of all feature instances hashed into the same cell C, and finds the maximal patterns each feature instance is included directly (without computing their sub-patterns first). The objects in the partition R_i^C (corresponding to feature f_i) in cell C are scanned in sorted order of their x-value (lines 3–17). Objects outside the cell are excluded for reasons we will explain shortly. For each object o_i, we initialize the *maximal* star pattern L, where o_i can participate as L's center. Then for each other feature, we sweep a vertical line along x-axis to find if there is any instance (i.e., object) within ϵ distance from o_i; if yes, we add the corresponding feature to L. Finally, L will contain the maximal pattern that includes f_i; for each subset of it we increase the support of the corresponding collocation rule.

The algorithm requires two database scans; one for hashing and one for reading the partitions, performing the spatial joins and counting the pattern supports. If the powerset of all features but f_i cannot fit in memory, we can still apply the algorithm with three database scans instead of two. First, the instances are hashed as usual. Then for each cell C, for each feature f_i: (i) the maximal patterns for all instances of f_i in C are found and stored locally as itemsets, (ii) a itemsets mining method (e.g., [19]) is used to find the locally frequent patterns in cell C and the itemsets are written to a temporary file T_i^C corresponding to f_i and C. Finally, after all cells have been visited, for each feature f_i, the global frequences of patterns which are locally frequent in at least one cell are counted at a single scan of all T_i^C's, and from the globally frequent patterns corresponding spatial collocation rules are generated. Note that the number of local max-patterns for a particular feature f_i and cell C are at most as many as the number of instances of f_i in C, and therefore are guaranteed to fit in memory (and mined there). Finally, note from Figure 3 that an object o_i can be hashed to many cells. However the maximal pattern instance that contains o_i as center will be counted only once, since we peform mining for o_i as center only at the (exactly one) cell, where the object is *inside* (line 5 of **synch_sweep**). However, we still need the replicated instances at the neighbor cells (if any) to assist finding the patterns having other features as center and o_i as neighbor object.

Our algorithm is related to hash-based spatial join techniques [12] and multiway spatial joins [7]. [12] proposes an algorithm that joins two spatial datasets, retrieving the subset of their cartesian product that qualifies a spatial predicate (usually overlap). It hashes the two datasets into buckets using a grid, in the same way as our algorithm and then joins bucket pairs to find the qualifying join pairs. [7] proposes methods that extend binary join algorithms to apply on multiple inputs. In this case, the problem is to find the subset of the Cartesian product of multiple relations, where the instances qualify some constraint graph, like the ones shown in Figure 2. Our method is essentially different since, even though it applies (and joins) multiple datasets, it finds the *maximal patterns* that each instance of a relation qualifies. Thus, the join results in our case may contain fewer feature instances than the total number of features.

3.2.1 Optimizing the multi-way join

The algorithm of Figure 4 may need to perform a large number of computations in order to find the maximal collocation pattern for each object o_i. In the worst case, we may need to compute the distance between o_i and *every* instance of every other feature f_j, $f_j \neq f_i$, in cell C. In order to decrease this cost (exponential to the number of features), we propose the following heuristic. For a given cell C, before processing the join, we perform a secondary spatial hashing in memory, this time using a *fine* grid $\mathcal{F}$; we divide C into smaller cells with $\delta = \epsilon/\sqrt{2}$ length side. Then the objects of all buckets R_i^C (for every feature f_i) are hashed into smaller buckets in memory, this time without replication; each object goes to exactly one *micro-cell* that includes it. Figure 5a shows an example of a cell C, where a number of instances of feature a have been hashed. C corresponds to the shaded region. Observe that the bucket R_a^C also contains two instances which are outside C (but they are within distance ϵ from it). The area defined by C extended by ϵ at all sides is divided into smaller micro-cells as shown in Figure 5.[2] After partitioning the instances using the micro-cells, we know the number of objects of R_a in each of them, as indicated by the numbers in the figure. Figure 5b shows partition R_b^C, corresponding to another feature b, but to the same cell C.

