Fraction-Score: A New Support Measure for Co-location Pattern Mining

Harry Kai-Ho Chan*, Cheng Long†, Da Yan‡, Raymond Chi-Wing Wong*
*The Hong Kong University of Science and Technology
†Nanyang Technological University
‡The University of Alabama at Birmingham
*{khchanak, raywong}@cse.ust.hk, †c.long@ntu.edu.sg, ‡yanda@uab.edu

***Abstract*—Co-location patterns are well-established on spatial objects with categorical labels, which capture the phenomenon that objects with certain labels are often located in close geographic proximity. Similar to frequent itemsets, co-location patterns are defined based on a support measure which quantifies the popularity (or prevalence) of a pattern candidate (a label set). Quite a few support measures exist for defining co-location patterns and they share an idea of counting the number of instances of a given label set C as its support, where an instance of C is an object set whose objects carry all the labels in C and are located close to one another. Unfortunately, these measures suffer from various weaknesses, e.g., some fail to capture all possible instances while some others overlook the cases when multiple instances overlap. In this paper, we propose a new measure called Fraction-Score whose idea is to count instances *fractionally* if they overlap. Compared to existing measures, Fraction-Score not only captures all possible instances, but also handles the cases where instances overlap appropriately (so that the supports defined are more meaningful and consistent with the desirable anti-monotonicity property). To solve the co-location pattern mining problem based on Fraction-Score, we develop efficient algorithms which are significantly faster than a baseline that adapts the state-of-the-art. We conduct extensive experiments using both real and synthetic datasets, which verified the superiority of Fraction-Score and also the efficiency of our developed algorithms.**

I. Introduction

With the advancement of technologies such as GPS, databases that record objects with both categorical labels and spatial information are prevalent. For instance, in ecology, animals and plants have not only their labels such as their species, but also location information about their habitats; in urban areas, point-of-interests (POIs) such as restaurants and shops have both some labels such as their business types and brands and their locations (e.g., in Google Maps); and in epidemiology, patients are usually recorded with not only demographic information like their jobs, ages and races, but also location information like their home addresses. We say an object is an *instance* of a label if the object carries the label. One interesting pattern on these objects is the *co-location pattern* [20], [22], [14], [13]. A co-location pattern corresponds to a set of labels whose instances are frequently located *in a close geographic proximity* (i.e., the instances are within distance d from each other). As an example, [20] found that Snack Bar shops and Beauty Salon shops are often located near each other, which forms a co-location pattern.

Similar to *frequent itemsets* in the context of transaction data [1], co-location patterns are defined based on a support measure which quantifies for a given label set how frequently those instances of the labels in the label set are located closely, e.g., within distance d from each other. In the context of transaction data, the support of an itemset is defined as the number of transactions that contain all objects in the itemset. Unfortunately, this definition cannot be straightforwardly adapted to our context since there exist no explicit transactions in spatial data.

A. Weaknesses of Existing Support Measures

We say that a set of objects is an *instance* of a label set if the objects carry all labels in the label set and are located within distance d from each other. The challenge of defining the support properly is mainly due to the fact that different instances of a label set usually overlap with each other, and this leads to a dilemma that enumerating all instances would over-count the support while using heuristics would miss some instances completely. In the literature, several support measures for co-location patterns have been proposed, namely *partitioning-based* [22], *construction-based* [20], *enumeration-based* [22], [14], [13], and *participation-based* [22], [14], [13], [33], [32], [31]. The major idea shared by these approaches is to count for a given label set the number of its instances for measuring the support. However, they all suffer from various weaknesses, which we explain as follows.

The partitioning-based approach uses a grid to partition the space into many cells, constructs for each cell a transaction involving all objects within the cell, and then defines supports based on the generated transactions as if they are on conventional transaction data [1]. With this approach, only those instances *within* individual cells are considered, while those *across* cells are missed since two objects within distance d but across cell boundaries are ignored. Consider Figure 1. Suppose the grid as indicated by the dashed lines in the figure is used. The object set {R_3, B_1, C_1} corresponds to an instance of the label set {restaurant, bank, church}, but since the objects are located in different cells, there would be no generated transactions which involve this instance, and thus it is missed.

The construction-based approach constructs instances of a given label set heuristically and counts the number of constructed instances as the support. This approach is not

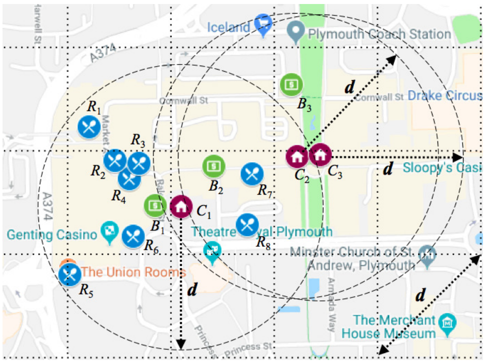


Fig. 1. A small portion of a real dataset of POIs in United Kingdom, including 8 restaurants (blue icons), 3 banks (green icons) and 3 churches (purple icons), where the icons indicate the labels of the spatial objects and the disks have their centers at C_1 - C_3 and radii all equal to d

robust simply because some instances of a label set might be missed due to the heuristic nature. To illustrate, consider Figure 1 and a label set {bank, church}. This approach may construct $\{B_2, C_1\}$ as one instance first (followed by dropping these the two objects from the database), then $\{B_3, C_2\}$ as another one second (followed by dropping the two objects from the database), and then stops since no more instances could be constructed among the remained objects. That is, in total two instances would be constructed by this approach only, while other instances such as $\{B_1, C_1\}$ are missed.

The enumeration-based approach counts for a given label set all its *row instances*, where an instance is said to be a row instance if it does not have a proper subset which is also an instance of the same label set. With this approach, no instances can be missed, but the support definition is not anti-monotonic and counterintuitive. To illustrate, consider Figure 1 again. We consider a label set {bank, restaurant} and one of its subset {bank}. The former label set has at least six row instances (e.g., there are four within the cell containing B_1 and two within the cell containing B_2) while the latter has three row instances only, namely $\{B_1\}$, $\{B_2\}$, and $\{B_3\}$. In this case, the support of a label set is larger than that of its subset, which breaks the anti-monotonicity property that is important both to make sense semantically, and to enable the design of efficient algorithms for frequent pattern mining. The insight into the problem is that this approach may reuse one object in many row instances, and since the object contributes wholly to every row instance that it is involved in, the support is over-measured. Due to this problem, the supports defined by this approach are not used on their own, but as components for defining the confidence of a rule candidate [22], [14], [13].

The participation-based approach is the most commonly used one as it captures all possible instances and its support definition satisfies the anti-monotonicity property. Similar to the enumeration-based approach, the participation-based approach considers all possible row instances, but instead of counting each individual row instance, it puts the row instances into different *groups* and then counts the groups. Specifically, it selects a label and then puts all row instances sharing the same object with the selected label in the same group. The rationale is that *all* row instances within a group are counted as *one* unit of prevalence since they are all based on the same object with a particular label. Nevertheless, in cases where some row instances *across* different groups share an object, this approach would count them as *multiple* units of prevalence (one for each group), i.e., the objects contribution is over-counted.

To illustrate, consider the example in Figure 1. Consider the label set {restaurant, bank, church}. Suppose that the label "restaurant" is the label used for grouping the row instances. There would be eight groups, each based on a restaurant R_1 - R_8. Within each group, all row instances contain the same restaurant. Thus, the support defined by the participation-based approach would be equal to 8. Nevertheless, among these eight groups, many share objects with labels of "bank" and/or "church" (e.g., $\{R_3, B_1, C_1\}$ and $\{R_6, B_1, C_1\}$ are two row instances from two different groups since they contain different restaurants but they share their restaurant and church, i.e., B_1 and C_1). In this case, the prevalence is over-measured.

B. A New Support Measure: Fraction-Score

We propose a new support measure called *Fraction-Score* which considers all possible row instances and at the same time, it avoids the over-counting problem when multiple row instances within a group share the same object (as the participation-based approach does) as well as when multiple row instances *across* groups share objects. The major idea is to count each group as a *fractional* unit of prevalence instead of an *entire* one, where the fraction value is calculated by amortizing the contribution of an object among all the row instances that the object is involved in.

