Fraction-Score: A Generalized Support Measure for Weighted and Maximal Co-location Pattern Mining

Harry Kai-Ho Chan, Cheng Long, Da Yan, Raymond Chi-Wing Wong, and Hua Lu

Abstract—Co-location patterns, which capture the phenomenon that objects with certain labels are often located in close geographic proximity, are defined based on a support measure which quantifies the prevalence of a pattern candidate in the form of a label set. Existing support measures share the 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 collectively carry all labels in C and are located close to one another. However, they suffer from various weaknesses, e.g., fail to capture all possible instances, or overlook the cases when multiple instances overlap. In this paper, we propose a new measure called Fraction-Score which counts instances *fractionally* if they overlap. Fraction-Score captures all possible instances, and handles the cases where instances overlap appropriately (so that the supports defined are more meaningful and anti-monotonic). We develop efficient algorithms to solve the co-location pattern mining problem defined with Fraction-Score. Furthermore, to obtain representative patterns, we develop an efficient algorithm for mining the maximal co-location patterns, which are those patterns without proper superset patterns. We conduct extensive experiments using real and synthetic datasets, which verified the superiority of our proposals.

Index Terms—Co-location pattern, spatial data mining

1 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 not only possess labels such as their species, but also location information about their habitats; in urban areas, point-of-interests (POIs) such as restaurants and shops are also associated with some labels such as their business types and brands as well as their locations (e.g., in Google Maps); also 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 call an object as an *instance* of a label if the object carries the label. One interesting pattern on these objects is the *co-location pattern* [26], [28], [19], [18]. 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 a distance d from each other). As an example, Snack Bar shops and Beauty Salon shops are often found located near each other [26], forming a co-location pattern.

- *H.K.-H. Chan is with the Information School, University of Sheffield, U.K. E-mail: h.k.chan@sheffield.ac.uk*
- *C. Long is with the School of Computer Science and Engineering, Nanyang Technological University, Singapore. E-mail: c.long@ntu.edu.sg*
- *D. Yan is with the Department of Computer Science, The University of Alabama at Birmingham. E-mail: yanda@uab.edu*
- *R.C.-W. Wong is with the Department of Computer Science and Engineering, The Hong Kong University of Science and Technology, Clear Water Bay, Kowloon, Hong Kong. E-mail: raywong@cse.ust.hk*
- *H. Lu is with the Department of People and Technology, Roskilde University, Denmark. E-mail: luhua@ruc.dk*

Corresponding Authors: Harry Kai-Ho Chan and Cheng Long.

Similar to *frequent itemsets* in the context of transaction data [1], co-location patterns are defined based on a support measure, which quantifies how frequently those instances of the labels in a given label set are located closely. 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.

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. Figure 1 shows an example. Both sets $\{R_7, C_1\}$ and $\{R_8, C_1\}$ are instances of the label set {restaurant, church}. However, the two sets are overlapped by the object C_1. In the literature, several support measures for co-location patterns have been proposed, namely *partitioning-based* [28], *construction-based* [26], *enumeration-based* [28], [19], [18], and *participation-based* [28], [19], [18], [46], [45], [44]. 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, as will be discussed in Section 2, they all suffer from various weaknesses such as missing or over-count instances, or is not anti-monotonic.

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. For example, the set $\{R_7, C_1\}$ is a row instance of the label set {restaurant, church} in Figure 1, while $\{R_7, R_8, C_1\}$ is not. In our prior work [8], we propose a

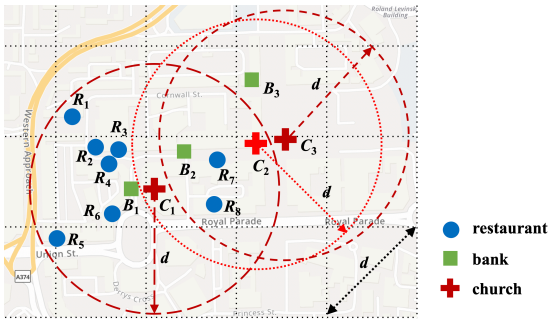


Fig. 1. A small portion of a real dataset of POIs in United Kingdom, including 8 restaurants (blue), 3 banks (green) and 3 churches (red), where the icons indicate the labels of the spatial objects and the disks have their centers at C_1 - C_3 and radii all equal to d.

new support measure called *Fraction-Score* which puts all possible row instances into different *groups* 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. Compared to the participation-based approach that also groups the row instances (to be detailed in Section 2), Fraction-Score avoids the over-counting problem. 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 3). 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, which is explained as follows. First, 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. Second, 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). Third, 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.

