Density Based Co-Location Pattern Discovery

Xiangye Xiao, Xing Xie[1], Qiong Luo, Wei-Ying Ma[1]
Department of Computer Science and Engineering, Hong Kong University of Science and Technology
Clear Water Bay, Hong Kong
{xiaoxy, luo}@cse.ust.hk
[1]Microsoft Research Asia, Sigma building, Zhichun Road, Beijing, 100080, P.R. China
[1]{xingx, wyma}@microsoft.com

ABSTRACT

Co-location pattern discovery is to find classes of spatial objects that are frequently located together. For example, if two categories of businesses often locate together, they might be identified as a co-location pattern; if several biologic species frequently live in nearby places, they might be a co-location pattern. Most existing co-location pattern discovery methods are generate-and-test methods, that is, generate candidates, and test each candidate to determine whether it is a co-location pattern. In the test step, we identify instances of a candidate to obtain its prevalence. In general, instance identification is very costly. In order to reduce the computational cost of identifying instances, we propose a density based approach. We divide objects into partitions and identifying instances in dense partitions first. A dynamic upper bound of the prevalence for a candidate is maintained. If the current upper bound becomes less than a threshold, we stop identifying its instances in the remaining partitions. We prove that our approach is complete and correct in finding co-location patterns. Experimental results on real data sets show that our method outperforms a traditional approach.

Categories and Subject Descriptors

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

General Terms

Algorithms, Experimentation, Performance

Keywords

co-location patterns, participation ratio, prevalence

1. INTRODUCTION

With the development of location-based services and the popularity of various mobile devices such as cellular phones, PDAs, and GPS-enabled devices, large amounts of data with spatial contexts have become available. Examples of such data include geographic search logs comprising keywords with associated locations, and GPS logs that maintain latitude, longitude, and velocity information.

Co-location pattern discovery is directed towards processing data with spatial contexts to find classes of spatial objects that are frequently located together. Examples of such spatial objects include businesses listed in yellow page-like databases and queries on geographical search engines. For example, {"shopping mall", "parking"} might be a co-location pattern discovered from logs of geographic search engines. The pattern indicates that the two queries "shopping mall" and "parking" are often searched for nearby locations.

Discovering co-location patterns has many useful applications.

Co-located queries discovered from geographic search logs can help query suggestion, location recommendation, and targeted local advertising. For example, popular hotel chains are good suggestions for a query such as "hotel" based on patterns mined from logs. In addition, different query suggestions may be provided for the same query issued over different locations. For example, a popular casino-themed hotel "MGM Grand" may be returned for people searching for Las Vegas hotels, because "hotel", "MGM Grand" is a co-located query pattern in Las Vegas, whereas "Ilikai hotel" may be suggested for people searching for a hotel in the Hawaii area because "hotel", "Ilikai hotel" is a co-located query pattern in Hawaii.

Co-location patterns discovered from GPS logs can be used for user modeling. We can identify users sharing co-located trajectories as a group of users having similar interests. Then, point of interests can be recommended to a user based on historical trajectories of other users in the same group.

Besides, co-location patterns discovered in biologic and medical data can help people to gain insights into the association between species or diseases in spatial environments.

A traditional approach of mining co-location patterns is proposed by Huang et al. [6]. It first identifies size-2 co-location patterns. Next, the approach generates size-k+1 ($k \geq 2$) candidates and tests the candidates to identify size-k+1 co-location patterns. The test step is as follows. For each size-k+1 candidate, the approach first joins the instances of its two size-k subset co-locations with first $k-1$ common classes to identify its instances. The approach then calculates the prevalence of the candidate based on the identified instances. If the prevalence is no less than a threshold,

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
ACM GIS '08, November 5-7, 2008. Irvine, CA, USA
(c) 2008 ACM ISBN 978-1-60558-323-5/08/11...$5.00.

it is identified as a co-location pattern. Although this approach is correct, the large number of join operations brings heavy computational cost.

In order to reduce the number of join operations, we propose a density based co-location pattern mining algorithm. Different from the previous algorithm [6], our approach does not identify all instances of a candidate. Briefly, it divides the objects into partitions, uses *density* to measure how many instances a partition may have, and processes the dense partitions first to identify instances of candidates. Our approach maintains the upper bounds of the prevalence of candidates using the identified instances in already processed partitions. In the event that the prevalence upper bound of a candidate falls below a threshold, we prune the candidate and do not identifying its instances in the remaining partitions. As a result, the overall number of joins to identify instances is reduced.

We conclude our contributions in this paper as follows:

- First, we propose a new co-location pattern mining approach. It uses density to select areas, in which instances are identified first. Through this method, the number of joins to identify instances is reduced.
- Second, we provide formal proofs about the completeness and the correctness of our approach.
- Third, we experimentally evaluate our approach on real data sets. The results show that our approach outperforms an existing approach on real data.

The remainder of the paper is organized as follows. Section 2 introduces the background of co-location pattern mining and discusses related work. Section 3 describes a motivating example of our approach and Section 4 presents the approach in detail. We prove the correctness and the completeness of our approach in Section 5 and show the experimental evaluation in Section 6. Finally, we conclude the paper in Section 7.

2. BACKGROUND AND RELATED WORK

We first present the basic concepts used in this paper.

In a spatial database, let T be a set of K *classes* $T = \{t_1, t_2, ..., t_K\}$. Let D be a set of *objects* of T, where each *object* is a tuple <object ID, class, location>. Let $\mathcal{R}$ be a *neighbor relationship* over locations of objects. $\mathcal{R}$ is symmetric and reflexive. In this paper, we use the Euclidean distance with a distance threshold ϵ as the neighbor relationship, that is, $\mathcal{R}(o_i, o_j) \Leftrightarrow Euclidean\ distance(o_i, o_j) \leq \epsilon$.

A *co-location* Cl is a subset of classes, $Cl \subseteq T$.

A set I of objects is an *instance* of a co-location Cl, if (1) I contains objects of all classes in Cl and no proper subset of I does so, and (2) $\forall o_i, o_j \in I$, $\mathcal{R}(o_i, o_j)$, that is, o_i and o_j are neighbors.

For example, in a yellow page database, "hotel" is a class. "Sheraton hotel", "'Hilton hotel" and so on are objects of "hotel". Each has an ID and a location.

Figure 1 shows examples of the concepts. In the figure, a point represents an object and a solid line between two points represent the neighbor relationship between two objects. X_i near a point denotes that the point represents the object of class X with the object ID of i. As can be seen in the figure, $\{A_3, B_3, C_2\}$ is an instance of co-location $\{A, B, C\}$ because A_3 is an object of A; B_3 is an object of B; C_2 is an object of C; A_3 and B_3, A_3 and C_2, and B_3 and C_2 are all neighboring object pairs.