Here, we briefly illustrate how the fraction values are calculated (the detailed definitions will be introduced in Section II). Consider Figure 1 and the label set {restaurant, church}. Suppose that label "restaurant" is the label used for grouping the row instances. In this case, there would be eight groups, formed by R_1 - R_8, respectively. Consider the group formed by R_1. It involves only one row instance, namely $\{R_1, C_1\}$. The fraction associated with the group by R_1 would be set to $1/8$, and the intuition is that it involves an object C_1 and there are 8 groups (or objects involving the label "restaurant", namely R_1 - R_8) that share C_1 and thus, each of the groups (including the one by R_1) would be associated with a fraction $1/8$ (of C_1). Similarly, the fraction associated with each group by R_2 - R_6 would be set to $1/8$. The fraction associated with the group by R_7 would be set to 1 since (1) the row instances in this group, namely $\{R_7, C_1\}$, $\{R_7, C_2\}$, and $\{R_7, C_3\}$, involve three churches, namely C_1, C_2, and C_3; (2) the fractions w.r.t. these objects are $1/8$, $1/2$, and $1/2$, respectively (the fraction $1/8$ of C_1 could be explained as above, the fraction $1/2$ of C_2 (C_3) could be explained by the fact that C_2 (C_3) is shared by two groups, namely those by R_7 and R_8); and (3) the fractions are first aggregated (using a sum function) and then bounded by 1 (using a min function) simply because each group cannot be counted as more than one unit. Similarly, the fraction associated with the group by R_8 is 1.

Then, the sum of fractions, $1/8{\cdot}6{+}1{+}1 = 2.75$, corresponds to the support of {restaurant, church} by Fraction-Score, which is more meaningful than 8 that is the support defined by the participation-based approach, since indeed there are roughly three units of prevalence of the label set (one in left region, one in the top-right region, and one in the middle region which overlaps with the other two).

Besides, as will be shown later, the support defined by Fraction-Score satisfies the desirable anti-monotonicity property.

C. Co-location Pattern/Rule Mining

Based on the new support measure Fraction-Score, we define the confidence of a rule candidate as in the context of transaction data. Then, we define co-location patterns and rules using pre-set parameters of a minimum support and a minimum confidence, respectively. Since Fraction-Score satisfies the anti-monotonicity property, we adopt an Apriori-like algorithm for mining the co-location patterns and rules. One key component of the algorithm is to compute the support of a given label set C, which is not as straightforward in our case as in the transaction data scenario. To compute C's support, we design an algorithm, where a basic operation is to decide whether there exists a row instance of C, which involves a particular object. We show that the decision problem of this operation is NP-hard (w.r.t. $|C|$). In fact, this operation is also necessary when the supports defined by the participation-based approach [22], [14], [13], [33], [32], [31] are computed and is solved by materializing all row instances of C there. Nevertheless, we observe that the complete materialization is an overkill since the operation could be finished by just finding *one* row instance involving the object if there exists one. Besides, we notice that though the decision problem in general is NP-hard, it can be easily solved in certain cases (e.g., if all objects within distance d from an object o do not carry all the labels in C, object o cannot be involved in any row instance of C and thus the answer to the decision problem is clearly "no") and/or with some information re-use strategies (details would be introduced later).

Motivated by these observations, we design a *filtering-and-verification* approach for the decision problem, which performs a few efficient pre-checking procedures (i.e., filtering) for cases where the decision problem could be answered easily, and for those cases where the decision problem cannot be answered in the filtering phase, it performs an exact verification procedure. For the filtering phase, we developed three filters, one based on information re-use and two on pruning. For the verification phase, we developed three methods based on optimization procedures and combinatorial search processes.

D. Main Contributions and Roadmap

In summary, our main contributions in this paper include: (1) we show the weaknesses of existing support measures and propose a new and better one called Fraction-Score, which avoids the weaknesses and also satisfies the desirable anti-monotonicity property; (2) for a fundamental operation involved in mining the co-location patterns and rules, we provide hardness results and design an efficient algorithm which is significantly faster than a baseline adapted from the state-of-the-art; and (3) we conducted extensive experiments on both real and synthetic datasets, which showed the superiority of Fraction-Score as well as the efficiency of the proposed algorithm (e.g., on the real datasets, our algorithm runs faster than the baseline by 1-3 orders of magnitude).

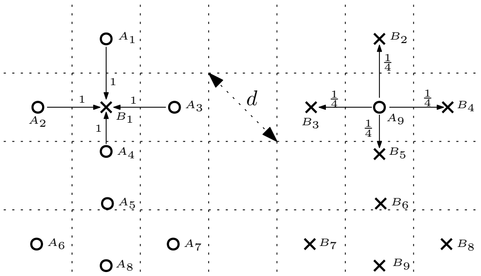


Fig. 2. A toy example where × and ∘ are two labels and A_1-A_9 and B_1-B_9 are eighteen objects each with exactly one label indicated by the shape representing the object.

The rest of the paper is organized as follows. In Section II, we give the formal definition of Fraction-Score and confidence measure and also introduce their properties. In Section III, we adopt an Apriori-like algorithm for mining the co-location patterns and rules and introduce an algorithm for computing the support defined by Fraction-Score. In Section IV, we present the experimental results and in Section V, we review some related work. Finally, in Section VI, we conclude the paper and provide some future research directions.

II. Definitions of Fraction-Score and Problem

We first introduce some notations in Section II-A and then give an overview of Fraction-Score in Section II-B followed by its formal definition in Section II-C, and then define the co-location pattern/rule mining problem in Section II-D.

A. Notations

Let O be a set of n objects. Each object $o \in O$ has a location, denoted by $o.\lambda$, and also a set of (categorical) labels (e.g., a shop brand name such as Starbucks). For ease of presentation, we assume that each object o has only one single label, denoted by $o.t$, but the concepts and algorithms introduced in this paper can easily be applied to the general case by making some duplications of each object with multiple labels, each with one label. Let T be the set of all possible labels of the objects, i.e., $T = \{o.t | o \in O\}$.

Given two objects o and $o^{'}$, we denote the distance between them by $d(o, o^{'})$. Depending on the applications, different metrics such as Euclidean distance and Haversine distance could be used for defining the distance. For ease of illustration, we use Euclidean distance in this paper. Given a set S of objects, we say that S is a *neighbor set* if the maximum pairwise distance within S is bounded by a distance threshold d, i.e., $max_{o,o^{'} \in S} d(o, o^{'}) \le d$. Given an object o and a real number r, we denote by $Disk(o, r)$ the disk with its center at the location of o and its radius equal to r.

TABLE I
NOTATION TABLE

Notation	Definitions
O	the set of spatial objects
o	a spatial object with its location $o.\lambda$ and its label $o.t$ and in Section II, it usually denotes an object that carries the label used for grouping row instances
$o^{'}$	a spatial object with its location $o^{'}.\lambda$ and its label $o^{'}.t$ and in Section II, it usually denotes an object that might be shared by multiple row instances across groups
d	the distance threshold for defining neighbor sets
T	the set of all possible labels of the objects
C	a label set (or a co-location pattern candidate)
t	a label in C and in Section II, it usually denotes the label used for grouping row instances
$t^{'}$	a label in C and in Section II, it usually denotes the label that is carried by an object which may be shared by row instances across groups

Let C be a label set. A set S of objects is said to be an *instance* of C if S is a neighbor set and covers all labels in C (i.e., $C \subseteq \{o.t | o \in S\}$). An instance of C is said to be a *row instance* of C if none of its proper subsets is an instance of C. To illustrate, consider Figure 2. Suppose the label set C is $\{\times, \circ\}$. Then, $\{A_1, B_1\}$ is a row instance of C. But, $\{A_9, B_2, B_4\}$ is not a row instance of C because its proper subset $\{A_9, B_2\}$ is an instance of C.

The main notations that are used throughout the paper are summarized in Table I.

B. Overview of Fraction-Score

Same as the participation-based approach, Fraction-Score groups the row instances of C by the objects with a given label t in C, i.e., all row instances involving the same object with label t are put in the same group. Note that this is always possible since each row instance involves exactly one object with the label t since otherwise, a subset of it will also be a row instance, a contradiction. To solve the over-counting problem when instances across different groups share an object, says $o^{'}$, with a label $t^{'}$ other than t, Fraction-Score assigns to each group among all groups whose row instances share $o^{'}$ a fraction of $o^{'}$, which is equal to 1 divided by the total number of such groups. That is, Fraction-Score splits object $o^{'}$ into some equal fractions and distributes these fractions to the objects based on which the groups of row instances may share $o^{'}$. Note that for each label other than t in C, the object o (and essentially the corresponding group of row instances) may receive multiple fractions since there are multiple objects other than o in the group that might be shared by other groups. We use an appropriate aggregation function on these fractions which gives an aggregated one for the object o (or equivalently the corresponding group) and then sum the (aggregated) fractions of all groups to be the support. We note that for each label t in C, we would have a grouping of the row instances of C and correspondingly a support. To capture the worst-case prevalence, we choose to use the minimum one among all supports as the final support which would then be normalized into $[0, 1]$ by being divided by a constant.