The sum of fractions, $1/8 \cdot 6 + 1 + 1 = 2.75$, corresponds to the support of {restaurant, church} by Fraction-Score. This is more meaningful than 8 that is the support defined by the participation-based approach, which we will see shortly in Section 2, 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).

The example above illustrates the cases in an unweighted dataset. However, in some cases, each object contains a weight attribute which quantifies its importance. For example, the NeuroSynth dataset [39] (details will be given in Section 6.1) contains a mapping between labels (e.g., "depression" and "anxiety") and the activated locations in the brain (i.e., location). Each object weight is a relevance score between the label and the location. As a generalization of the definition in [8] that defined Fraction-Score based on unweighted objects, our Fraction-Score proposed in this journal extension also works well on these weighted datasets, since it seamlessly captures object weights in its support definition. The unweighted case proposed in [8] is a special case with all weights equal to 1.

Moreover, as will be shown later, the support defined by Fraction-Score satisfies the desirable anti-monotonicity property. Based on Fraction-Score, we define co-location patterns using a pre-set parameter minimum support.

Since Fraction-Score satisfies the anti-monotonicity property, we adopt an Apriori-like algorithm for mining the co-location patterns. 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 [28], [19], [18], [46], [45], [44] are computed, and it 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. 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 performs a verification procedure for those remaining cases. Note that the algorithm improved over the one in [8] in both memory usage and efficiency by additionally including a memory-saving strategy and filtering and pruning steps.

In addition, we found that the number of patterns returned is large in some cases, which might cause difficulty for users to interpret the results. Thus, we study the *maximal co-location patterns* [43] mining problem based on Fraction-Score, where a pattern is maximal if it has no proper superset pattern. It is particularly useful when we want to obtain a smaller set of patterns that can concisely represent all the co-location patterns. We propose an efficient algorithm for mining all maximal co-location patterns. The major idea is to generate candidate maximal patterns from the size-2 patterns, and verify them in a top-down manner. Thus, it avoids the unnecessary computations in the above Apriori-like algorithm which is designed for mining all patterns. Compared to the existing maximal pattern mining algorithm [38] that generates the candidate patterns from size-2 instance table, we do not need to materialize the instances. In the verification, the filtering-and-verification approach is

also adopted, with an additional filter for better efficiency.

The contributions of this paper are summarized as follows.

- We show the weaknesses of existing support measures and propose a new and better one called Fraction-Score, which avoids the weaknesses and satisfies the desirable anti-monotonicity property.
- For a fundamental operation involved in mining the co-location patterns, we provide hardness results and design an efficient algorithm.
- We propose an efficient algorithm for answering the maximal co-location pattern mining problem based on Fraction-Score.
- 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 algorithms.

This journal extension adds substantial new technical contributions over [8] by (1) generalizing the definition of Fraction-Score to be applicable on weighted objects (Section 3.3); (2) improving the algorithms to be more memory-saving and efficient (Section 4); (3) proposing an efficient algorithm to find the maximal patterns (Section 5); (4) including an additional real dataset NeuroSynth [39] to evaluate our algorithms (Section 6); and (5) releasing the source code of our algorithms[1].

The rest of the paper is organized as follows. Section 2 reviews some related work. Section 3 gives the formal definition of Fraction-Score and defines our problems. Section 4 adopts an Apriori-like algorithm for mining the co-location patterns and introduces an algorithm for computing the support defined by Fraction-Score. Section 5 discusses the maximal co-location pattern mining based on Fraction-Score. Section 6 presents the experimental results. Section 7 concludes the paper and provides some future directions.