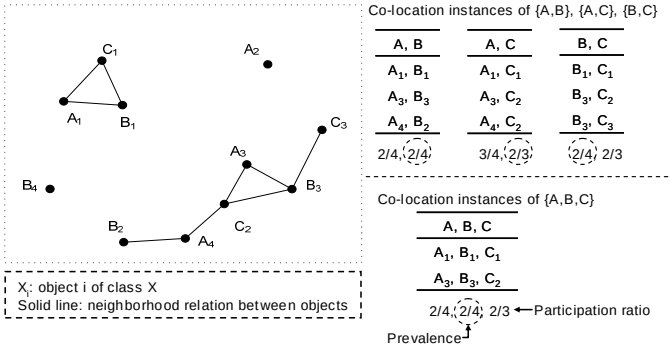


Figure 1: An example of co-location instances.

To characterize how frequently objects of different classes in a co-location are neighbors, we use the *participation ratio* and the *prevalence* defined by Huang et al. [6].

The *participation ratio* of class t_i in a co-location Cl is the fraction of the objects of t_i that participate in co-location instances of Cl. Let $IS(Cl)$ denote the set of co-location instances of Cl and D_i denote the set of objects of t_i. The participation ratio of t_i in Cl can be written as:

$$pr(t_i, Cl) = \frac{|\{o \in D_i | \exists I \in IS(Cl)\ s.t.\ o \in I\}|}{|D_i|} \quad (1)$$

The *prevalence* of a co-location Cl is the minimum participation ratio $pr(t_i, Cl)$ among all classes t_i in Cl:

$$prev(Cl) = \min_{\forall t_i \in Cl} pr(t_i, Cl) \quad (2)$$

A co-location Cl is a *prevalent co-location pattern*, if its prevalence is no less than a given prevalence threshold, that is, $prev(Cl) \geq min\_prev$. In the remaining of this paper, we call a prevalent co-location pattern a *pattern* for short.

For example, in Figure 1, the instances of $\{A, B\}$ are $\{A_1, B_1\}$, $\{A_3, B_3\}$, and $\{A_4, B_2\}$. Thus, the participation ratio $pr(A, \{A, B\})$ of class A in $\{A, B\}$ is $\frac{3}{4}$ because 3 out of 4 objects of A participate in instances of $\{A, B\}$. Similarly, $pr(B, \{A, B\}) = \frac{3}{4}$ because 3 out of 4 objects of B participates in instances of $\{A, B\}$. Therefore, $prev(\{A, B\}) = min(\frac{3}{4}, \frac{3}{4}) = \frac{3}{4}$. Suppose that a prevalence threshold is 50%. $\{A, B\}$ is then a pattern. The right of Figure 1 lists the instances and prevalence scores of size-2 co-locations $\{A, B\}$, $\{A, C\}$, $\{B, C\}$, and size-3 co-location $\{A, B, C\}$. These co-locations are all patterns.

Based on the basic concepts, we formalize the prevalent co-location pattern mining problem as follows:

Input:

1. A set of classes T.
2. A set of objects D. $\forall o \in D$, its class $t \in T$.
3. A neighbor relationship $\mathcal{R}$ over locations of objects. $\mathcal{R}$ is symmetric and reflexive.
4. A prevalence threshold min_prev.

Output:

- All prevalent co-location patterns.

We then review some related work on co-location pattern mining.

Pattern mining based on spatial relations, such as "close to" and "inside", was first proposed by Kopersik and Han [4, 9]. More recently, spatial data mining work on discovering co-location patterns has attracted much attention [5, 6, 10, 14].

Morimoto et al. [10] first defined the problem of finding frequent neighboring co-locations in spatial databases. They used *support*, that is, the number of instances of a co-location, to measure the interestingness of a co-location. Their approach uses a non-overlapping space partitioning scheme and identifies instances of a size-k+1 co-location by grouping instances of a size-k co-location with instances of another class. Their space partitioning approach may miss co-location instances across partitions and therefore may not find all patterns.

Huang et al. [6, 11] proposed a general framework for apriori-gen [1] based co-location pattern mining. They defined the participation ratio, which is more statistically meaningful than support, to measure the interestingness of a co-location. The method is correct and complete, but not very efficient when the data set is large due to the large number of joins. Based on the general approach, Yoo et al. [12, 13] proposed methods that materialize neighborhoods of objects. The joins to identify size-k+1 co-location instances were substituted by scanning the materialized neighborhood tables and looking-up size-k instance tables. The performance of their methods depends on how efficient the scanning and looking-up operations are. In addition, it brings extra memory cost to store the materialized neighborhoods.

In addition to apriori-gen based approaches, there are clustering based co-location pattern discovery methods [2, 3, 8]. The method proposed by Estivill-Castro et al. [2, 3] first clusters the objects of a class and approximates each cluster by a polygon. The method then overlays the polygons of all classes together. The overlapping area of polygons are used to measure how frequently different classes are neighbors. A method proposed by Huang et al. [5] constructs a proximity matrix and applies existing clustering methods to find co-location class clusters. The common problem of the clustering based methods is that the resulting patterns are not fixed. When different clustering methods or parameters are used, the resulting patterns may be different.

There is also work [7, 14] extending the prevalent co-location pattern mining problem. Zhang et al. [14] generalized the co-location mining problem to discover three types of patterns: *star*, *clique*, and *generic* patterns. Their approach is an extension of the approach proposed by Huang et al. [6] to find new types of patterns. Huang et al. [7] extended the problem to mining confident co-location patterns. Different from the minimum participation ratio, the maximum participation ratio (confidence) was used to measure the interestingness of confident co-location patterns. The method they proposed to mine confident patterns is a variant of their method [11] to mine prevalent co-location patterns.

Our method is an apriori-gen based method to discover prevalent co-location patterns. Existing approaches [6, 10, 13] identify all instances of a candidate, whereas our algorithm identifies *false candidates*, i.e., candidates that are not patterns, without identifying all of their instances, thus reduces the computational cost.

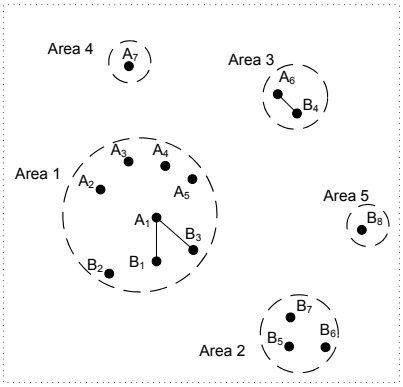
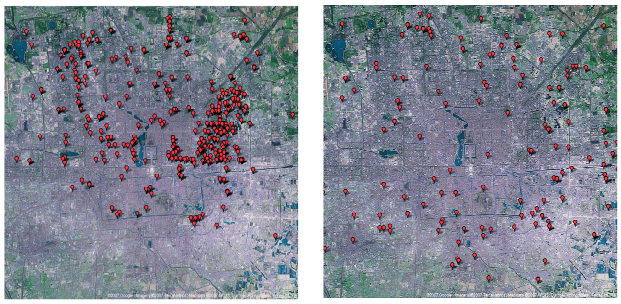


Figure 2: An example data set.



(a) Restaurants. (b) Gas stations.

Figure 3: Location distributions of two classes of businesses in a city.

3. A MOTIVATING EXAMPLE

We provides a motivating example of our approach in this section.