C. Formal Definition of Fraction-Score

We start by defining some concepts related to *fraction*. Let t be the label used for grouping the row instances of C. We denote by $Obj(t, C)$ the set of objects o which has the label t and there are some row instances of C involving o. Conceptually, each object o in $Obj(t, C)$ corresponds to a group of row instances of C (by label t). To illustrate, consider Figure 2. Suppose C is $\{\times, \circ\}$ and $\times$ is used for grouping the row instances of C (we will use this setting as our running example in this section unless otherwise specified). Then, $Obj(\times, C)$ is $\{B_1, B_2, ..., B_5\}$ and each object in $Obj(\times, C)$ corresponds to a group of C's row instances.

Consider an object o in $Obj(t, C)$ and another object $o^{'}$ with its label different from t (i.e., $o^{'}.t \neq t$). If some row instances in the group formed by o involve $o^{'}$, i.e., $o^{'}$ is shared by this group, we know that o must be located in $Disk(o^{'}, d)$ since otherwise o and $o^{'}$ cannot be involved in the same row instance of C. Thus, the potential number of groups that $o^{'}$ could be shared by is bounded by the number of objects which are located in $Disk(o^{'}, d)$ and have the label t. Motivated by this, Fraction-Score splits $o^{'}$ into $|Neigh(o^{'}, t, d)|$ equal fractions each equal to $1/|Neigh(o^{'}, t, d)|$ and then distributes each fraction to an object in $Neigh(o^{'}, t, d)$, where $Neigh(o^{'}, t, d)$ denotes the set of objects which are located in $Disk(o^{'}, d)$ and carry the label t (note that $o \in Neigh(o^{'}, t, d)$). To illustrate, consider Figure 2. We have $Neigh(A_1, \times, d) = \{B_1\}$ and $Neigh(A_9, \times, d) = \{B_2, B_3, B_4, B_5\}$. Thus, a fraction 1 of A_1 is distributed to B_1 and a fraction $1/4$ of A_9 is distributed to each of B_2 - B_5. The intuition here is that A_1 could be shared by 1 group (one with the fraction 1) and A_9 by 4 groups (each with an equal fraction of $1/4$).

Now, we take the perspective of how object o receives fractions of objects located nearby. Specifically, it would receive a fraction of each of those objects $o^{'}$ with $o \in Neigh(o^{'}, t, d)$. Besides, the amount of fraction of an object $o^{'}$ that o receives, denoted by $\Delta_{obj}(o, o^{'})$, is equal to $1/|Neigh(o^{'}, t, d)|$, i.e.,

$$\Delta_{obj}(o, o^{'}) = \frac{1}{|Neigh(o^{'}, t, d)|} \tag{1}$$

Consider the example in Figure 2. We have $\Delta_{obj}(B_1, A_1) = \frac{1}{|Neigh(A_1, \times, d)|} = 1$, which means B_1 receives a fraction 1 of A_1. Similarly, $\Delta_{obj}(B_2, A_9) = \frac{1}{|Neigh(A_9, \times, d)|} = \frac{1}{4}$, which means B_2 receives a fraction $1/4$ of A_9.

Object o may receive fractions from multiple objects, which need to be aggregated. This is achieved in two steps. First, we aggregate the fractions from those objects with the same label using a *sum* function since the fraction of one object could contribute to forming a row instance and that of another object could also contribute to forming another row instance within the same group (i.e., these fractions are complementary to one another for forming row instances). Second, we bound the aggregated fraction for a label by one unit since each group cannot be counted as more than one unit (recall that the row instances within each group share one single object with the label used for grouping the row instances). In summary, the aggregated fraction of objects sharing a label $t' \in C - \{t\}$ that o receives (these objects form the set $Neigh(o, t', d)$), denoted by $\Delta_{label}(o, t')$, is defined as follows.

$$\Delta_{label}(o, t') = \min\{\sum_{o' \in Neigh(o,t',d)} \Delta_{obj}(o, o'), 1\} \quad (2)$$

Consider the example in Figure 2. We have $\Delta_{label}(B_1, \circ) = \min\{4, 1\} = 1$ since B_1 receives 1 of each of A_1 - A_4. Similarly, $\Delta_{label}(B_2, \circ) = \min\{\frac{1}{4}, 1\} = \frac{1}{4}$.

Now, we are ready to introduce the formal definition of Fraction-Score. Instead of materializing all row instances of C and then grouping the row instances by the objects with the label t explicitly as existing studies did [22], [14], [13], we only maintain the grouping conceptually. Recall that $Obj(t, C)$ denotes the set of objects o which have the label t and are involved in some row instances of C. For each object o in $Obj(t, C)$, we aggregate the fractions it receives w.r.t. all labels t' in $C - \{t\}$ using a *min* function since the minimum fraction w.r.t. a label corresponds to the worst-case scenario that one object is shared by multiple groups, and denote the aggregated fraction o receives w.r.t. C by $\Delta_{labelSet}(o, C)$, i.e.,

$$\Delta_{labelSet}(o, C) = \min_{t' \in C - \{t\}} \Delta_{label}(o, t') \quad (3)$$

The above definition is for cases where $|C| \geq 2$, and in the case when C contains one label only, i.e., $|C| = 1$, we simply define $\Delta_{labelSet}(o, C) = 1$. Consider the example in Figure 2. We have $\Delta_{labelSet}(B_1, C) = \Delta_{label}(B_1, \circ) = 1$ and $\Delta_{labelSet}(B_2, C) = \Delta_{label}(B_2, \circ) = \frac{1}{4}$.

We then define the support given the label t for grouping row instances, denoted by $sup(C|t)$, as the sum of the aggregated fractions that the objects in $Obj(t, C)$ receive w.r.t. C, i.e.,

$$sup(C|t) = \sum_{o \in Obj(t,C)} \Delta_{labelSet}(o, C) \quad (4)$$

Consider the example in Figure 2. In this case, we have $Obj(\times, C) = \{B_1, B_2, ..., B_5\}$. Then, $sup(C|\times) = \Delta_{labelSet}(B_1, C) + \Delta_{labelSet}(B_2, C) + ... + \Delta_{labelSet}(B_5, C) = 1 + 4 \cdot \frac{1}{4} = 2$.

Note that depending on different choices of label t, we may have different supports. To capture the worst-case prevalence, we choose the label given which the support is the smallest. Besides, we normalize the support to $[0, 1]$ by dividing it the maximum number of objects that have a specific label in T. In summary, the support of a given label set C, denoted by $sup(C)$, is defined as follows.

$$sup(C) = \frac{\min_{t \in C} sup(C|t)}{\max_{t \in T} |\{o.t = t | o \in O\}|} \quad (5)$$

Consider the example in Figure 2 again. Consider that $C = \{\times, \circ\}$. We have $sup(C) = \frac{\min\{sup(C|\times), sup(C|\circ)\}}{9} = \frac{2}{9}$.

It is worth mentioning that all row instances are captured and counted appropriately by Fraction-Score. Moreover, Fraction-Score satisfies the anti-monotonicity property.

Lemma 1 (Anti-monotonicity property): Given two label sets C' and C, where C' is a subset of C, we have $sup(C') \geq sup(C)$.

Proof 1: The correctness relies on the fact $sup(C'|t) \geq sup(C|t)$ for any t in C' which could be verified using the following facts: (1) $Obj(t, C) \subseteq Obj(t, C')$ for any t and (2) $\Delta_{labelSet}(o, C) \leq \Delta_{labelSet}(o, C')$ for any $o \in Obj(t, C)$ (which is based on the Equation (3) and the fact that $C' \subseteq C$). ■

Suppose C is co-location pattern. Then, "$C' \rightarrow C - C'$" would be a co-location rule candidate, where C' is a subset of C and $C - C'$ means the difference between C and C'. We define the confidence of "$C' \rightarrow C - C'$", denoted by $conf(C' \rightarrow C - C')$, as follows.

$$conf(C' \rightarrow C - C') = \min_{t \in C'} \frac{sup(C|t)}{sup(C'|t)} \quad (6)$$

Consider the example in Figure 2. The confidence of "$\{\circ\} \rightarrow \{\times\}$" is equal to $\frac{2}{9}$ since $sup(\{\circ\}) = \frac{9}{9} = 1$ and $sup(\{\times, \circ\}) = \frac{2}{9}$.

D. Co-location Pattern/Rule Mining Problem

In this paper, we study the problem of finding co-location patterns and rules on a database with spatial objects. Specifically, given a set O of objects, each with a location and a label, a distance threshold d for defining neighbor sets, and two user parameters $min\text{-}sup$ and $min\text{-}conf$, the problem is to find all co-location patterns and rules, where a label set C is a co-location pattern if $sup(C) \geq min\text{-}sup$ and two labels sets C' and C with $C' \subseteq C$ forms a co-location rule "$C' \rightarrow C - C'$" if $conf(C' \rightarrow C - C') \geq min\text{-}conf$.