2 Related Work

2.1 Support Measures for Co-location Pattern Mining

The co-location pattern mining problem has been studied extensively using different support measures. We illustrate the weaknesses of different approaches as follows. Table 1 summarizes them and compares with Fraction-Score.

TABLE 1
Existing support measures

Approach	Prevalence	Anti-monotonic
Partitioning-based [28]	miss-count	yes
Construction-based [26]	miss-count	no
Enumeration-based [28], [19], [18]	counter-intuitive	no
Participation-based [28], [19], [18], [44], [45], [46]	over-measured	yes
Fraction-Score	meaningful	yes

Partitioning-based approach [28] 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.

Construction-based approach [26] constructs instances of a given label set heuristically and counts the number of constructed instances as the support. This approach is not robust simply because some instances of a label set might be missed due to the heuristic nature.

Enumeration-based approach [28], [19], [18] counts for a given label set all its *row instances*. With this approach, no instances can be missed, but the support definition is not anti-monotonic and counter-intuitive. That is, 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 [28], [19], [18].

Participation-based approach [28], [19], [18], [46], [45], [44] 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 object's contribution is over-counted. To illustrate, consider 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.

Note that the partitioning-based, construction-based and participation-based approaches can be adapted to handle weighted objects. The details can be found in Appendix A.

In [49], Zhang et al. proposed to improve the efficiency of co-location pattern mining by adopting a multi-way join approach. In [20], Huang et al. 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 joining the row instances of two size-k label sets, in [46], [45], [44], 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 [28], but for those row instances across different transactions, they still use the join operation. In [4], Boinski and Zakrzewicz developed a new method to

1. https://github.com/harryckh/TKDE-colocation

efficiently process co-location pattern queries using materialized, improved candidate pattern instance tree (iCPI-tree).

2.2 Condensed Co-location Pattern Mining

In [43], Yoo and Bow studied the closed top-k co-location pattern mining problem. The authors also studied the maximal co-location pattern mining problem [42]. In [38], Yao et al. proposed to construct a graph based on size-2 co-location patterns, and then find maximal cliques as the maximal co-location pattern candidates for better efficiency. In [23], Liu et al. studied the problem of summarizing co-location patterns. In [31], Wang et al. proposed a redundancy reduction for co-location patterns. All these studies aimed at finding a representative set of patterns that is of a smaller size. However, their definitions and methods are designed based on the participation-based measures, and thus cannot be used in our Fraction-Score.

2.3 Variants of Co-location Pattern Mining

Some works defined the spatial co-location pattern based on regions and polygons. In [35], Xiong et al. presented a framework for mining co-location patterns for extended spatial objects, e.g., polygons and line strings. In [10], Ding et al. studied the problem mining *regional* (or local) co-location patterns. In [11], Eick et al. studied the problem of finding regions that 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 [13], [12], the authors studied of finding 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 [6], Celik et al. 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 [33], Wang et al. studied the problem of finding regions that each represented by a set of cells linking with each other where two labels co-occur more frequently than globally. In [25], Long et al. proposed to find the co-location patterns from regional objects, and defined the proximity relationship between instances by their overlapping area.

Some other studies related to the co-location pattern mining problem are reviewed as follows. In [22], Koperski and Han aimed to find strong association rules where a rule indicates certain association relationship among a set of spatial and possibly non-spatial predicates. In [3], Barua and Sander studied the problem of finding statistically significant co-location patterns based on hypothesis testing, where some models are assumed which limits its application scope. In [21], Huang and Zhang proposed to cluster on the set of spatial labels where the similarity between two labels is measured with some spatial statistical functions [9]. In [37], Yang et al. studied the co-location pattern mining problem with the consideration of distance decay effects and also the direction information. In [36], Yang et al. studied the problem of finding the co-location patterns with or without rare features. In [41], [29] (resp. [40], [2], [27]), MapReduce based methods (resp. parallel algorithms on GPU) were developed for the co-location pattern mining problem.