While trying to find patterns that are centered with an a in C, recall that we need to see for every instance of a, which

[2]For the ease of discussion, in this example we assume that the side of each cell C is a multiple of δ; this technique is still applicable in the general case.

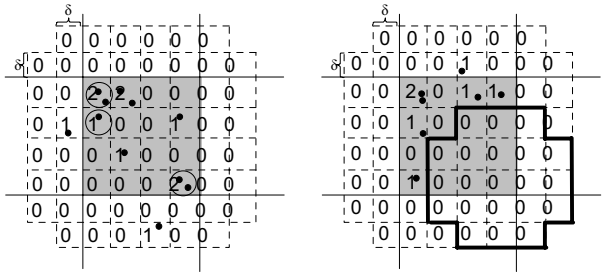
```
/* R_i stores the coordinates of all objects with feature f_i */
Algorithm find_centric_collocations(R_1, R_2, ..., R_n)
1.   /* 1. Spatial-hashing phase */
2.   super-impose a regular grid G over the map;
3.   for each feature f_i
4.      hash the objects from R_i to a number of buckets:
5.         each bucket corresponds to a grid cell;
6.         each object o is hashed to the cell(s) intersected by
7.            the disk centered at o with radius ε;
8.   /* 2. Mining phase */
9.   for each cell C of the grid;
10.     for each feature f_i
11.        load bucket R_i^C in memory;
12.        /* R_i^C containing objects in R_i hashed into C */
13.        sort points of R_i^C according to their x co-ordinate;
14.     for each feature f_i
15.        synch_sweep(f_i, R_1^C, R_2^C, ..., R_n^C);

function synch_sweep(feature f_i, buckets R_1^C, R_2^C, ..., R_n^C)
1.   for each 1 ≤ j ≤ n, j ≠ i
2.      a_j = 1; /* pointer to the first object in R_j^C */
3.   while there are more objects in R_i^C
4.      o_i := next object in R_i^C;
5.      if o_i is in C then /* exclude objects outside C */
6.         L := ∅; /* initialize an empty feature-set */
7.         for each 1 ≤ j ≤ n, j ≠ i
8.            while a_j ≤ |R_j^C| and R_j^C[a_j].x < o_i − ε
9.               a_j := a_j + 1;
10.           p := a_j; /* start moving a pointer from a_j */
11.           while p ≤ |R_j^C| and R_j^C[p].x ≤ o_i + ε then
12.              /* o_i is x-close to R_j^C[p] */
13.              if dist(o_i, R_j^C[p]) ≤ ε then
14.                 L := L ∪ f_j; /* found feature f_j near o_i */
15.                 j := j + 1; go to line 8;
16.        for each I ⊆ L
17.           conf(f_i ⇒ I) := conf(f_i ⇒ I) + 1;
```

Figure 4: An algorithm for reference feature collocations

is in R_a^C and *inside* C, if there are some nearby instances of the other features. By only looking at the numbers of the small cells in Figure 5a and Figure 5b, we can directly know that the objects in the first two circled micro-cells certainly have a b neighbor since the corresponding micro-cell in R_b^C is occupied. Also, we can infer that the objects in the circled micro-cell at the bottom-right corner can have no b neighbor, since this and the surrounding cells (marked by the thick line) are empty in R_b^C. Thus, we can adjust the algorithm of Figure 4 as follows:

- for each object $o_i \in R_i^C$ in a microcell c_x, we can know that it joins with at least those features f_j for which the corresponding microcell c_x is not empty. We need not include these combinations in the **synch_sweep** function (we add extra checks at lines 5 and 7)
- for each object $o_i \in R_i^C$ that is not filtered, we perform the **synch_sweep** join only for those features f_j for which the neighbor cells (up to two neighborhoods) are not empty.

This optimization is expected to be effective when the feature instances are skewed in space. In this case, we can save a lot of computations, by (i) inferring patterns directly (if the corresponding micro-cells are non-empty) and (ii) excluding features (if the corresponding and neighbor micro-cells are empty).