III. Algorithms for Co-location Pattern/Rule Mining

Since the fraction-based prevalence measure satisfies the anti-monotonicity property (Lemma 1), we design an Apriori-like algorithm for computing all co-location patterns and rules from O. The major idea is to iteratively construct co-location pattern candidates and then verify them in an ascending order of their sizes. Specifically, we use C_k ($k \geq 1$) to denote the set of co-location pattern candidates with the size of k and L_k ($k \geq 1$) the set of co-location patterns with the size of k. The algorithm proceeds iteratively. At the first iteration, it computes C_1 as $\{\{t\} | t \in T\}$ and L_1 as $\{\{t\} | sup(\{t\}) \geq min\text{-}sup, t \in T\}$. At the k^{th} iteration ($k \geq 2$), it generates C_k as $\{L \cup L' | L \in L_{k-1}, L' \in L_{k-1}, |L \cup L'| = k\}$ and L_k as $\{C | C \in C_k, sup(C) \geq min\text{-}sup\}$. Here, C_k is generated by combining any two patterns in L_{k-1} only, and the rationale is that by the anti-monotonicity property, it cannot happen that an object set is in L_k while one of its subsets is not in L_{k-1}. Based on each found co-location pattern C, it then checks each possible candidate of co-location rule in the form of "$C' \rightarrow C - C'$" by enumerating all proper subsets of C as C' and returns it as a co-location rule if its confidence is at least $min\text{-}conf$.

Algorithm 1 FractionComputation(O, T, d)

```
Input: an object set $O$, a label set $T$, a distance threshold $d$
Output: the aggregated fraction each object $o \in O$ receives
    w.r.t. each $t \in T$, i.e., $\Delta_{label}(o, t)$
 1: for all object $o$ in $O$ do
 2:     for all label $t$ in $T$ do
 3:         $|Neigh(o, t, d)| \leftarrow 0$
 4:         $\Delta_{label}(o, t) \leftarrow 0$
 5: for all object $o$ in $O$ do
 6:     for all object $o^{'}$ in $Disk(o, d)$ do
 7:         $|Neigh(o, o^{'}.t, d)|$ += 1
 8:     for all object $o^{'}$ in $Disk(o, d)$ do
 9:         $\Delta_{obj}(o^{'}, o) \leftarrow 1/|Neigh(o, o^{'}.t, d)|$
10:         $\Delta_{label}(o^{'}, o.t)$ += $\Delta_{obj}(o^{'}, o)$
11:         if $\Delta_{label}(o^{'}, o.t) > 1$ then
12:             $\Delta_{label}(o^{'}, o.t) \leftarrow 1$
```

Algorithm 2 SupportComputation(C, O)

```
Input: a label set $C$ and an object set $O$
Output: the support of $C$, i.e., $sup(C)$
 1: $sup(C) \leftarrow \infty$
 2: for all label $t$ in $C$ do
 3:     $sup(C|t) \leftarrow 0$
 4:     for all object $o$ with the label $t$ do
 5:         if there is a row instance of $C$ which involves $o$ then
 6:             $sup(C|t)$ += FractionAggregation($O$, $C$, $o$)
 7:     if $sup(C|t) \leq sup(C)$ then
 8:         $sup(C) \leftarrow sup(C|t)$
 9: Return $sup(C)$
```

Algorithm 3 FractionAggregation(O, C, o)

```
Input: an object set $O$, a label set $C$, and an object $o$ in $O$
Output: the aggregated fraction object $o$ receives w.r.t. $C$,
    i.e., $\Delta_{labelSet}(o, C)$
 1: $\Delta_{labelSet}(o, C) \leftarrow \infty$
 2: for all label $t$ in $C - \{o.t\}$ do
 3:     if $\Delta_{label}(o, t) < \Delta_{labelSet}(o, C)$ then
 4:         $\Delta_{labelSet}(o, C) \leftarrow \Delta_{label}(o, t)$
 5: Return $\Delta_{labelSet}(o, C)$
```

As could be noticed, a key procedure involved in the above Apriori-like algorithm is to compute for a given label set C its support, i.e., $sup(C)$. Different from the case on transaction databases [1], where the procedure could be finished by scanning the transactions once and counting how many transactions involve the label set, this procedure is non-trivial in our scenario. Besides, none of the algorithms proposed for this procedure in existing studies on mining co-location patterns [20], [22], [14], [13] could be used for the procedure based on Fraction-Score: (1) the procedure based on the partitioning-based approach is the same as that on transaction databases and thus it is not applicable, (2) that based on the construction-based approach [20] is far from being applicable here since it is based on some heuristics only and involves no concepts of fraction, and (3) those based on the enumeration-based and participation-based approaches [22], [14], [13] all *materialize* and count all row instances of a given label set while the support by Fraction-Score does not rely on counting row instances of a given label set.

We note here that our main technical focus in this paper is on computing the supports defined by Fraction-Score, which is orthogonal to existing studies aiming for faster and more scalable frequent pattern mining techniques [24], [25]. In fact, these techniques could be easily adapted to our problem since the supports defined by Fraction-Score satisfy the anti-monotonicity property.

A. An Algorithm for Computing the Support Measure

Our algorithm consists of two procedures, namely FractionComputation which collects the information of $\Delta_{label}(o, t)$ for all objects o's and all labels t's and SupportComputation which computes the support of a given label set C based on the information that has been computed by FractionComputation.

FractionComputation. The pseudo-code of FractionComputation is presented in Algorithm 1. First, it initializes $|Neigh(o, t, d)|$ and $\Delta_{label}(o, t)$ for each object $o \in O$ and each label $t \in T$ as 0 (lines 1-4). Second, for each object $o \in O$, it proceeds as follows. It counts the number of objects in $Disk(o, d)$ which have a label t (lines 6-7). Then, it distributes a fraction $1/|Neigh(o, o^{'}.t, d)|$ of o to each object $o^{'}$ in $Disk(o, d)$ (lines 8-9) and updates the fraction $o^{'}$ receives w.r.t. $o.t$ (line 10). Finally, it bounds the fraction an object receives w.r.t. a label by 1 (lines 11-12). A straightforward implementation of this algorithm would occupy $O(|O| \cdot |T|)$ memory for storing the information $\Delta_{label}(o, t)$. For better storage efficiency, we adopt a *maintenance-on-demand* strategy, i.e., only those $\Delta_{label}(o, t)$'s with $t \in \bigcup_{o' \in Disk(o,d)} \{o^{'}.t\}$ are computed. In this way, the memory usage would be much smaller than $O(|O| \cdot |T|)$ since the objects within the neighborhood of an object usually involve not that many labels.

SupportComputation. The pseudo-code of the SupportComputation procedure is presented in Algorithm 2. First, it initializes $sup(C)$ to be infinity (line 1). Then, it tries to use different labels in C for grouping the row instances of C *conceptually* (line 2). For a specific label t, it first initializes $sup(C|t)$ as 0 (line 3) and then adds up for each object o which has the label t and is involved in some row instances of C the fraction it receives w.r.t. C, which is computed by the "FractionAggregation" procedure (whose details are presented in Algorithm 3), as $sup(C|t)$ (lines 4-6). At the end, it returns the smallest $sup(C|t)$ for a label $t \in C$ as $sup(C)$ (lines 7-9).

The "FractionAggregation" procedure, which computes for a object o in O the fraction it receives w.r.t. a label set C, i.e., $\Delta_{labelSet}(o, C)$, is presented in Algorithm 3. First, it initializes the fraction o receives w.r.t. C as ∞ (line 1). Second, for each label t in $C - \{o.t\}$ (line 2), it updates $\Delta_{labelSet}(o, C)$ if $\Delta_{label}(o, t) < \Delta_{labelSet}(o, C)$ (lines 3-4). Finally, it returns $\Delta_{labelSet}(o, C)$ (line 5).

B. `RI`: Is Object o Involved in a Row Instance of C

There is one issue in Algorithm 2 that remains unsolved, namely, the step to decide whether there is a row instance of a given label set C which involves an object o (line 5 in Algorithm 2). We denote by `RI` the problem corresponding to this remaining issue. Unfortunately, the `RI` problem is NP-hard, which we present in the following lemma.

Lemma 2: The `RI` problem, which is to decide for given label set C and an object o whether there exists a row instance of C involving o is NP-hard.

Proof 2: We prove by reduction from the existing Collective Spatial Keyword Query with the Diameter cost function (Dia-CoSKQ) problem [19], [6] which is NP-hard as follows.