Figure 2 shows an example data set. It includes objects of classes A and B, with neighboring objects connected with solid lines. 50% is used as the prevalence threshold. Then, in order for $\{A, B\}$ to be a co-location pattern, 4 out of 7 objects of A need to have neighboring objects of B and 4 out of 8 objects of B need to have neighboring objects of A.

As can be seen from Figure 2, objects of both classes are densely located in circled Area 1. Therefore, it is likely that neighboring objects of A and B are found there. Excluding the 5 objects of A in Area 1, only 2 objects of A are in the other areas. Thus, checking the neighbor relationship of A and B in Area 1 helps identify early on whether $\{A, B\}$ is a pattern. Consider that in Area 1, only A_1 is found neighboring to objects of B. Learning that there are a total of 7 objects of A, we can then derive that at most 2 objects of A can be neighbors to objects of B right after checking objects of A in Area 1. Since the participation ratio of A in co-location $\{A, B\}$ is at most $\frac{3}{7}$, which is already less than the prevalence threshold, $\{A, B\}$ cannot be a pattern. Therefore, we discard $\{A, B\}$ and do not checking neighbor relationship of $\{A, B\}$ in remaining areas.

As illustrated by this example, our algorithm identifies co-location instances in dense areas first and estimates an upper bound of the prevalence using the instances in checked

areas. When the prevalence upper bound of a candidate is smaller than the threshold, we do not identify its instances in remaining areas. Therefore, it reduces the number of joins.

We design this algorithm based on the observation on real-world data: objects of a class are usually distributed non-uniformly. For example, in a yellow page database, different categories of businesses, such as restaurants and gas stations, tend to be located in different places. Figure 3 displays the locations of restaurants and gas stations in a city. As shown in this figure, restaurants are often located in downtown areas, while gas stations can be found along main roads. Based on the non-uniform distributions of classes in a geographic space, co-location instances may be checked in the more promising areas first in order to decide whether a candidate is a co-location pattern in an earlier stage.

4. DENSITY BASED MINING

Algorithm 1 Density based co-location pattern mining

Input
T, D, $\mathcal{R}$, min_prev
Output
All prevalent co-location patterns
Variables
k: co-location size
C_k: a set of size-k candidates
L_k: a set of size-k patterns
PIS_k: a set of instances of size-k candidates
Method
1: impose a grid over the object space
2: hash objects into partitions
3: $L_1 = T, k = 1$
4: **while** L_k not empty **do**
5: generate size-k+1 candidates C_{k+1} generate
6: identify size-k+1 patterns L_{k+1} based on density test
7: $k = k + 1$
8: **return** $\cup(L_2, ..., L_k)$;

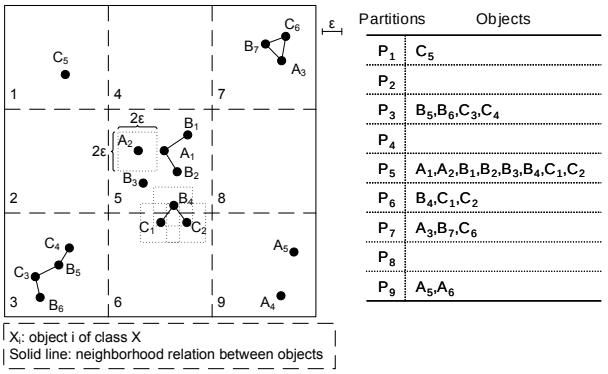
Algorithm 1 describes our density based co-location mining algorithm.

In the first step, our approach imposes a grid of equal-sized rectangular cells over the object space. We require the side length of a cell to be no less than 2ϵ.

The second step hashes the object set D into partitions using the grid. An object o is hashed into cells that intersect with the square centered at o with a side length 2ϵ. The objects hashed into the same cell form a partition of D. We can see that each partition corresponds to a cell.

Figure 4 shows an example. A grid of 3×3 cells is imposed over the object space. Object A_2 is only hashed into partition P_5, which corresponds to cell 5 because the square centered at A_2 (shown as a dash square) with a side length 2ϵ only overlaps with cell 5. Object B_4 is hashed into partitions P_5 and P_6 because the square centered at B_4 with a side length 2ϵ intersects with cell 5 and cell 6. Similarly, objects C_1 and C_2 are also hashed into partitions P_5 and P_6. After hashing the objects, partition P_5 contains 8 objects: A_1, A_2, B_1, B_2, B_3, B_4, C_1, and C_2. Partition P_6 contains objects B_4, C_1, and C_2. The right table of Figure 4 shows the objects hashed in each partition.

This hashing scheme has two advantages. First, there is no need to read multiple partitions at the same time to identify neighboring objects. Second, because the side length of a cell is no less than 2ϵ, an object is hashed into at most four partitions, whereby the duplication of the object set D is controlled.



Partitions	Objects
P_1	C_5
P_2	
P_3	B_5, B_6, C_3, C_4
P_4	
P_5	$A_1, A_2, B_1, B_2, B_3, B_4, C_1, C_2$
P_6	B_4, C_1, C_2
P_7	A_3, B_7, C_6
P_8	
P_9	A_5, A_6

Figure 4: Grid partitioning of objects.

After all objects are hashed, the algorithm iteratively identifies the set of size-k+1 ($k \geq 1$) patterns L_{k+1}. It consists of two steps. The first step generates candidates of the size-k+1 co-location patterns C_{k+1}. The second step identifies the set L_{k+1} of size-k+1 patterns in C_{k+1}. The process stops when L_k is empty. We give a detailed description of the two steps in the following sub-sections.

4.1 Candidate Generation

When $k = 1$, size-2 candidates are all pairs of classes. For example, in Figure 4, size-2 candidates are $\{A, B\}$, $\{A, C\}$, and $\{B, C\}$.

When $k \geq 2$, candidate generation uses apriori-gen to generate the set of size-k+1 candidates C_{k+1} from the size-k patterns L_k. It consists of two steps. In the first step, two size-k patterns in L_k with exactly $k-1$ common classes are joined to generate a size-k+1 candidate. In the second step, if any size-k subset of a candidate is not a pattern in L_k, the candidate is pruned immediately. The correctness of this pruning is based on a result [6] that, given two co-locations Cl and Cl', if $Cl \subseteq Cl'$, $prev(Cl) \geq prev(Cl')$.

For example, assume that $\{A, B\}$ and $\{A, C\}$ are size-2 patterns but $\{B, C\}$ is not. Then, $\{A, B\}$ and $\{A, C\}$ are joined to generate a size-3 candidate $\{A, B, C\}$. Since $\{B, C\}$ is not a pattern, $\{A, B, C\}$ cannot be a pattern and is therefore pruned.

4.2 Pattern Identification

We first describe our definition of *density*, then present the density based pattern identification step.

4.2.1 Density

An instance of a co-location is *in a partition* if all of its objects are hashed into the partition.