(a) bucket R_a^C (b) bucket R_b^C

Figure 5: Using the finer grid to infer patterns

3.3 Clique collocation patterns

The methodology described in the previous paragraphs can be extended to mine (prevalent or confident) clique collocation patterns. In this case, we are interested in finding patterns like the one in Figure 2b. In instances of such patterns every feature instance is close to every other feature instance.

The algorithm of Figure 4 stops when it finds the features with at least one instance close to the current object o_i. This suffices to discover the *unique* maximal pattern instance centered at o_i. However, when it comes to discovery of cliques the problem becomes more complicated, since there could be multiple maximal cliques that can contain the current object o_i. For example, consider an instance a_1 of feature a which is part of two cliques; $\{a_1, b_1, c_1\}$ and $\{a_1, b_1, d_1\}$; although b_1, c_1, and d_1 they are all close to a_1, they do not form a clique. As another example, consider cliques $\{a_1, b_1, c_1\}$ and $\{a_1, b_2, d_1\}$; in this case, a_1 is in instances of patterns $\{a, b, c\}$ and $\{a, b, d\}$, but the instances of b in them is not common.

The implication is that for each instance o_i (i) we have to find *all* maximal clique patterns it participates in ($\{a, b, c\}$ and $\{a, b, d\}$ for a_1 in the previous example) and (ii) we should not count the subpatterns more than once (we should not count the occurence of a_1 in $\{a, b\}$ twice, although it participates in two super-patterns of $\{a, b\}$).

Our algorithm extends the **synch_sweep** function to compute instances (and prevalences) of clique patterns as follows. For a given o_i, the plane sweep algorithm finds *all* clique pattern instances where o_i participates in. This is performed by checking the distance between the instances of other features that are close to o_i, using a search heuristic based on backtracking [7]. After obtaining all clique instances that contain o_i, we then mark the corresponding patterns and all their subpatterns, such that each distinct pattern where o_i takes part is marked only once. The prevalences and confidences of all those marked patterns are then increased accordingly, before the algorithm proceeds to the next o_i. For example, consider an instance a_1 of feature a which is part of two cliques; $\{a_1, b_1, c_1\}$ and $\{a_1, b_2, d_1\}$; the patterns whose prevalences and confidences will be increased are $\{a, b\}$, $\{a, c\}$, $\{a, d\}$, $\{a, b, c\}$, and $\{a, b, d\}$.

We note here that, since our method attempts to count at one scan the instances of the powerset of all possible patterns, the space required is too high for a large number of

database features (exponential to the total number of features). This makes our approach slow (for clique patterns) when the total number of features is large. However, we observe that in typical applications, although there can be a large number of features only few of them usually participate in long patterns [17]. In order to alleviate the bottleneck of our approach, when there are many features we perform mining in two steps; first we apply our technique to find the prevalences (and/or confidences) of all *pairs of features* only. We then prune those features which do not appear in any prevalent (and/or confident) 2-pattern and reduce the powerset of patterns, which we have to count in our algorithm.

3.4 Generic collocation patterns

Generic collocation patterns associate the existence of features by an arbitrary (connected) graph (like the one of Figure 2c). We could find patterns that form arbitrary graphs, by extending the **synch_sweep** function to find all combinations of feature instances where o_i appears. However, the space of possible patterns is huge in this case. For a given set of features the number of possible graphs that connect them is exponential to the size of the set. Moreover, we have to consider the powerset of the total number of features to consider. For example, if we have 4 features $\{a, b, c, d\}$, we should consider all possible graphs that connect $\{a, b, c, d\}$, all possible graphs that connect $\{a, b, c\}$, etc. Nevertheless our techniques are still applicable when the space of considered graphs is restricted (e.g., patterns where all but one features form a clique). In the future, we plan to investigate the discovery of arbitrary patterns by alternative means (i.e., use of Apriori-based methods).