First, we give the formal definition of the decision problem of Dia-CoSKQ. Given a set D of point-of-interests (POIs) where each POI p has a location $p.\lambda$ and a set of keywords $p.\psi$ and a query q with a query location $q.\lambda$, a set of query keywords $q.\psi$, and a real number c, the decision problem of Dia-CoSKQ is to decide whether there is a set S of POIs in D such that S covers all the query keywords (i.e., $q.\psi \subseteq \cup_{p \in S} p.\psi$) and the diameter of $S \cup \{q\}$, which corresponds to the maximum pairwise distance of $S \cup \{q\}$, is at most c.

Second, we transform a given decision problem instance of Dia-CoSKQ to a `RI` problem instance as follows. We construct a set O of objects by creating for each POI p in D $|p.\psi|$ objects each with $p.\lambda$ as its location and a keyword in $p.\psi$ as its label and another object o with its location as $q.\lambda$ and its label as a fictious one f. We create a set C containing those labels corresponding to the query keywords in $q.\psi$ and also t, i.e., $C = q.\psi \cup \{t\}$. Lastly, we set d to be c. The `RI` problem is to decide whether there exists a row instance of C which involves o. Clearly, the above transformation step is in polynomial time.

Third, it could be verified that the decision problem of Dia-CoSKQ is equivalent to that of `RI`, which finishes the proof. ■

C. A Filtering-and-Verification Approach for `RI`

A naive method for `RI` is to enumerate all row instances of C and check whether there exists one involving object o. However, as has been known in existing studies [33], [32], [31], the procedure of materializing all row instances of a given label set is very expensive. In this paper, we develop a *filtering-and-verification* approach for `RI`, which involves two phases, namely a filtering phase and a verification phase. The filtering phase is to solve `RI` for easy cases and the verification phase for all remaining cases. The details are introduced as follows.

1) Filtering Phase: The filtering phase is motivated by the fact that the remaining issue `RI` could be easy to solve with some information re-used and/or in certain cases:

- **Filter 1.** Check if there exists a row instance S of C, which was found previously when answering another `RI` instance for a different object $o^{'}$ and label set C, such that o is involved in S. If so, return "yes". To support this checking, we could keep track of all those objects that are involved in row instances that have been found.
- **Filter 2.** Check if all objects in $Disk(o, d)$ together carry all labels in C. If no, return "no" (since all possible sets of objects in $Disk(o, d)$ correspond to subsets of the set containing all objects in $Disk(o, d)$ and thus they cannot carry all labels in C either).
- **Filter 3.** Check if all objects in $Disk(o, d/2)$ together carry all labels in $C - \{o.t\}$. If so, return "yes" (since there exists a set of objects in $Disk(o, d/2)$ including o that corresponds to a row instance of C).

2) Filtering Phase: We propose three methods for the verification phase as follows.

Dia-CoSKQ-Adapt. This method is based on the close relationship between `RI` and Dia-CoSKQ. In the proof of the NP-hardness of `RI`, we show that any decision problem instance of Dia-CoSKQ could be transformed to a `RI` problem instance. Here, we further show that an arbitrary instance `RI` could be answered by solving a corresponding optimization problem instance of Dia-CoSKQ. Specifically, given an instance of `RI` which involves a set O of spatial objects, a set C of labels, a real number d, and one object o in O, we consider a Dia-CoSKQ problem which is to find a set S of POIs from a given set D of POIs which covers all query keywords of a given query q and has the diameter of $S \cup \{q\}$ the *smallest*, where the set D of POIs includes one POI for each object o in $Disk(o, d)$ with its location as $o.\lambda$ and its set of keywords as $\{o.t\}$ and the query q has its location at $o.\lambda$ and its set of query keywords as $C - \{o.t\}$. It could be verified that if the diameter of $S \cup \{q\}$ is at most d, the answer of the `RI` is "yes", and otherwise, the answer is "no". Based upon this, we can utilize the exact algorithm proposed in [19] for `RI`. Note that we could do slightly better by adopting an *early-stopping* strategy that whenever a set S with the diameter of $S \cup \{q\}$ at most d is found, it returns "yes" immediately.

Combinatorial-Search. We notice that enumerating all row instances of C is more than necessary for answering the question of `RI`. In fact, it would be sufficient to find one row instance of C which involves o if it exists to answer the question. Besides, there are two constraints that could be utilized for refining the search space. First, it is safe to focus the search on those objects which are near o, specifically, those in $Disk(o, d)$, since those objects outside this disk have their distances from o larger than d and cannot be involved in the same row instance together with o. Second, it is enough to consider those sets of objects each corresponding to a combination of objects with different labels in C since other sets of objects either cannot not carry the labels in C or have proper subsets which carry all the labels in C. Based upon the above two constraints, we design an algorithm for searching a possible row instance of C involving o if there exists one as follows.

- **Step 1.** it finds all objects in $Disk(o, d)$ by performing a range query with its center at o and its radius of d.
- **Step 2.** it indexes the objects found in Step 1 using an *inverted list* which stores the objects using different lists each corresponding to a label and contains all objects

with this label.

- **Step 3.** it tries all combinations of objects from those lists corresponding to the labels in $C - \{o.t\}$ and for each combination S which contains $|C - \{o.t\}|$ objects it checks whether the maximum pairwise distance of S is at most d. If such a combination is found, it stops by returning "yes", and otherwise, it returns "no".

Optimization-Search. In Combinatorial-Search, there is a step which is to enumerate all combinations of some objects in $Disk(o, d)$ indexed by their labels in $C' = C - \{o.t\}$ and see whether there exists a combination with the diameter at most the value d. An alternative for this step is to compute the set of objects in $Disk(o, d)$ which covers all labels in C' and has the smallest diameter and then compare this diameter against d to answer the question, i.e., if this diameter is at most d, it returns "yes", and otherwise, it answers "no". In the literature, the problem of finding a set objects which covers a given set of labels/keywords and has the smallest diameter has been studied [34], [35], [12] and is called the *m-closest keywords* (mCK) problem. Based upon this, we can utilize the exact algorithm proposed in [12] for mCK to do this step, and the resulting method corresponding to Optimization-Search. Similar to the Dia-CoSKQ-Adapt method, an early-stopping strategy could be adopted here.

Time Complexity Analysis. Since the verification phase dominates the time cost of the approach, we focus on the verification phase only. The complexity of Dia-CoSKQ-Adapt is $O(n_1 \cdot (C_{range} + k_3^{|C|-2} \cdot |C|^2))$, where n_1 ($n_1 << |O|$) is the number of objects that carry a label $t \in C - \{o.t\}$, k_3 ($k_3 << |O|$) is the number of objects shared by results of range queries. The complexity of Combinatorial-Search is $O(C_{range} + k_1 + k_2^{|C|})$, where C_{range} is the cost of performing the range query in Step 1, k_1 ($k_1 << |O|$) is the number of objects returned by the range query in Step 1, and k_2 ($k_2 << |O|$) is maximum number of objects in an inverted list constructed in Step 2. While the worst-case time complexity is exponential, the algorithm is feasible in practice with the help of index structures such as inverted lists and also because of the problem nature (e.g., the exponent $|C|$ is small in most cases), and this will be verified by the experiments. The complexity of Optimization-Search is $O(C_{range} + k_1 + n_1 \cdot k_1^{|C|-2})$.

IV. Empirical Studies

A. Experimental Set-up

Datasets. We use both real and synthetic datasets. The real dataset is the set of POIs of the United Kingdom (http://www.pocketgpsworld.com). Each POI has a textual description (e.g., supermarket, bank, cinema) and a GPS location. It consists of 182,334 objects with 36 types (i.e., labels).

The synthetic datasets are generated by following existing studies [13], [22] as follows. Step 1 (Label Set Generation): We generate N_{co_loc} subsets of labels one by one, and for each one, we construct it by sampling a certain number of labels randomly where the number follows a Poisson distribution with mean λ_1. We then construct $m_{overlap}$ maximal co-location patterns (i.e., label sets) from each set of labels constructed by augmenting it with one more random label. Step 2 (Instance Construction): For each maximal co-location pattern, we construct a certain number of instances where the number follows a Poisson distribution with mean λ_2, each by creating m_{clump} objects for each label in this instance and putting them inside a random grid cell with size $d \times d$ from the spatial frame of size $D \times D$. Step 3 (Noise Injection): We generate $(r_{noisy\_label} \times n_1)$ noisy labels, where n_1 is equal to the number of non-noisy labels (i.e., those generated in Step 1). We then construct $(r_{noisy\_num} \times n_2)$ noisy instances based on the noisy labels similarly as we did based on non-noisy labels (i.e., via Step 2), and put each noisy instance at a random grid cell, where n_2 is equal to the number of non-noisy instances (i.e., those generated in Step 2). We set $N_{co_{l}oc}$, λ_1, D, d, r_{noisy_label}, and r_{noisy_num} as 20, 5, 10^6, 10, 0.5, and 0.5, respectively. By following existing studies [13], [22], we set the other parameters as shown in Table II (with the default ones in bold). Note that the numbers of objects and labels in the synthetic datasets depend on the parameter settings. Under the default settings, the dataset contains $94,028$ objects and 462 labels.