The *density* of size-k instances of a partition P is the summation of the ratios of the number of instances of each size-k pattern Cl in P to the total number of instances of Cl. Recall that $IS(Cl)$ denotes the set of instances of a size-k pattern $Cl, Cl \in L_k$. Let $IS(Cl, P)$ denote the set of instances of Cl in P. Then, the density of size-k instances of P can be written as:

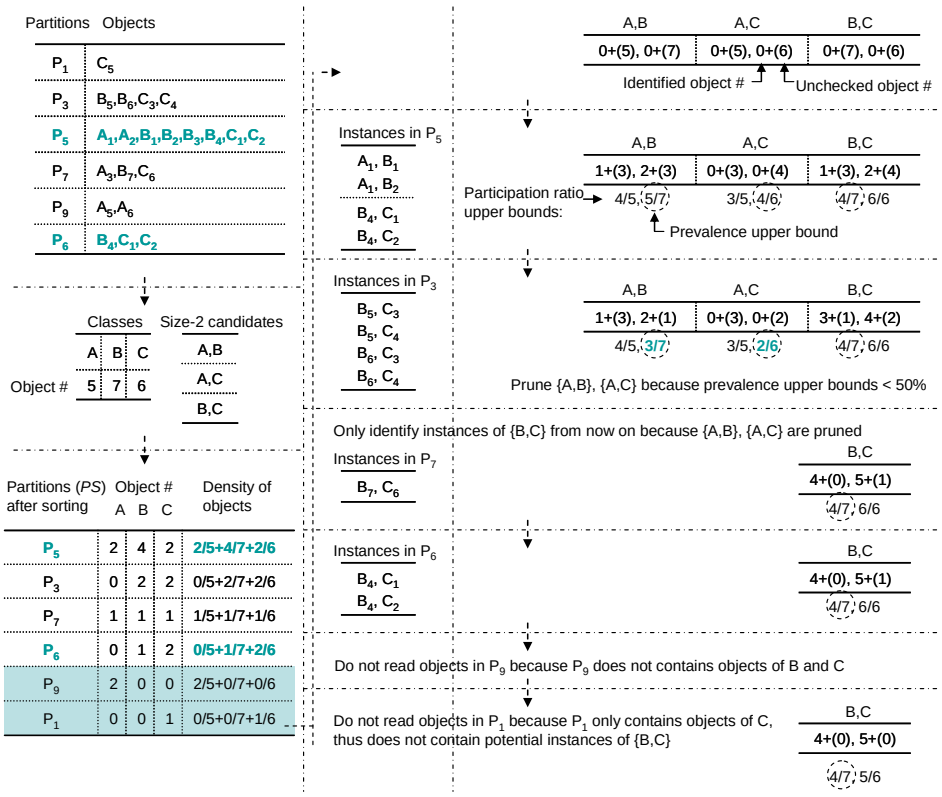


Figure 5: Flow of the density based co-location mining algorithm on the example data set in Figure 4.

$$d(P,k) = \sum_{\forall Cl \in L_k} \frac{|IS(Cl,P)|}{|IS(Cl)|} \tag{3}$$

When $k = 1$, the density of a partition is simply the summation of the ratios of the number of objects of each class t in P to the total number of objects of t in D. Recall that D_i denote the number of objects of t_i, for $t_i \in T$. Let $D_i(P)$ denote the set of objects of t_i in P. Then, the density of objects (size-1 instances) of P can be also written as:

$$d(P,1) = \sum_{\forall t_i \in T} \frac{|D_i(P)|}{|D_i|} \tag{4}$$

Density of size-k instances of a partition measures how many size-k+1 instances are likely to be found in the partition. The denser a partition is, the more instances are likely to be found in the partition.

We first give an example of the density of objects. In Figure 4, A, B, and C have 5, 7, and 6 objects in total, respectively. Partition P_5 contains A_1, A_2, B_1, B_2, B_3, B_4, C_1, and C_2. Thus, its density of objects is $\frac{2}{5} + \frac{4}{7} + \frac{1}{6}$. Partition P_6 contains objects B_4, C_1, and C_2. So, its density of objects is $\frac{1}{7} + \frac{2}{6}$. The left bottom table of Figure 5 shows the density of objects of all partitions.

We then show an example of the density of size-2 instances. Assume that $\{B, C\}$ is a pattern but $\{A, B\}$ and $\{A, C\}$ are not. In Figure 4, among all seven instances of $\{B, C\}$: $\{B_4, C_1\}$, $\{B_4, C_2\}$, $\{B_5, C_3\}$, $\{B_5, C_4\}$, $\{B_6, C_3\}$, $\{B_6, C_4\}$, and $\{B_7, C_6\}$, P_5 contains two2 of them: $\{B_4, C_1\}$ and $\{B_4, C_2\}$. Thus, the density of size-2 instances of P_5 is $\frac{2}{7}$. Partition P_5 also contains instances $\{A_1, B_1\}$ and $\{A_1, B_2\}$. However, $\{A, B\}$ is not a pattern. Therefore, these 2 instances are not taken into account in the density of size-2 instances of P_5.

4.2.2 *Density Based Pattern Identification*

Algorithm 2 shows the density based pattern identification algorithm. Figure 5 illustrates the flow of the algorithm on the example data set in Figure 4. The prevalence threshold is 50%.

In the first two steps, the algorithm calculates the densities of size-k instances of all partitions and sorts the sequence of partitions PS by descending density.

The third step initializes the the number of *unchecked objects* and the number of *identified objects* of each class in each candidate.

Unchecked objects D_i^{unchec} of t_i refer to the objects of t_i that are not in checked partitions. Let PS^{chec} denote the set of checked partitions. Then, D_i^{unchec} can be written as follows. We initialize the number of unchecked objects of each class to the number of its objects.

$$D_i^{unchec} = D_i - \{o \in D_i | \exists P \in PS^{chec} \ s.t. \ o \in P\} \tag{5}$$

Identified objects $D_i^{iden}(Cl)$ of t_i in Cl refer to the objects of t_i that participate in identified instances of Cl in checked partitions. $D_i^{iden}(Cl)$ can be written as follows. We initialize the number of identified objects of each class in each candidate to 0.

Algorithm 2 Density based Pattern Identification

Input
C_{k+1}, PIS_k, $\mathcal{R}$, min_prev
Output
L_{k+1}, PIS_{k+1}
Variables
PS: a sequence of partitions
N^{unchec}: number of unchecked objects of each class
N^{iden}: number of identified objects of each class in each candidate
Method
1: calculate density of size-k instances of partitions
2: sort partitions PS by descending density of size-k instances
3: initialize N^{unchec} and N^{iden}
4: **while** C_{k+1} not empty **and** more partitions in PS **do**
5: P = next partition in PS
6: **if** P contains objects of classes in C_{k+1} **then**
7: **if** P contains potential instances of C_{k+1} **then**
8: read size-k instances $PIS_k(P)$ in P
9: identify size-k+1 instances of C_{k+1} in P
10: update N^{unchec} and N^{iden}
11: update $\overline{prev}$ of each candidate in C_{k+1}
12: remove candidates from C_{k+1} with $\overline{prev} < min\_prev$
13: $L_{k+1} = C_{k+1}$

$$D_i^{iden}(Cl) = \{o \in D_i | \exists P \in PS^{chec}\ \exists I \in IS(Cl, P)\ s.t.\ o \in I\} \quad (6)$$

In the remaining steps, the algorithm identifies instances of C_{k+1} in partitions in the order of them in PS. The loop will stop when C_{k+1} is empty or there is no more partitions in PS.