4. EXPERIMENTAL EVALUATION

In this section, we compare our methods with previous bottom-up Apriori-based approaches [5, 18, 4, 11]. We evaluate the performance of the algorithms using synthetic and real datasets. The algorithms were implemented in C++ and the experiments were run using a 700MHz Pentium III machine with 4Gb of main memory. In the next subsection, we describe the synthetic data generator. Section 4.2 compares the algorithm of Section 3.2 with the methods proposed by [5, 18] for reference feature collocation patterns (i.e., star patterns). Our algorithm for clique patterns (Section 3.3) is compared with the methods of [18] and [4] in Section 4.3. Last, Section 4.4 compares the performance of the algorithms using a real dataset.

4.1 Synthetic Data Generation

Table 1 summarizes the parameters used in the data generator. Firstly, we set L features, which we call non-noise features and can appear in the longest collocation pattern which is generated. We also set n_noise noise features. The number of points for noise features is $r\_noise \times N$. We assign these points to the noise features uniformly. The remaining points are assigned to non-noise features uniformly. The participation ratio of a feature in the longest pattern which has participation ratio larger than the confidence threshold is $\delta_{max} + \theta$. The number of points N_i, which must appear in the instances of longest pattern of a feature f_i is $(\delta_{max} + \theta) \times \frac{N \times (1 - r\_noise)}{L}$. For other features, the participation ratios are $\delta_{min} + \theta$ and the number of points in the instances of longest pattern is $(\delta_{min} + \theta) \times \frac{N \times (1 - r\_noise)}{L}$.

Parameter	Meaning
L	# features in the longest pattern
m	# confident features in the longest pattern
N	# points on the map
d	# generated longest pattern instances
θ	min. prevalence/confidence threshold
δ_{min}	min. difference between prevalence of longest pattern and θ
δ_{max}	max. difference between prevalence of longest pattern and θ
ϵ	distance threshold
map	the x- and y- extent of the map
n_noise	the number of noise features
r_noise	$\frac{\text{number of points with noise features}}{N}$

Table 1: Data Generation Parameters

We generate instances of the longest pattern as follows. We divide the map using a regular grid of cell-side length ϵ. At first, we generate a point randomly. We use the point as the center and r as the radius to generate points for a feature in the longest pattern around a circle. The coordinates for the $i-th$ point of the feature f_j is $(x_c + r \times \sin \frac{2\pi i}{n_j}, y_c + r \times \cos \frac{2\pi i}{n_j})$, $(n_j = \frac{N_j}{d})$. Because r is in $(0, \frac{\epsilon}{2}]$, we can assign r_s the value $s(\frac{\epsilon}{2L})$, $(1 \leq s \leq L)$. In this way, any point in the cell can participate in an instance of the longest pattern. After selecting the first center point, we mark the cells in the grid which intersect the circle centered at it with radius ϵ, such that no other longest pattern instance can be generated in them. Next, we continue generating pattern instances from random points, whose extended circle does not intersect used cells. After generating pattern instances d times, the process ends. The remaining points of the features which appear in the longest pattern, are generated randomly on the map. Finally, we generate the points of noise features randomly on the map.

The generator described above generates instances of a long pattern with length L. The number of features which have participation ratios larger the confidence threshold in the longest pattern is set to m.

4.2 Mining star patterns

First, we experimented with the discovery of star patterns. We compare the performance of the algorithm of Section 3.2 (called FC for "fast collocations") with the methods proposed by [5, 18] that (i) generate 2-patterns using binary spatial joins (ii) perform level-wise mining [1] to discover the patterns from the instances of 2-patterns (called LW for "level-wise" approach). For this set of experiments, we used synthetic datasets.

First, we study the effect of the number of points in a dataset generated using a square map 7070×7070. Table 2 shows the standard generation parameter values used in this set of experiments. Note that we use a rather small dataset size (30K), because the level-wise mining methods are quite slow and it would be quite hard to compare our technique with them for larger databases. Because the constraints in star patterns are quite loose compared to those of clique patterns, the number of patterns and their instances is very large. This causes the Apriori-like algorithm to be very slow as shown in Figure 6. However, our algorithm is scalable, since the long patterns are discovered directly, without going through the discovery (and TID-join) of their subpatterns.