TABLE II
Parameters and Settings

Parameter	Settings
λ_2	40, **50**, 60, 70, 80
m_{clump}	**1**, 2 ,3 ,4 ,5
$m_{overlap}$	1, 5, **10**, 15, 20
min-sup	0.2, 0.3, **0.4**, 0.5, 0.6

Algorithms. We test our Filtering-and-Verification approach. For comparison, we adapt the "Join-less" algorithm from [31] for two reasons. First, it is the state-of-the-art algorithm for co-location pattern mining. Second, though originally designed for participation-based measure, it involves procedures of computing the row instances of given label set, which is shared by our Fraction-Score measure. Specifically, the adapted algorithm works as follows. First, it generates all star neighborhoods. Second, for each label set C, it finds all the row instances from the corresponding star neighborhoods. Third, to check whether an object o is involved in C, it checks whether o exists in one of the row instances of C.

All algorithms were implemented in C/C++ and are memory-based. All experiments were conducted on a Linux platform with a 2.66GHz machine and 32GB RAM.

B. Experiment Results

1) Effectiveness Results on Synthetic Datasets: We compare Fraction-Score with the other approaches in terms of how close the supports measured are from the ground-truths. Note that we did not include the enumeration-based approach here since it is used for defining the confidence of a rule candidate only as mentioned in Section I. Besides, we use synthetic datasets only for the study here since it allows the flexibility to generate the datasets such that the ground-truths of supports could be estimated accurately. For this particular experiment, we set the parameter m_{clump}, i.e., the number objects to be generated for a label, to be a random number from a uniform distribution of $[1, 5]$ instead of a fixed number as we do for other experiments, and the purpose here is to test the robustness of support measures. Specifically, we estimate the ground-truth support of a pattern as the maximum number of disjoint row instances of the pattern. Based on the way we generate the synthetic datasets, this is close to the number of instances of a label (which follows $Pois(\lambda_2)$) with the smallest m_{clump} values among the labels in the pattern. For normalization, we then divide this number by the maximum number of objects that have a specific label in T.

Figure 3 shows the results of patterns with top-10 supports, where the x-axis corresponds to the patterns (in a descending order of their supports) and the y-axis shows the actual supports. According to these results, the supports by Fraction-Score are closest to the ground-truths among all approaches. This could probably be explained by the fact that the row instances that overlap with each other are not counted multiple times when collecting ground-truths, which is reasonable, while the participation-based approach would count those row instances which share some objects with their labels different from the one used for grouping the row instances as if they share nothing. The partitioning-based approach under-measures the supports since it misses some of the row instances, and the construction-based approach misses some of the row instances due to its heuristic nature.

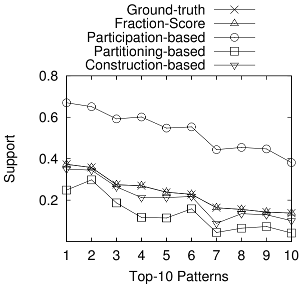


Fig. 3. Support value comparison

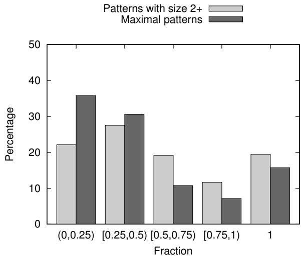


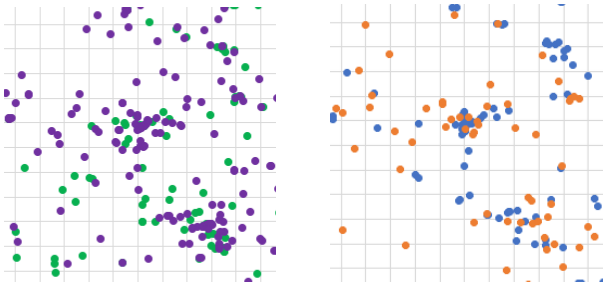
Fig. 4. Fractions in Fraction-Score

To look deeper into the supports defined by Fraction-Score, we are interested in knowing how the fractions in Fraction-Score are distributed. We consider two types of patterns, one corresponds to those patterns with size at least 2 (since those with size 1 are trivial) and the other corresponds to those patterns that are maximal (i.e., no subsets of them are patterns). Figure 4 shows the results, where the x-axis shows different ranges of fraction and the y-axis shows the percentage of groups (or the objects by which they are formed) that have their factions fall in a range. According to these results, around one-fifth of the groups have their fractions equal to 1 (which means that these groups are counted as an entire unit of prevalence) and the remaining groups have their fractions smaller than 1 (which means that these groups are counted fractionally only by Fraction-Score, but entirely by existing support measures such as the participation-based approach). Besides, for the type of maximal patterns, the percentages of those smaller fractions (e.g., those below 0.5) are higher than those for the patterns with size at least 2, and this is probably because the maximal patterns are usually of larger sizes, leading to a higher chance that their row instances overlap with each other.

2) Effectiveness Results on Real Datasets: In this part, we study the effectiveness of different support measures on real datasets. Specifically, we ran our algorithm and found the co-location patterns with top-5 supports (with the setting of $d = 1000m$). Table III presents the patterns, each with its supports computed using other approaches also shown. According to the results, we know that the supports by the participation-based approach are very close to 1 (which is mainly because this measure has a normalization step of dividing by the number of occurrences of the label but not the maximum among all labels as Fraction-Score does) and the supports by the construction-based and partitioning-based approaches are slightly smaller than those by Fraction-Score (which is mainly because the former ones miss some row instances while Fraction-Score captures all instances appropriately).

TABLE III
PATTERNS IN REAL DATASET

Pattern	Fraction-Score	Partici-pation	Partit-ioning	Constr-uction
{church, restaurant}	0.7017	0.9613	0.6164	0.6829
{church, gas station}	0.5908	0.9832	0.5076	0.5595
{restaurant, gas station}	0.5132	0.9577	0.4324	0.4856
{church, restaurant, gas station}	0.5027	0.9552	0.3952	0.4659
{ATM, church}	0.4301	0.9093	0.4025	0.4280



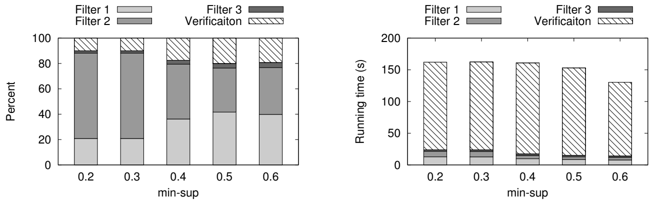
(a) church (green dots) and restaurant (purple dots)

(b) recreation (orange dots) and supermarket (blue dots)

Fig. 5. Distributions of the objects in the real dataset

Besides, we visualize the objects involving the labels in the first pattern C_1 ={church, restaurant} in Figure 5 (a) and those involving the labels in another pattern C_2 ={recreation, supermarket} for comparison, where the side length of each cell is $d = 500m$. We can observe that the pattern C_1 occurs 3-4 times more than C_2, which matches our result of $sup(C_1) \approx 3.7 \cdot sup(C_2)$. Consider the co-location rule "church → restaurant". The confidence of the rule is 0.7362, which matches the distribution shown in Figure 5(a) that around 70% of churches have one restaurant nearby. Consider another co-location rule "supermarket → recreation" with confidence 0.6025. This also matches the distribution shown in Figure 5(b) that around two-thirds of the supermarkets have a recreation nearby.

3) Results on the Filtering-and-Verification Approach: **Filtering phase.** In this part, we show the results reflecting the effectiveness of the filtering phase. Consider Figure 6(a), where we vary $min\text{-}sup$ and measure the percentage of RI instances that are solved by each of the three filters in the filtering phase and also that by the verification phase. These results show that more than 80% of RI instances could be solved in the filtering phase, and thus less than 20% RI instances would be left in the verification phase. Besides, we notice that when $min\text{-}sup$ increases, the filtering power of Filter 1 increases while that of Filter 2 decreases. The former is because the number of large co-location patterns decreases when $min\text{-}sup$ increases and as a consequence, it is more likely to find a row instance of a label set, which benefits Filter 1. And the latter is because when $min\text{-}sup$ increases, it becomes rare for $Disk(o, d)$ to not cover all labels of a label set (which is of a small size) and thus the filtering power of Filter 2 decreases. The results on the real datasets provide similar clues and thus they are omitted due to page limit.