Let PIS_k denote the set of instances of size-k candidates and let $PIS_k(P)$ denote the set of instances of size-k candidates in partition P.

When the algorithm comes to a partition P in PS, it checks whether P contains objects of at least one class in C_{k+1} in Step 6. If yes, we moves to Step 7. Otherwise, a new loop begins.

A partition contains potential instances of a size-k+1 colocation Cl, if it contains instances of all size-k subset colocations of Cl. For example, suppose that a partition contains instances of $\{A, B\}$, $\{A, C\}$, and $\{B, D\}$ and size-3 candidates include $\{A, B, C\}$ and $\{A, B, D\}$. Since the partition does not contain instances of $\{B, C\}$, it does not contain instances of $\{A, B, C\}$. Similarly, it can not contain instances of $\{A, B, D\}$ either.

Step 7 checks whether P contains potential instances of C_{k+1}. It yes, it reads $PIS_k(P)$ and identifies instances of C_{k+1}. Otherwise, there is no need to read $PIS_k(P)$, we jump to Step 10.

The instance identification step (Step 9) works as follows. For each size-k+1 candidate Cl that has potential instances in P, the algorithm joins instances of two size-k subset patterns Cl_1 and Cl_2 with $k-1$ common classes. Two instances of Cl_1 and Cl_2 having the same objects of the $k-1$ common classes and neighboring objects of the two different classes are joined to identify an instance of Cl.

Step 10 updates N^{unchec} and N^{iden} and Step 11 uses the updated N^{unchec} and N^{iden} to calculate a prevalence upper bound of each candidate in C_{k+1}. We calculate a participation ratio upper bound $\overline{pr(t_i, Cl)}$ and a prevalence upper bound $\overline{prev(Cl)}$ using Equation 7 and 8.

$$\overline{pr(t_i, Cl)} = \frac{|D_i^{unchec}| + |D_i^{iden}(Cl)|}{|D_i|}, \quad (7)$$

$$\overline{prev(Cl)} = \min_{\forall t_i \in Cl} \overline{pr(t_i, Cl)} \quad (8)$$

After obtaining prevalence upper bounds of C_{k+1}, we delete candidates with prevalence upper bounds less than min_prev from C_{k+1} in Step 12. Through this step, the algorithm does not identify instances of deleted candidates in the remaining unchecked partitions in the following loops.

As a conclusion, the algorithm reduces the computational cost in two aspects. First, through hashing objects into partitions, instances are also divided into partitions. The algorithm does not read partitions of size-k instances that do not contain potential instances of size-k+1 candidates. As a result, the computational cost of reading and joining instances in such partitions is saved. Second, the algorithm does not identify instances of already deleted candidates in the remaining partitions. Thus, the computational cost of joins to identify these instances is saved.

We show an example flow of Algorithm 2 in Figure 5. The left column shows the partitions of objects and the density of objects of partitions, the middle shows the instances in partitions, and the right shows the numbers of identified objects and unchecked objects of candidates.

The algorithm first calculates densities of objects of the six non-empty partitions and sorts the partitions by density. The resulting sorted partition sequence is P_5, P_3, P_7, P_6, P_9, and P_1. It then initializes the numbers of identified and unchecked objects of $\{A, B\}$, $\{A, C\}$, and $\{B, C\}$. For example, the number of identified objects of A in $\{A, B\}$ is 0 and the unchecked objects of A is 5.

The loop of identifying instances in partitions begins. The first partition is P_5.

As shown in Figure 4, size-2 instances in P_5 include $\{A_1, B_1\}$, $\{A_1, B_2\}$, $\{B_4, C_1\}$, and $\{B_4, C_2\}$. They are listed in the first table in the middle column of Figure 5. There are one object of A and two objects of B in the instances of $\{A, B\}$. So, the numbers of identified objects of A and B in $\{A, B\}$ are set to 1 and 2, respectively. It remains three unchecked objects of A and four unchecked objects of B in the leftover partitions. Using the updated states of $\{A, B\}$, the algorithm changes the the participation ratio upper bounds of A and B in $\{A, B\}$ to $\frac{1+3}{5}$ and $\frac{2+3}{7}$ using Equation 7. So the prevalence upper bound of $\{A, B\}$ is $\min\{\frac{4}{5}, \frac{4}{7}\} = \frac{5}{7}$. It is greater than 50%, so the algorithm does not delete $\{A, B\}$. In a similar way, the algorithm updates the prevalence upper bounds of $\{A, C\}$ and $\{B, C\}$ based on the size-2 instances of $\{A, C\}$ and $\{B, C\}$ in P_5. $\{A, C\}$ and $\{B, C\}$ are not deleted either because their prevalence upper bounds are also greater than 50%.

After P_5, the algorithm comes to P_3 in PS. P_3 contains four size-2 instances: $\{B_5, C_3\}$, $\{B_5, C_4\}$, $\{B_6, C_3\}$, and $\{B_6, C_4\}$ as shown in the second table in the middle column of Figure 4. There is no object of A in P_3. So, the numbers of identified and unchecked objects of A There is no instance of $\{A, B\}$ in P_3. So the number of identified objects of B in $\{A, B\}$ does not change and the number of unchecked objects of B in $\{A, B\}$ is changed from 3 to 1. Consequently, the upper bound of the participation ratio of A in $\{A, B\}$ does not change but that of B in $\{A, B\}$ is up-

dated to $\frac{3}{7}$. Then, the prevalence upper bound of $\{A, B\}$ becomes $\frac{3}{7}$, which is less than the threshold of 50%. So, the algorithm prunes $\{A, B\}$. In a similar, the prevalence upper bounds of $\{A, C\}$ and $\{B, C\}$ are updated based on their instances in P_3. The algorithm also prunes $\{A, C\}$ based on its updated prevalence threshold. After checking P_3, only $\{B, C\}$ remains as a size-2 candidate.

After P_3, the algorithm examines P_7 and P_6 in order. It only identifies instances, and updates the numbers of identified and unchecked objects and the prevalence upper bound of $\{B, C\}$ in the two partitions. We do not identify instances $\{A_3, B_7\}$ and $\{A_3, C_6\}$ in P_7 and $\{B_4, C_1\}$ and $\{B_4, C_2\}$ in P_6 because $\{A, B\}$ and $\{A, C\}$ have been pruned.

The algorithm then moves to P_9. It does not read objects in P_9 because P_9 does not contain objects of either B or C.

Finally, the algorithm comes to P_1. P_1 only contains objects of C, which implies that it does not contain instances of $\{B, C\}$. Therefore, the algorithm does not read objects in P_1. The final prevalence of $\{B, C\}$ is $\frac{4}{7} > 50\%$. So, the algorithm identifies $\{B, C\}$ as a pattern.