TABLE 2
Notation table

Notation	Definitions
t	a label
o	a spatial object with its location $o.\lambda$, its label $o.t$ and its weight $o.w$
T	the set of all possible labels of the objects
C	a label set (or a co-location pattern candidate)
O	the set of spatial objects
O_t	the set of spatial objects with the label t
W_t	the total weight of objects with the label t
W_{max}	the largest total weight among all W_t
d	the distance threshold for defining neighbor sets
$\Theta(o', t, d)$	the set of objects which are located in $Disk(o', d)$ and carry the label t
$\Delta_{obj}(o, o')$	the amount of fraction of o' that o receives
$\Delta_{label}(o, t')$	the aggregated fraction of objects sharing a label $t' \in C - \{t\}$ that o receives
$\Delta_{labelSet}(o, C)$	the aggregated fraction o receives w.r.t. C

3 Fraction-Score and Problem Definition

Section 3.1 introduces some notations. Section 3.2 gives an overview of Fraction-Score, and Section 3.3 presents its formal definition. Section 3.4 defines our problems.

3.1 Notations

Let O be a set of n objects. Each object $o \in O$ has a location $o.\lambda$, a weight $o.w$ in range $[0, 1]$ that represents the importance of the object, 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. For example, object A_1 in Figure 2 has the label $\circ$ and a weight 0.8.

Let T be the set of all possible labels of the objects, i.e., $T = \{o.t | o \in O\}$. Let O_t be the set of objects with label t, i.e., $O_t = \{o | o.t = t\}$. Given a label t, we use W_t to denote the sum of weights of the objects in O_t, i.e., $W_t = \sum_{o \in O_t} o.w$, and W_{max} to denote the largest W_t among all $t \in T$.

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., $max_{o,o' \in S} d(o, o') \leq d$. Given an object o and a real number r, we denote by $Disk(o, r)$ the disk with its center at $o.\lambda$ and its radius equal to r. Given a label set C, 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. The main notations that are used throughout the paper are summarized in Table 2.

3.2 Overview of Fraction-Score

Same as the participation-based approach, Fraction-Score groups the row instances of C by the objects with a given

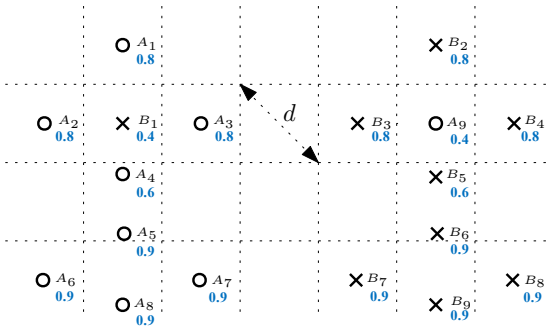


Fig. 2. A toy example where × and ∘ are two labels, and A_1-A_9 and B_1-B_9 are 18 objects each with exactly one label indicated by the shape representing the object, and its weight is indicated by the values in blue.

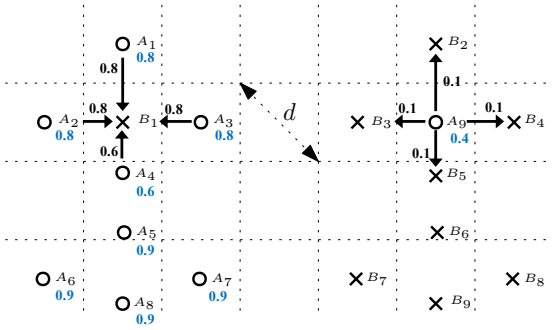


Fig. 3. The distribution of the fractions from the perspective of objects with label ∘, represented by the arrows with solid lines and the values in black

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 a fraction of $o^{'}$ to each group among all groups whose row instances share $o^{'}$. This fraction is equal to $o^{'}.w$ divided by the total number of such groups. That is, Fraction-Score splits object weight $o^{'}.w$ into some equal fractions and distributes these fractions to all groups of row instances that 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.