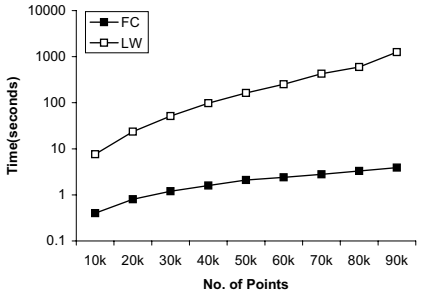


Figure 6: Effect of N (star)

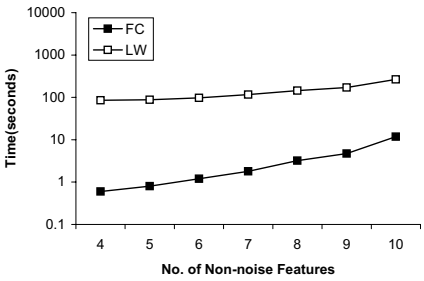


Figure 7: Effect of L (star)

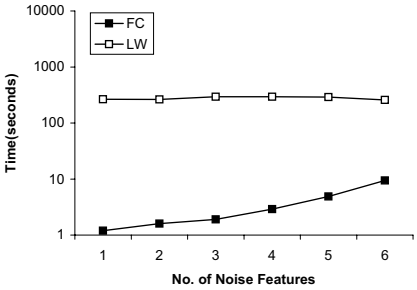


Figure 8: Effect of n_noise (star)

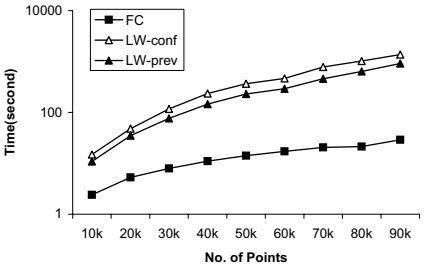


Figure 9: Effect of N (clique)

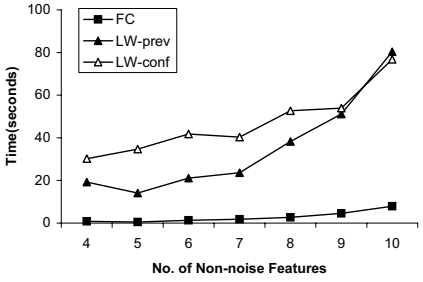


Figure 10: Effect of L (clique)

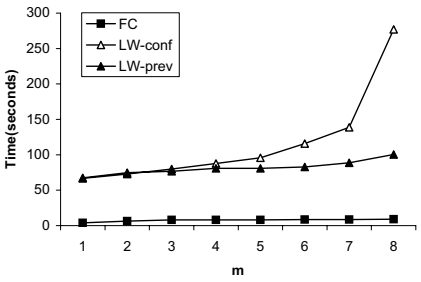


Figure 11: Effect of m (clique)

Parameter	value
N	30000
L	6
m	3
d	1500
θ	0.3
δ_{min}	-0.23
δ_{max}	0.08
ϵ	10
n_noise	2
r_noise	0.1

Table 2: Standard parameter values for star pattern mining

Parameter	Value
N	30000
L	10
m	3
d	1000
θ	0.2
δ_{min}	-0.13
δ_{max}	0.08
ϵ	10
n_noise	2
r_noise	0.1

Table 3: Standard parameter values for clique pattern mining

In the next experiment, we compare the performance of the two techniques as a function of the number of non-noise features in the datasets. Figure 7 shows that our algorithm maintains great advantage compared to the Apriori-like technique.

In the previous experiments, we used only 2 noise features. When the number of noise features increases, the number of 2-patterns increases since more combinations of features are likely to be found frequently together. Figure 8 shows that the time for Apriori-like mining is not affected by the change of this parameter. The cost of our technique increases slightly, however, it is still much faster.