(a) Percent of No. of RI

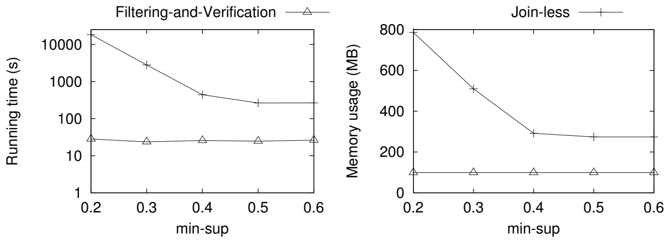
(b) Running time

Fig. 6. Effectiveness of the filtering phase (Synthetic datasets)

Verification phase. We conducted experiments on both real and synthetic datasets for studying the performance of the three methods proposed for the verification phase. According to the results, Combinatorial-Search runs the fastest consistently under all settings. This could probably explained by the fact that the exact algorithms employed in Dia-CoSKQ-Adapt and Optimization-Search were originally designed for some optimization problem (i.e., Dia-CoSKQ and mCK problems) while RI is a decision problem. These exact algorithms involve extra steps for finding an optimal solution and thus they take more time. Therefore, we focus on Combinatorial-Search in the verification phase for the remaining experiments. With Combinatorial-Search used in the verification phase, the breakdown of the running time is shown in Figure 6(b).

4) Filtering-and-Verification vs State-of-the-art: In this part, we show the results reflecting the performance comparison between Filtering-and-Verification and the state-of-the-art, Join-less, in terms of running time and memory consumption.

Figure 7 shows the results on the real dataset where we vary $min\text{-}sup$. According to Figure 7(a), the running times of both algorithms decrease when $min\text{-}sup$ increases. This is because fewer co-location patterns would be found when $min\text{-}sup$ increases. Besides, our Filtering-and-Verification approach runs much faster than the Join-less method, which could be explained by the fact that the former only needs to check whether some objects are involved in any of the row instances while the latter needs to find all row instances of each co-location pattern. According to Figure 7(b), our Filtering-and-Verification approach consumes significantly less memory than the Join-less method, which is because the former only maintains the fractions received by each object for each label while the latter needs to store all row instances of each co-location pattern.



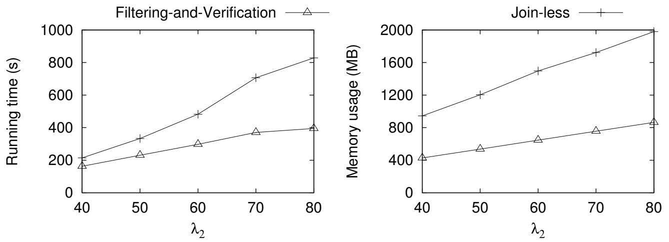
(a) Running time

(b) Memory usage

Fig. 7. Effect of $min\text{-}sup$ (Real dataset)

The results on synthetic datasets when we vary $min\text{-}sup$ provide similar clues and thus they are omitted.

Figure 8 shows the results on synthetic datasets where we vary λ_2. According to Figure 8(a), the running times of both algorithms increase when λ_2 increases. This is because when the average size of the row instances increases, more objects need to be checked. Our Filtering-and-Verification approach outperforms the Join-less method, and the gap increases with λ_2. According to Figure 8(b), the memory consumptions of both algorithms increase with λ_2. This is because the datasets would involve more objects when λ_2 increases. The memory usage of our Filtering-and-Verification approach is much smaller than that of the Join-less method consistently.

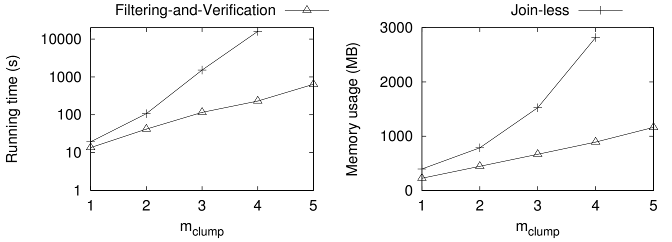


(a) Running time

(b) Memory usage

Fig. 8. Effect of λ_2 (Synthetic datasets)

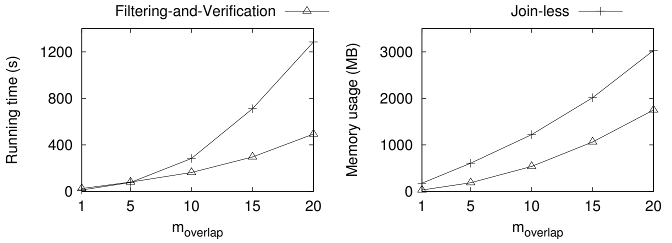
Figure 9 shows the results on synthetic datasets where we vary m_{clump}, where the results of the Join-less method for m_{clump} = 5 are missed which is simply because it ran for a very long time, i.e., more than 6 hours (this strategy applies for all the following results). According to Figure 9(a), the running times of both algorithms increase when m_{clump} increases. This is because the number of co-location patterns increases when m_{clump} increases. Besides, our Filtering-and-Verification approach runs faster than the Join-less method by orders of magnitude. This is because the latter needs to enumerate all row instances for each co-location pattern, which is very time-consuming (and memory-consuming). This also shows that the Filtering-and-Verification approach is scalable to m_{clump} while the Join-less method is not. According to Figure 9(b), the memory consumptions of both algorithms increase when m_{clump} increases, which is simply because the total number of objects increases with m_{clump}.



(a) Running time (b) Memory usage

Fig. 9. Effect of m_{clump} (Synthetic datasets)

Figure 10 shows the results on synthetic datasets when we vary $m_{overlap}$. According to Figure 10(a), the running times of both algorithms increase when $m_{overlap}$ increases and our Filtering-and-Verification approach outperforms the Join-less method. According to Figure 10(b), the memory consumption of our Filtering-and-Verification approach is consistently much smaller than that of the Join-less method.

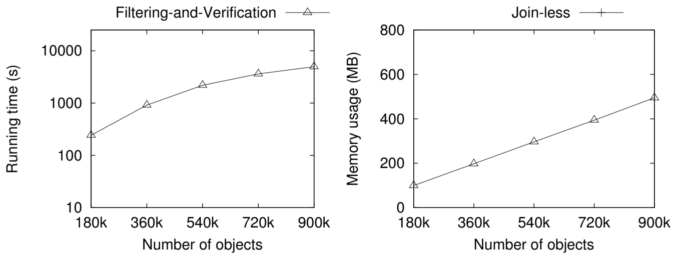


(a) Running time (b) Memory usage

Fig. 10. Effect of $m_{overlap}$ (Synthetic datasets)

5) Scalability Test: We generated 5 synthetic datasets from the real dataset for scalability test. Specifically, for each object o in the original dataset, we create n new objects each with location set to be a random location from the original dataset by following the distribution and label set to be $o.t$. We vary the number n from 1 to 5 and obtain synthetic datasets with sizes $\{180k, 360k, 540k, 720k, 900k\}$. Figure 11 shows the results, according to which, we see that the Join-less method cannot scale to large datasets, e.g., it ran for more than 2 days on dataset of size about 180k (and thus its plots are missing), and our Filtering-and-Verification method could scale up on large datasets of size 1M.

Conclusion on Results. Our Fraction-Score measures the prevalence of co-location pattern candidates more properly than existing ones. Three filters in the filtering phase are effective (e.g., they filter more than 80% `RI` instances), and among three methods in the verification phase, Combinatorial-Search works the best. Besides, our Filtering-and-Verification approach works consistently better than the state-of-the-art in terms of both running time and memory consumption.



(a) Running time (b) Memory usage

Fig. 11. Scalability test

V. Related Work

The co-location pattern mining problem has been studied extensively using different support measures [22], [20], [14], [13], [33], [32], [31]. In [36], the authors proposed to improve the efficiency of co-location pattern mining by adopting a multi-way join approach. In [15], the authors developed a FP-tree based algorithm for the co-location pattern mining problem. Motivated by the fact that it is expensive to generate row instances of a size $(k + 1)$ label set via join the row instances of two size k label sets, in [33], [32], [31], the authors proposed some partial join and joinless techniques which materialize some transactions of spatial objects such that those row instances within transactions could be generated without the join process [22], but for those row instances across different transactions, they still use the join process.