In this example, in six non-empty partitions, our approach identifies instances of $\{A, B\}$, $\{A, C\}$, and $\{B, C\}$ in P_3 and P_5. It only identifies instances of $\{B, C\}$ in P_6 and P_7, and does not read objects in P_1 and P_9 at all. In comparison, a traditional approach [6, 11] identifies all instances of $\{A, B\}$, $\{A, C\}$, and $\{B, C\}$. Thus, our approach reduces the number of joins through this example.

5. CORRECTNESS AND COMPLETENESS

In this section, we prove our density based approach is both correct and complete.

Completeness means that the mining algorithm finds all patterns. Correctness means that all co-locations found by the mining algorithm have a prevalence greater than *min_prev*.

LEMMA 1. *Any instance of a co-location is in at least one partition.*

Proof. Suppose on the contrary that there exists at least one co-location instance I that is not in any partition. Then, there exists two objects o_i and o_j in I such that the partitions of o_i are different from the partitions of o_j. According to our hashing scheme, the square centered at o_i with a side length 2ϵ does not intersect with the square centered at o_j with a side length 2ϵ. In this case, the distance of o_i and o_j is greater than ϵ, which contradicts with the definition that each pair of objects in an instance are neighbors. □

LEMMA 2. *Algorithm 1 identifies all instances of a co-location pattern.*

Proof. The apriori-gen based instance identification approach is complete [6]. It ensures that our approach identifies all instances of a co-location pattern in a partition. Since any instance of a co-location is in at least one partition (Lemma 1), our approach identifies all instances of a co-location pattern. □

LEMMA 3. *The prevalence upper bound we define in Equation 8 never underestimates the prevalence of a co-location.*

Proof. We have:

$$\begin{aligned}
pr(t_i, Cl) &= \frac{|\{o \in D_i | \exists I \in IS(Cl) \ s.t. \ o \in I\}|}{|D_i|} \\
&= \frac{|\{o \in D_i | \exists P \in PS \ \exists I \in IS(Cl, P) \ s.t. \ o \in I\}|}{|D_i|} \\
&\leq \frac{|\{o \in D_i | \exists P \in PS^{chec} \ \exists I \in IS(Cl, P) \ s.t. \ o \in I\}|}{|D_i|} \\
&+ \frac{|\{o \in D_i | \exists P \in PS^{chec} \ s.t. \ o \in P\}|}{|D_i|} \\
&= \frac{|D_i^{iden}(Cl)|}{|D_i|} + \frac{|D_i - \{o \in D_i | \exists P \in PS^{chec} \ s.t. \ o \in P\}|}{|D_i|} \\
&= \frac{|D_i^{iden}(Cl)|}{|D_i|} + \frac{|D_i^{unchec}|}{|D_i|} = \overline{pr(t_i, Cl)}
\end{aligned}$$

We then have:

$$\begin{aligned}
prev(Cl) &= \min_{\forall t_i \in Cl} pr(t_i, Cl) \\
&\leq \min_{\forall t_i \in Cl} \overline{pr(t_i, Cl)} = \overline{prev(Cl)}
\end{aligned}$$

□

THEOREM 1. *Algorithm 1 is complete.*

Proof. Apriori-gen based candidate generation is complete [6], that is, C_k contains all size-k patterns. In addition, our algorithm identifies all instances of each candidate(Lemma 2) and the prevalence upper bound we define in Equation 8 never underestimates the true prevalence of a co-location (Lemms 3). So, Algorithm 2 will not prune a pattern from C_k. □

LEMMA 4. *For a co-location pattern Algorithm 1 identifies, its prevalence upper bound we maintain converges to the final prevalence of this co-location.*

Proof. Algorithm 1 identifies all instances of a co-location pattern Cl (Lemma 2). Therefore, when Algorithm 1 terminates,

$$\begin{aligned}
|D_i^{unchec}| &= 0 \\
|D_i^{iden}(Cl)| &= |\{o \in D_i | \exists I \in IS(Cl) \ s.t. \ o \in I\}|
\end{aligned}$$

We then have $\overline{pr(t_i, Cl)} = pr(t_i, Cl)$ when Algorithm 1 finishes. □

THEOREM 2. *Algorithm 1 is correct.*

Proof. For a co-location pattern Algorithm 1 identifies, its prevalence upper bound we maintain converges to the true prevalence of this co-location (Lemma 4), which is greater than the prevalence threshold. Therefore, our algorithm is correct. □

6. EXPERIMENTAL EVALUATION

In this section, we compare the efficiency of our density based approach with a traditional approach proposed by Huang et al. [6]. Different from our algorithm, their method identifies all instances of each candidate. We use *Dense* to denote our algorithm and *General* to denote the latter method.

1	10:03:36	gold coast	-28.068558	153.432632	…
2	11:18:09	restaurant	-23.654176	147.478237	…
3	13:23:21	taronga park zoo	-42.964294	147.333642	…
4	15:21:52	hotel	-38.141655	144.330927	…
5	15:43:21	restaurant	-36.693958	144.066913	…
…					

Figure 6: A piece of a geographic search log.

Table 1: Information about the base data set

Number of classes	500
Number of objects per class	2,000
Total number of objects	600,000

6.1 Experimental Setup

We picked a one-month geographic search log from a commercial geographic search engine to evaluate our algorithm.

Figure 6 shows a piece of the search log. Each line is a log entry, which comprises several fields, such as request time, query keywords, the latitude and longitude of a search location, where the user is located. For example, the first log entry at line 1 in this figure means that a user searched "gold coast" at latitude -28.068558 and longitude 153.432632 at 10:03:36 on the geographic search engine.

We view a distinct query, such as "gold coast" and "hotel", as a class. We view an occurrence of a query in a log entry as an object of the query class. For example, "restaurant" in log entries at line 2 and line 5 in Figure 6 are two objects of class "restaurant". We view the latitude and longitude pair in a log entry as the location of an object. Thus, we can extract an object from each log entry. For example, from the log entry at line 1 in Figure 6, we obtained an object, of which the class is "gold coast" and the location is (-28.068558, 153.432632). Then, a co-location pattern here means a set of queries that are often searched by nearby users.

In order to pick representative queries and their locations, we selected queries each appearing in more than 2000 log entries. From the resulting queries, we randomly picked 300 queries as the class set. For each query in the class set, we randomly picked 2000 of its log entries to extract objects of the query class. Thus, the base data set we used in the experiments contains 300 classes and 600,000 objects in total. The information about the base data set is shown in Table 1. All of our experiments picked classes and objects from the base data set.

We compared *Dense* with *General* in the experiments. Note that various candidate pruning strategies, such as the multi-resolution pruning [6], are applicable to both approaches. So we compared the performance of *Dense* with *General* both without a pruning strategy, denoted by *Dense-NoPrune* and *General-NoPrune* respectively. We also compared the performance of *Dense* and *General* both with the multi-resolution pruning, denoted by *Dense-MultiRes* and *General-MultiRes* respectively.

We used execution time and *speedup* as the performance metrics. *Speedup without pruning* is the execution time of *General-NoPrune* to that of *Dense-NoPrune*. *Speedup with the multi-resolution pruning* is the execution time of *General-MultiRes* to that of *Dense-MultiRes*. We also studied the memory consumption in each iteration of pattern discovery.