4.3 Mining clique patterns

In this section, we validate the performance of our algorithm in mining clique patterns, by comparing it with the previous approaches proposed in [18] and [4].

The effect of number of points in the dataset was evaluated with datasets generated on a square map 8000×8000. Table 3 shows the generator's parameters for this set of experiments. Figure 9 shows the results. LW-prev corresponds to the method that mines prevalent patterns using θ (as described in [18]). LW-conf corresponds to the method that mines confident patterns using θ (as described in [4]). Our method (FC) can mine prevalent and confident patterns at the same cost (and at the same time) since it is not a level-wise technique. As we can see from the figure, it is much faster compared to the previous techniques. As the number of points increases, the number of instances for all patterns is increases greatly and this affects all algorithms; the level-wise methods are affected more by the boost of the pattern instances at the various levels.

Next, we evaluated the performance of the three techniques as a function of the number L of non-noise features. We used the same parameters as the previous experiment, after fixing $N = 30K$. Figure 10 plots the results. Observe that as the number of features increases, with fixed number N of points, the number of points for each feature decreases. On the other hand, the number of candidates increases, which in general affects more the cost of the algorithms. When the effect of the number of candidates dominates the reduction of points per feature, the time for mining

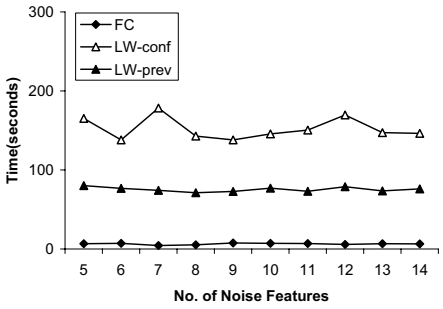


Figure 12: Effect of n_noise (clique)

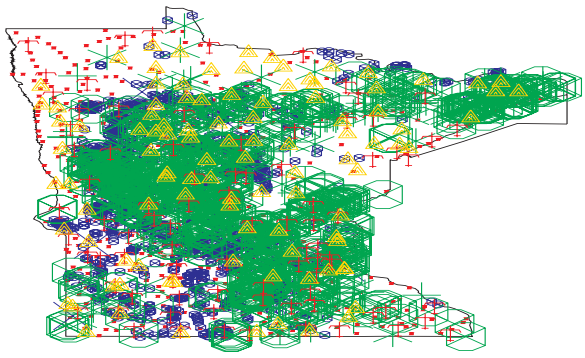


Figure 13: Layers Mapped for Minnesota

increases. Note that in all cases, our method maintains significant advantage over level-wise techniques.

Next, we test the performance effect of the number of m confident features in the longest pattern (i.e., the number of features with participation ratio larger than θ in the longest pattern). As m increases, the number of confident sub-patterns of the longest pattern increases. For the level-wise algorithms, the number of candidates increases as m increases. FC, on the other hand, needs to discover longer maximal patterns. Figure 11 compares the three methods. The x-axis is the number of confident features in the longest pattern. Note that the level-wise methods are more sensitive to m, due to the larger number of candidates to be generated and counted for both prevalent and confident patterns. On the other hand, our technique is almost insensitive to m.

Finally, we test the performance of the algorithms as a function of the number of noise features in the database. Figure 12 illustrates the effects. Observe that our algorithm maintains its advantage over previous techniques, due to the heuristic we apply to remove irrelevant features after a preprocessing step that discovers the prevalent/confident 2-pattens (discussed in Section 3.3).

4.4 Experiments on Real Datasets

We downloaded a real dataset from Digital Chart of the World[3] (DCW), which is an Environmental Systems Research Institute. We downloaded 8 layers of Minnesota state, as shown in Figure 13 and described in Table 4. We treat each layer as a feature so that there are 8 features in our experiments.