Some other studies related to the co-location pattern mining problem are reviewed as follows. In [17], the authors aimed to find strong association rules where a rule indicates certain association relationship among a set of spatial and possibly nonspatial predicates. In [27], the authors presented a framework for mining co-location patterns for extended spatial objects, e.g., polygons and line strings. In [8], the authors studied the problem mining *regional* (or local) co-location patterns. In [4], the authors developed a new method to efficiently process co-location pattern queries using materialized, improved candidate pattern instance tree (iCPI-tree). In [26], the problem studied is to find regions each represented by a set of cells linking with each other where two labels co-occur more frequently than globally. In [9], it was studied to find regions each represented as a set of spatial objects by using a clustering-like algorithm where the interestingness score of a region is based on how much the objects representing the region have their continuous values co-related with each other. In [5], the authors proposed to find zonal or local co-location patterns which represent subsets of label types that are frequently located in a subset of space (i.e., zone). In [18], the authors studied the problem of summarizing co-location patterns. In [3], the problem studied was to find statistically significant co-location patterns based on hypothesis testing, where some models are assumed which limits its application scope. In [30], [23] and [29], [2], [21], Map Reduce based methods and parallel algorithms on GPU were developed for the co-location pattern mining problem, respectively. In [11],

[10], it was studied to find co-location patterns where a set C of spatial labels corresponds to a pattern if the clusterings each based on the objects with a spatial label in C have at least a certain degree of overlap which is captured by the area intersected by the polygons formed based on the clusters. In [16], it performs clustering on the set of spatial labels where the similarity between two labels is measured with some spatial statistical functions [7]. In [28], the authors studied the co-location pattern mining problem with the consideration of distance decay effects and also the direction information.

VI. Conclusion

In this paper, we studied the co-location pattern mining problem on spatial objects with categorical labels. We showed that existing support measures suffer from various weaknesses and thus we proposed a new measure called Fraction-Score which quantifies the prevalence of pattern candidates properly. We developed an Apriori-like algorithm for mining co-location patterns based on Fraction-Score, which involves a fundamental operation of deciding whether an object is involved in a row instance of a label set. We proved that the problem of performing the operation is NP-hard and then developed a filtering-and-verification algorithm for the operation. We conducted experiments on both real and synthetic datasets, which verified that our Fraction-Score measures the prevalence better than existing approaches and also our algorithm runs significantly faster than the adaption of the state-of-the-art. In the future, we plan to study the co-location pattern mining problem on spatio-temporal data, i.e., a time dimension is taken into consideration, and this is interesting since some co-location patterns occur only at certain time stamps.

Acknowledgements: The research of Harry Kai-Ho Chan and Raymond Chi-Wing Wong is supported by HKRGC GRF 16214017 and ITS/227/17FP. The research of Cheng Long is supported by NTU SUG M4082302.020. The research of Da Yan is supported by NSF OAC-1755464 and NSF DGE-1723250.

References

[1] R. Agrawal and R. Srikant. Fast algorithms for mining association rules. In *PVLDB*, volume 1215, pages 487–499, 1994.

[2] W. Andrzejewski and P. Boinski. Parallel gpu-based plane-sweep algorithm for construction of icpi-trees. *Journal of Database Management (JDM)*, 26(3):1–20, 2015.

[3] S. Barua and J. Sander. Mining statistically significant co-location and segregation patterns. *TKDE*, 26(5):1185–1199, 2014.

[4] P. Boinski and M. Zakrzewicz. Collocation pattern mining in a limited memory environment using materialized icpi-tree. In *DaWaK*, pages 279–290. Springer, 2012.

[5] M. Celik, J. M. Kang, and S. Shekhar. Zonal co-location pattern discovery with dynamic parameters. In *ICDM*, pages 433–438. IEEE, 2007.

[6] H. K.-H. Chan, C. Long, and R. C.-W. Wong. On generalizing collective spatial keyword queries. *TKDE*, 30(9):1712–1726, 2018.

[7] N. Cressie. Statistics for spatial data. *Terra Nova*, 4(5):613–617, 1992.

[8] W. Ding, C. F. Eick, J. Wang, and X. Yuan. A framework for regional association rule mining in spatial datasets. In *ICDM*, pages 851–856. IEEE, 2006.

[9] C. F. Eick, R. Parmar, W. Ding, T. F. Stepinski, and J.-P. Nicot. Finding regional co-location patterns for sets of continuous variables in spatial datasets. In *SIGSPATIAL*, page 30. ACM, 2008.

[10] V. Estivill-Castro and I. Lee. Data mining techniques for autonomous exploration of large volumes of geo-referenced crime data. In *Proc. of the 6th International Conf. on Geocomputation*, pages 24–26, 2001.

[11] V. Estivill-Castrol and A. T. Murray. Discovering associations in spatial dataan efficient medoid based approach. In *PAKDD*, pages 110–121. Springer, 1998.

[12] T. Guo, X. Cao, and G. Cong. Efficient algorithms for answering the m-closest keywords query. In *SIGMOD*, pages 405–418. ACM, 2015.

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

[14] Y. Huang, H. Xiong, S. Shekhar, and J. Pei. Mining confident co-location rules without a support threshold. In *Proceedings of the 2003 ACM symposium on Applied computing*, pages 497–501. ACM, 2003.

[15] Y. Huang, L. Zhang, and P. Yu. Can we apply projection based frequent pattern mining paradigm to spatial co-location mining? *Advances in Knowledge Discovery and Data Mining*, pages 131–141, 2005.

[16] Y. Huang and P. Zhang. On the relationships between clustering and spatial co-location pattern mining. In *ICTAI*, pages 513–522. IEEE Computer Society, 2006.

[17] K. Koperski and J. Han. Discovery of spatial association rules in geographic information databases. In *Advances in spatial databases*, pages 47–66. Springer, 1995.

[18] B. Liu, L. Chen, C. Liu, C. Zhang, and W. Qiu. Rcp mining: Towards the summarization of spatial co-location patterns. In *SSTD*, pages 451–469. Springer, 2015.

[19] C. Long, R. C.-W. Wong, K. Wang, and A. W.-C. Fu. Collective spatial keyword queries: a distance owner-driven approach. In *SIGMOD*, pages 689–700. ACM, 2013.

[20] Y. Morimoto. Mining frequent neighboring class sets in spatial databases. In *KDD*, pages 353–358. ACM, 2001.

[21] A. M. Sainju and Z. Jiang. Grid-based colocation mining algorithms on gpu for big spatial event data: A summary of results. In *SSTD*, pages 263–280. Springer, 2017.

[22] S. Shekhar and Y. Huang. Discovering spatial co-location patterns: A summary of results. In *SSTD*, pages 236–256. Springer, 2001.

[23] M. Sheshikala, D. R. Rao, and R. V. Prakash. Join-less approach for finding co-location patterns-using map-reduce framework. *Journal of Theoretical and Applied Information Technology*, 87(2):355, 2016.

[24] T. Uno, M. Kiyomi, and H. Arimura. Lcm ver. 3: collaboration of array, bitmap and prefix tree for frequent itemset mining. In *Proceedings of the 1st international workshop on open source data mining: frequent pattern mining implementations*, pages 77–86. ACM, 2005.

[25] L. Wang, Y. Baoa, and Z. Lu. Efficient discovery of spatial co-location patterns using the icpi-tree. *Open Information Systems Journal*, 3(1):69–80, 2009.

[26] S. Wang, Y. Huang, and X. S. Wang. Regional co-locations of arbitrary shapes. In *SSTD*, pages 19–37. Springer, 2013.

[27] H. Xiong, S. Shekhar, Y. Huang, V. Kumar, X. Ma, and J. S. Yoc. A framework for discovering co-location patterns in data sets with extended spatial objects. In *SDM*, pages 78–89. SIAM, 2004.

[28] X. Yao, L. Chen, L. Peng, and T. Chi. A co-location pattern-mining algorithm with a density-weighted distance thresholding consideration. *Information Sciences*, 396:144–161, 2017.

[29] J. S. Yoo and D. Boulware. Incremental and parallel spatial association mining. In *Big Data (Big Data), 2014 IEEE International Conference on*, pages 75–76. IEEE, 2014.

[30] J. S. Yoo, D. Boulware, and D. Kimmey. A parallel spatial co-location mining algorithm based on mapreduce. In *BigData Congress*, pages 25–31. IEEE, 2014.

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

[32] J. S. Yoo, S. Shekhar, and M. Celik. A join-less approach for co-location pattern mining: A summary of results. In *ICDM*. IEEE, 2005.

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

[34] D. Zhang, Y. M. Chee, A. Mondal, A. K. Tung, and M. Kitsuregawa. Keyword search in spatial databases: Towards searching by document. In *ICDE*, pages 688–699. IEEE, 2009.

[35] D. Zhang, B. C. Ooi, and A. K. Tung. Locating mapped resources in web 2.0. In *ICDE*, pages 521–532. IEEE, 2010.

[36] X. Zhang, N. Mamoulis, D. W. Cheung, and Y. Shou. Fast mining of spatial collocations. In *KDD*, pages 384–393. ACM, 2004.