3.3 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. Let us denote by $\Theta(o^{'}, t, d)$ the set of objects which are located in $Disk(o^{'}, d)$ and carry the label t (note that $o \in \Theta(o^{'}, t, d)$). Motivated by the previous observation, Fraction-Score splits $o^{'}$ into $|\Theta(o^{'}, t, d)|$ equal fractions each equal to $o^{'}.w/|\Theta(o^{'}, t, d)|$ and then distributes each fraction to an object in $\Theta(o^{'}, t, d)$, To illustrate, consider Figure 2. We have $\Theta(A_1, \times, d) = \{B_1\}$ and $\Theta(A_9, \times, d) = \{B_2, B_3, B_4, B_5\}$. Thus, a fraction 0.8 from A_1's weight is distributed to B_1 and a fraction 0.1 from A_9's weight is distributed to each of B_2 - B_5. The intuition here is that A_1's weight could be shared by 1 group (one with a fraction of 0.8) and A_9 by 4 groups (each with an equal fraction 0.1, i.e., $1/4$ of 0.4).

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 $o \in \Theta(o^{'}, t, d)$. Besides, the amount of fraction of an object $o^{'}$ that o receives, denoted by $\Delta_{obj}(o, o^{'})$, is equal to $o^{'}.w/|\Theta(o^{'}, t, d)|$, i.e.,

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

Consider the example in Figure 2. We have $\Delta_{obj}(B_1, A_1) = \frac{A_1.w}{|\Theta(A_1, \times, d)|} = 0.8$, which means B_1 receives a fraction 0.8 from A_1. Similarly, $\Delta_{obj}(B_2, A_9) = \frac{A_9.w}{|\Theta(A_9, \times, d)|} = 0.1$, which means B_2 receives a fraction 0.1 from A_9. Note that this is a generalization of the definition of unweighted case in [8].

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 $\Theta(o, t^{'}, d)$), denoted by $\Delta_{label}(o, t^{'})$, is defined as

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

Consider the example in Figure 3 where $C = \{\times, \circ\}$. We have $\Delta_{label}(B_1, \circ) = \min\{3, 1\} = 1$ since B_1 receives $0.8 + 0.8 + 0.8 + 0.6 = 3$ from A_1 - A_4. Similarly, $\Delta_{label}(B_2, \circ) = \min\{0.1, 1\} = 0.1$.

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 [28], [19], [18], 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 it corresponds to the worst-case scenario that one object is shared by multiple groups. We denote the aggregated fraction o receives w.r.t. C by $\Delta_{labelSet}(o, C)$, i.e.,

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

The above definition is for cases where $|C| \geq 2$, and in the case when $|C| = 1$, we simply define $\Delta_{labelSet}(o, C) = o.w$. Consider the example in Figure 2 where $C = \{\times, \circ\}$. We have $\Delta_{labelSet}(B_1, C) = \Delta_{label}(B_1, \circ) = 1$ and $\Delta_{labelSet}(B_2, C) = \Delta_{label}(B_2, \circ) = 0.1$.

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.,

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

Consider the example in Figure 3 where $C = \{\times, \circ\}$. In this case, we have $Obj(\times, C) = \{B_1, B_2, ..., B_5\}$. Then, $sup(C|\times) = \sum_{i=1}^{5} \Delta_{labelSet}(B_i, C) = 1 + 4 \cdot 0.1 = 1.4$.

Note that depending on different choices of label t, we may have different $sup(C|t)$. To capture the worst-case prevalence, we choose the label given which the value is the smallest. Besides, we normalize the value to $[0, 1]$ by dividing it by the maximum total weight W_{max} among the labels in T. In summary, the support of a given label set C, denoted by $sup(C)$, is defined as follows.

$$sup(C) = \frac{\min_{t \in C} sup(C|t)}{W_{max}} \tag{5}$$

Consider the example in Figure 2 again where $C = \{\times, \circ\}$. We have $sup(C) = \min\{\frac{sup(C|\times)}{7}, \frac{sup(C|\circ)}{7}\} = \frac{1.4}{7} = 0.2$.

It is worth mentioning that all row instances are captured and counted appropriately by Fraction-Score. All instances that are involved in any row instance (and thus possibly contributing to the support of the label set) are considered, and thus no instance is missed. 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. The correctness relies on the fact $sup(C'|t) \geq sup(C|t)$ for any t in C' which could be verified by checking the following facts against Equation (4): (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 $C' \subseteq C$). □

3.4 Problem Definition

We formally define the co-location pattern mining problem.