Figure 14 compares the mining algorithms for star pattern

[3]http://www.maproom.psu.edu/dcw/

Name	No. of Points
Populated Places	517
Drainage	6
Drainage Supplemental	1338
Hypsography	72
Hypsography Supplemental	687
Land Cover	28
Aeronautical	86
Cultural Landmarks	103

Table 4: Layer Names

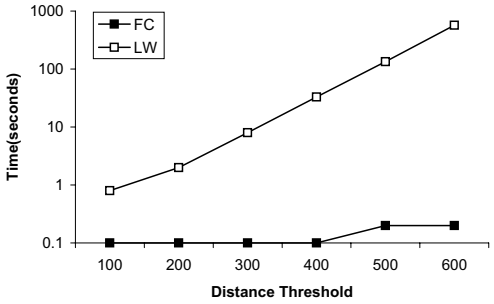


Figure 14: Mining Star Patterns in a Real Dataset

mining. The performance of our algorithm for star patterns is almost not affected by the the distance threshold. On the other hand, the LW approach deteriorates fast with ϵ. Figure 15 shows the performance of algorithms for clique pattern mining. Both figures plot the mining cost as a function of the distance threshold ϵ used for mining. Note that the performance trends are the same compared to the experiments with synthetic data; our method is always much faster compared to the level-wise approaches.

Figure 16 shows the relative performance for clique pattern mining as a function of the prevalence (confidence) threshold θ. Note that our method is very fast even for small values of θ, as opposed to level-wise methods which are very sensitive to it. Notably, the prevalent patterns mining algorithm is also quite fast. This is attributed to the fact that there are very few (and short) prevalent patterns, which are discovered quite fast.

Some long real patterns found include a reference feature pattern with "populated places" as center and "Aeronautical", "Drainage Supplemental", "Hypsography", "Hypsography Supplemental", "Land Cover" as neighbor features;

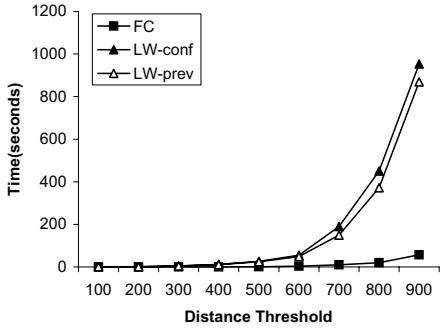


Figure 15: Mining Clique Patterns in a Real Dataset

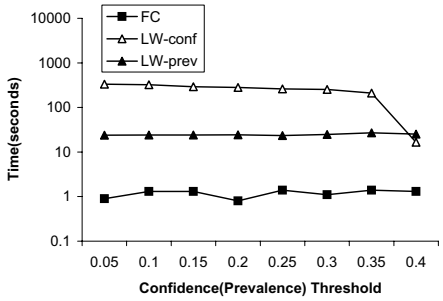


Figure 16: Mining Clique Patterns in a Real Dataset

and a confident clique pattern with "Cultural Landmarks", "Drainage", "Drainage Supplemental", "Hypsography", "Hypsography Supplemental". Both are found for $\epsilon = 600$ and $\theta = 0.05$.

5. CONCLUSION

In this paper, we have extended the collocation pattern model and proposed a fast technique for mining collocation patterns. The extended model generalizes collocation patterns by a constraint graph, where each node corresponds to a feature and the edges correspond to spatial constraints.

As stated in [17] (p. 205), there are typically much fewer features in a spatial mining problem (never more than a few dozens), compared to the number of items in transactional mining problems. Thus the enumeration of spatial neighborhood computations for each feature instance dominates the mining cost. Based on this ground truth, we proposed a mining technique which naturally extends multi-way spatial join algorithms to discover pattern instances of *any length* and directly compute the participation ratios of features in them, which can be used to derive confidences and prevalences of collocation patterns. We proposed a number of heuristics that optimize the mining process and deal with memory constraints.