Table 2: Default setting of parameters

Distance threshold(ϵ)	0.01 (roughly one kilometer)
min_prev	0.4
Side length of a cell	25ϵ
Number of classes	150
Number of objects per class	500

Table 3: Various settings of parameters under study

Side length of a cell	10ϵ,20ϵ,40ϵ,80ϵ,160ϵ,320ϵ
Number of classes	100,150,200,250,300
Number of objects per class	125,250,500,1000,2000

The default setting of parameters in the experiments is shown in Table 2. The parameters that affect the performance of co-location discovery including the side length of a cell, number of classes, and number objects per class. We studied the performance of *Dense* and *General* when the values of three parameters change. The various settings of the three parameters is shown in Table 3. When studying the effect of one parameter, we used default values of the other two parameters in Table 2.

6.2 Experimental Results

6.2.1 *Effect of the Side Length of a Cell*

We evaluate the effect of the side length of a cell on the efficiency of our algorithm with and without the multi-resolution pruning. Figure 7 shows the result. The two dash curves are the execution time of *Dense-NoPrune* and *General-NoPrune*. The other two solid curves are the execution time of *Dense-MultiRes* and *General-MultiRes*. Note that the side length of a cell is the parameter to control object partitioning in *Dense*. So, the performance of *General-NoPrune* and *General-MultiRes* does not change with this parameter.

Figure 7 indicates that, *Dense* outperforms *General* both with and without the multi-resolution pruning. For example, when the side length of a cell is 40ϵ, the execution time of *Dense-NoPrune* and *General-NoPrune* is 193 and 391 seconds respectively. The speedup of without pruning is 2.0. The execution time of *Dense-MultiRes* and *General-MultiRes* is 67 and 162 seconds when the side length of a

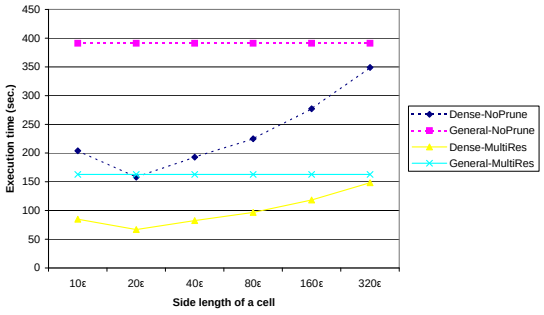


Figure 7: Effect of the side length of a cell on the execution time.

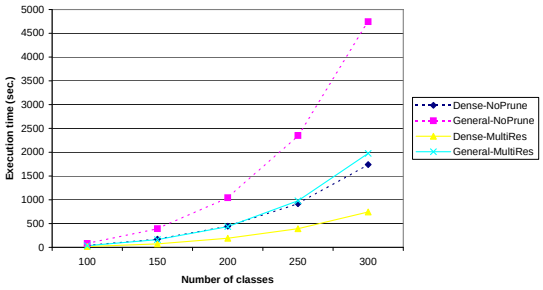


Figure 8: Effect of the number of classes on the execution time.

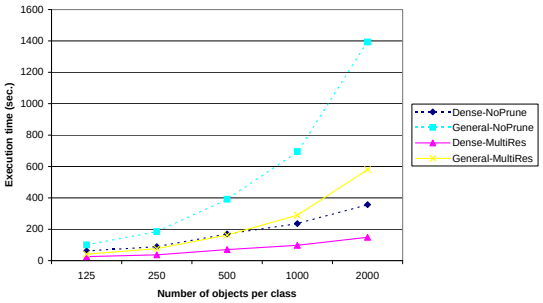


Figure 9: Effect of the number of objects per class on the execution time.

cell is 20ϵ. The speedup with the multi-resolution pruning is 2.4.

As the side length of a cell increases from 10ϵ to 320ϵ, the execution time of *Dense-NoPrune* first reduces from 204 seconds to 159 seconds, then increases to 349 seconds. Correspondingly, the speedup without pruning first raises from 1.9 to 2.5, then drops to 1.1. Similarly, the execution time of *Dense-MultiRes* first reduces from 85 seconds to 67 seconds when the side length of a cell increases from 10ϵ to 20ϵ, then increases to 148 seconds when the side length of a cell increases to 320ϵ. The corresponding speedup with the multi-resolution pruning raises from 1.9 to 2.4, then drops to 1.1.

When the side length of a cell increases, fewer objects near cell boundaries are hashed into multiple partitions. This results in less overhead of identifying duplicated instances in multiple partitions. At the same time, when the side length of a cell increases, the upper bound of the prevalence becomes more coarse and therefore our approach prunes a candidate that is not a pattern latter. The two factors together determines the effect of the side length of a cell on time gain ratio of *Dense*.

As the side length of a cell increases from 10ϵ to 20ϵ, the first factor dominates. Therefore, the speedup increases with the side length of a cell. In comparison, as the side length of a cell increases from 20ϵ to 320ϵ, the second factor dominates, which causes the speedup to drop. When the side length of a cell is very large, such as 320ϵ, the object set is only divided into a very few of partitions. In this case, the prevalence upper bound is very coarse, which results in a low speedup of *Dense*.

6.2.2 Effect of the Number of Classes

We study the effect of the number of classes on the efficiency of *Dense* and *General* with and without the multi-resolution pruning.

The result shown in Figure 8 indicates that, our algorithm is faster than *General* with and without the multi-resolution pruning under various settings of the number of classes. For example, when the number of classes is 200, the execution time of *Dense-NoPrune* and *General-NoPrune* is 444 and 1044 seconds. The speedup without pruning is 2.3. When the number of classes is 100, the execution time of *Dense-MultiRes* and *General-MultiRes* is 18 and 34 seconds. The speedup with the multi-resolution pruning is 1.9.

The execution time of *General* increases faster than that of *Dense* when the number of classes increases. In Figure 8, as the number of classes increases from 100 to 300, the execution time of *Dense-NoPrune* raises from 40 to 1739 seconds and that of *General-NoPrune* raises from 81 to 4744 seconds. The speedup without pruning increases from 2.0 to 2.7. The execution time of *Dense-MultiRes* raises from 18 to 745 seconds and that of *General-MultiRes* raises from 33 to 1976 seconds. The speedup with the multi-resolution pruning raises from 1.9 to 2.7. We can see that the speedup with and without the multi-resolution pruning both increases with the number of classes. It indicates that our approach is more scalable than *General*.

6.2.3 Effect of the Number of Objects per Class

We study the effect of the number of objects per class on the efficiency of *Dense* and *General* with and without the multi-resolution pruning.

The result shown in Figure 9 indicates that our algorithm is faster than *General* both with and without the multi-resolution pruning under various settings of the number of objects per class. For example, the execution time of *Dense-NoPrune* and *General-NoPrune* is 61 and 100 seconds when the number of object per class is 125. The speedup without pruning is 1.6. The execution time of *Dense-MultiRes* and *General-MultiRes* is 235 and 695 seconds when the number of object per class is 1000. The speedup up with the multi-resolution pruning is 2.9.