Problem (Co-location Pattern Mining)**.** *Given a set O of objects, each with a location, a weight and a label, a distance threshold d for defining neighbor sets, and a user parameter* min-sup*, the co-location pattern mining problem is to find all co-location patterns, where a label set C is a co-location pattern if $sup(C) \geq$* min-sup*.*

A closely related problem called co-location rule mining problem [8] can be answered easily once we found the co-location patterns. Due to page limit, please refer to our previous work [8] for the details.

Besides, we define the maximal pattern mining problem as follows. Formally, a pattern C is a maximal pattern if there is no superset $C' \supset C$ that is a pattern. For example, if both label sets $C = \{\times\}$ and $C' = \{\times, \circ\}$ are co-location patterns, C must not be a maximal pattern since $C' \supset C$. It is noteworthy that the *closed co-location pattern mining* is not suitable in our setting, where a pattern C is closed if there is no superset $C' \supset C$ that is closed and $sup(C) = sup(C')$, since our Fraction-Score definition usually leads to different support values for a pattern and its subsets.

Problem (Maximal Co-location Pattern Mining)**.** *Given a set O of objects, each with a location, a weight and a label, a distance threshold d for defining neighbor sets, and a user parameter* min-sup*, the maximal co-location pattern mining problem is to find all maximal co-location patterns, where a label set C is a maximal co-location pattern if $sup(C) \geq$* min-sup *and there is no superset $C' \supset C$ that is a pattern.*

4 Co-location Pattern Mining Algorithms

Section 4.1 presents an algorithm for mining the co-location patterns based on Fraction-Score. Section 4.2 details the support computation algorithms. Section 4.3 discusses the problem of deciding whether an object is involved in any row instance of a given label set, and Section 4.4 presents a filtering-and-verification approach for it.

4.1 An Apriori-like Algorithm

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 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 confirmed 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}.

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 [26], [28], [19], [18] could be used for the procedure based on Fraction-Score. First, the procedure based on the partitioning-based approach is the same as that on transaction databases and thus not applicable, Second, that based on the construction-based approach [26] is far from being applicable here since it is based on some heuristics only and involves no concepts of fraction. Third, those based on the enumeration-based and participation-based approaches [28], [19], [18] 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 [30], [32]. In fact, these techniques could be easily adapted to our problem since the supports defined by Fraction-Score satisfy the anti-monotonicity property.

4.2 An Algorithm for Computing the Support

Algorithm 1 FractionComputation(O, T, d, *min-sup*)

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

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 these information.

FractionComputation. Algorithm 1 presents the FractionComputation. 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 $o.w/|Neigh(o,o^{'}.t,d)|$ of o to each object $o^{'}$ in $Disk(o,d)$ (line 9), which is generalized from the unweighted case in [8] that distributes a fraction $1/|Neigh(o,o^{'}.t,d)|$. It then 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 have the following two strategies. First, we do not need to store the fractions of those objects $o \in O_t$ that t have a total weight $W_t \leq$ *min-sup*$/W_{max}$, since these labels t cannot be involved in any co-location pattern. This is a new strategy that cannot be found in [8]. Second, 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, given the fact that the objects within the neighborhood of an object usually involve not that many labels. Based on these strategies, the memory usage for storing the fractions would be much smaller than $O(|O| \cdot |T|)$.

Algorithm 2 SupportComputation(C, O)

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

SupportComputation. Algorithm 2 presents the SupportComputation procedure. 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 for each object $o \in O_t$ which is involved in some row instances of C, it adds up the fraction it receives w.r.t. C, which is computed by the "FractionAggregation" procedure (whose details are presented in Algorithm 3), as $sup(C|t)$. To speed up the additions, if $sup(C|t) > sup(C)$, it terminates the search on t and proceeds with the next label, as $sup(C|t)$ cannot contribute to a smaller $sup(C)$ (lines 4-7). Finally, it returns the smallest $sup(C|t)$ for a label $t \in C$ as $sup(C)$ (lines 8-9).