Finally, we conducted a comprehensive experimental evaluation using synthetic and real datasets. The results show that our technique is orders of magnitude faster in the discovery of long collocation patterns, compared to previous methods. Another notable advantage of our method is that it can compute both prevalent and confident patterns for multiple values of prevalence and confidence thresholds at a single process, since no candidate pruning takes place. Thus, after the join process it suffices to scan the features and their participation ratios in pattern instances in order to derive the prevalences and confidences of all patterns, and finally keep the ones that are more interesting.

Acknowledgments

The authors would like to thank Zhong Zhi for his help with the implementation of the mining algorithm.

6. REFERENCES

[1] R. Agrawal and R. Skrikant. Fast algorithms for mining association rules. In *Proc. of the 20th Int. Conf. on Very Large Data Bases*, pages 487–499, 1994.

[2] T. Brinkhoff, H.-P. Kriegel, and B. Seeger. Efficient processing of spatial joins using r-trees. In *Proc. of ACM SIGMOD Int'l Conference*, 1993.

[3] A. Guttman. R-trees: a dynamical index structure for spatial searching. In *Proc. of ACM SIGMOD Int'l Conference*, 1984.

[4] Y. Huang, H. Xiong, S. Shekhar, and J. Pei. Mining confident co-location rules without a support threshold. In *Proc. of the 18th ACM Symposium on Applied Computing (ACM SAC)*, 2003.

[5] K.Koperski and J.Han. Discovery of spatial association rules in geographic information databases. In *Proc. of the 4th Int. Symp. Advances in Spatial Databases, SSD*, volume 951, pages 47–66, August 1995.

[6] K.Koperski, J.Han, and N.Stefanovic. An efficient two-step method for classification of spatial data. In *Proc. Symp. on Spatial Data Handling (SDH '98)*, 1998.

[7] N. Mamoulis and D. Papadias. Multiway spatial joins. *ACM Trans. Database Syst.*, 26(4), 2001.

[8] M.Ester, A.Frommelt, H.-P.Kriegel, and J.Sander. Algorithms for characterization and trend detection in spatial databases. In *Proc. of the 4th Int. Conf. on Knowledge Discovery and Data Mining*, pages 44–50, 1998.

[9] M.Ester, A.Frommelt, J.Han, and J.Sander. Spatial data mining: Database primitives, algorithms and efficient dbms support. In *Proc. of Int. Conf. on Databases in Office, Engineering and Science*, 1999.

[10] Y. Morimoto. Mining frequent neighboring class sets in spatial databases. In *Proc. of ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining*, 2001.

[11] R. Munro, S. Chawla, and P. Sun. Complex spatial relationships. In *Proc. of the 3rd IEEE International Conference on Data Mining (ICDM)*, 2003.

[12] J. M. Patel and D. J. DeWitt. Partition based spatial-merge join. In *Proc. of ACM SIGMOD Int'l Conference*, 1996.

[13] F. P. Preparata and M. I. Shamos. *Computational geometry: an introduction*. Springer-Verlag New York, Inc., 1985.

[14] R.T.Ng and J.Han. Efficient and effective clustering methods for spatial data mining. In *Proc. of the 20th Int. Conf. on Very Large Data Bases*, pages 144–155, September 1994.

[15] J. Sander, M. Ester, H.-P. Kriegel, and X. Xu. Density-based clustering in spatial databases: A new algorithm and its applications. *Data Mining and Knowledge Discovery*, 2(2):169–194, 1998.

[16] S.Guha, R.Rastogi, and K.Shim. CURE: an efficient clustering algorithm for large databases. In *Proc. of ACM SIGMOD Int. Conf. on Management of Data*, pages 73–84, 1998.

[17] S. Shekhar and S. Chawla. *Spatial Databases: A Tour*. Prentice Hall, 2003.

[18] S. Shekhar and Y. Huang. Discovering spatial co-location patterns: A summary of results. In *Proc of the 7th International Symposium on Advances in Spatial and Temporal Databases (SSTD)*, 2001.

[19] M. J. Zaki and K. Gouda. Fast vertical mining using diffsets. In *Proc. of ACM SIGKDD Conference*, 2003.