The execution time of our algorithm increases slower than that of *General* with the number of objects per class increases. In Figure 9, when the number of objects per class increases from 125 to 2000, the execution time of *Dense-NoPrune* raises from 62 to 357 seconds and that of *General-NoPrune* raises from 100 to 1392 seconds. the speedup without pruning raises from 1.6 to 3.9. The execution time of *Dense-MultiRes* raises from 26 to 149 seconds and that of *General-MultiRes* raises from 42 to 580 seconds. the speedup with the multi-resolution pruning increases from 1.5 to 3.7. The raise of the speedup with the number of objects per class indicates a better scalability of our approach.

6.2.4 Investigation of the Speedup Reason

Our approach is faster than *General* because that it identifies partial false candidates without identifying all of their instances.

For example, we show the percentages of false candidates that *Dense-NoPrune* deletes when identifying less than 25%, 50%, 75%, and 99% of their instances. We used the data set with the default setting of parameters in Table 2.

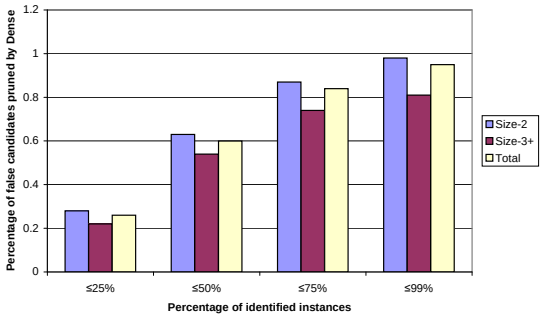


Figure 10: Percentage of false candidates that *Dense* prune when identifying less than 25%, 50%, 75%, and 99% of their instances.

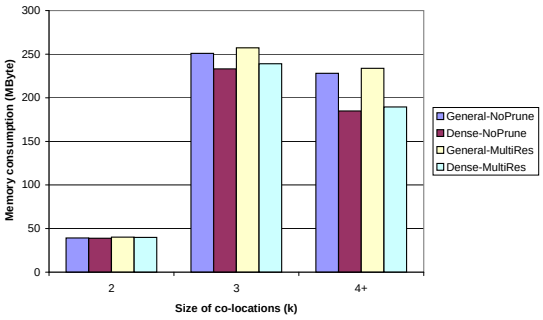


Figure 11: Memory consumption of discovering size-k patterns.

Figure 10 shows that, *Dense-NoPrune* identifies about 63% of the size-2 false candidates when identifying less than 50% of their instances. *Dense-NoPrune* also identifies about 54% of the false candidates of larger sizes without identifying more than half of their instances. Overall, about 62% of the false candidates are identified by *Dense-NoPrune* before half of their instances are identified in this example. It explains the speedup of *Dense-NoPrune* in Figure 7 to Figure 9.

6.2.5 Comparison of Memory Consumption

We report the memory consumption of *Dense-NoPrune*, *General-NoPrune*, *Dense-MultiRes*, and *General-MultiRes* when mining size-k patterns. We used the data set with the default setting of parameters in Table 2.

Figure 11 shows that our approach reduces the memory consumption in each iteration with and without the multi-resolution pruning. When mining size-3 patterns, the memory consumption of *Dense-NoPrune* and *General-NoPrune* is 233 and 251 MBytes. The saving of our approach is 7%. When mining size-4+ patterns, the memory consumption of *Dense-MultiRes* and *General-MultiRes* is 189 and 233 MBytes. The saving of our approach is 18%.

The memory consumption in the iteration of mining size-$k+1$ patterns includes the memory cost of 1) the object set D, 2) instances of size-k patterns PIS_k, 3) instances of size-$k+1$ candidates PIS_{k+1}, 4) size-$k+1$ candidates C_{k+1}, and 5) identified patterns $\cup(L_2, ..., L_{k+1})$. The cost of D, $\cup(L_2, ..., L_{k+1})$, and C_{k+1} is the same in the two approaches. Our approach reduces the memory cost of PIS_k and PIS_{k+1}. The cost of PIS_k is reduced because we delete partitions in PIS_k that do not contain potential instances of size-$k+1$ candidates. The cost of PIS_{k+1} is reduced because we identify partial false size-k candidates without identifying all of their instances.

7. CONCLUSION

In this paper, we propose a density based co-location pattern mining approach. It identifies instances of candidates in dense partitions first and maintains the prevalence upper bounds of candidates. When the prevalence upper bounds of candidates are smaller than the prevalence threshold, our approach prunes them immediately and does not identify their instances in remaining partitions. The experimental results on real data sets show that the density based approach is faster and has less memory consumption than the traditional approach.

8. REFERENCES

[1] R. Agrawal and R. Srikant. Fast algorithms for mining association rules. *Proceedings of VLDB*, pages 487–499, 1994.

[2] V. Estivill-Castro and I. Lee. Data mining techniques for autonomous exploration of large volumes of geo-referenced crime data. *Proceedings of Geocomputation*, pages 24–26, 2001.

[3] V. Estivill-Castro and A. Murray. Discovering associations in spatial data-an efficient medoid based approach. *Proceedings of PAKDD*, pages 110–121, 1998.

[4] J. Han, K. Koperski, and N. Stefanovic. GeoMiner: a system prototype for spatial data mining. *Proceedings of SIGMOD*, pages 553–556, 1997.

[5] Y. Huang, J. Pei, and H. Xiong. Mining co-location patterns with rare events from spatial data sets. *Geoinformatica*, 10(3):239–260, 2006.

[6] Y. Huang, S. Shekhar, and H. Xiong. Discovering colocation patterns from spatial data sets: A general approach. *TKDE*, 16(12):1472–1485, 2004.

[7] Y. Huang, H. Xiong, S. Shekhar, and J. Pei. Mining confident co-location rules without a support threshold. *Proceedings of SAC*, pages 497–501, 2003.

[8] Y. Huang and P. Zhang. On the Relationships Between Clustering and Spatial Co-location Pattern Mining. *Proceedings of ICTAI*, pages 513–522, 2006.

[9] K. Koperski and J. Han. Discovery of spatial association rules in geographic information databases. *Proceedings of SSD*, pages 47–66, 1995.

[10] Y. Morimoto. Mining frequent neighboring class sets in spatial databases. *Proceedings of SIGKDD*, pages 353–358, 2001.

[11] S. Shekhar and Y. Huang. Co-location rules mining: A summary of results. *SSTD*, 2001.

[12] J. S. Yoo and S. Shekhar. A joinless approach for mining spatial colocation patterns. *TKDE*, 18(10):1323–1337, 2006.

[13] J. S. Yoo, S. Shekhar, J. Smith, and J. P. Kumquat. A partial join approach for mining co-location patterns. *Proceedings of GIS*, pages 241–249, 2004.

[14] X. Zhang, N. Mamoulis, D. W. Cheung, and Y. Shou. Fast mining of spatial collocations. *Proceedings of SIGKDD*, pages 384–393, 2004.