In practice, we can speed up the procedure if we only need to compute the support of label sets that have at least *min-sup* as follows. Specifically, we keep track of an upper bound of $sup(C|t)$, denoted by $sup(C|t)_{UB}$, by assuming that there is a row instance of C which involves the remaining objects $o \in O_t$. If $sup(C|t)_{UB} <$ *min-sup*, we know that C cannot be a pattern.

The "FractionAggregation" procedure, which for an object o in O, computes 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).

4.3 `OIRI`: 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 an object o is involved

Algorithm 3 FractionAggregation(O, C, o)

Require: an object set O, a label set C, and an object o in O
Ensure: 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** 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** $\Delta_{labelSet}(o, C)$

in any row instance of a given label set C (line 5 in Algorithm 2). We denote this problem by OIRI. Unfortunately, the OIRI problem is NP-hard, which we present in the following theorem.

Theorem 1. *The OIRI 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. The proof can be found in Appendix B. □

4.4 A Filtering-and-Verification Approach for OIRI

A naive method for OIRI 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 [46], [45], [44], 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 OIRI, which involves two phases, namely a filtering phase and a verification phase. The filtering phase is to solve OIRI for easy cases and the verification phase for all remaining cases. The details are introduced as follows.

4.4.1 Filtering Phase

The filtering phase is motivated by the fact that the remaining issue OIRI could be easy to solve with some information re-used in certain cases:

- **Filter 1.** (For $|C| = 2$ only.) Let t' denote the label in $C \setminus \{o.t\}$. We check if $\Delta_{label}(o, t') > 0$. If so, we return "yes". Otherwise, we return "no" (since $\Delta_{label}(o, t') > 0$ if and only if o is involved in a row instance of C). Note that compared to [8], this filter is newly included.
- **Filter 2.** We check if there exists a row instance S of C, which was found previously when answering another OIRI instance for a different object o' and label set C, such that o is involved in S. If so, we 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 3.** We check if all objects in $Disk(o, d)$ together carry all labels in C. If no, we 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 4.** We check if all objects in $Disk(o, d/2)$ together carry all labels in $C - \{o.t\}$. If so, we return "yes" (since there exists a set S of objects in $Disk(o, d/2)$ including o that has $\max_{o,o' \in S} d(o, o') \leq d$ and corresponds to a row instance of C).

4.4.2 Verification Phase

We propose three methods for verification phase as follows.

Dia-CoSKQ-Adapt. This method is based on the close relationship between OIRI and Dia-CoSKQ. In the proof of the NP-hardness of OIRI, we show that any decision problem instance of Dia-CoSKQ could be transformed to a OIRI problem instance. Here, we further show that an arbitrary instance OIRI could be answered by solving a corresponding optimization problem instance of Dia-CoSKQ. Specifically, given an instance of OIRI 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 OIRI is "yes"; otherwise, the answer is "no". Based upon this, we can utilize the exact algorithm proposed in [24] for OIRI. 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 OIRI. 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 object sets that only contain objects corresponding to different labels in C, since other object sets either do 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 prunes the objects that already returned "no" as the answer in the previous iterations for the same label set C. Note that this step is new as compared to [8].
- **Step 3.** it indexes the remaining objects using an *inverted index* which stores the objects using different lists each corresponding to a label and contains all objects with this label.
- **Step 4.** 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"; 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 [47], [48], [15] and is called the *m-closest keywords* (mCK) problem. Based upon this, we can utilize the exact algorithm proposed in [15] for mCK to do this step, and the resulting method corresponds to Optimization-Search. Similar to the Dia-CoSKQ-Adapt method, an early-stopping strategy could be adopted here.

4.4.3 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 $O(n_1 \cdot (C_{range} + k_3^{|C|-2} \cdot |C|^2))$ [24], 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 3. 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 $O(C_{range} + k_1 + n_1 \cdot k_1^{|C|-2})$ [15].

5 Maximal Co-location Pattern Mining

Section 5.1 presents an algorithm for mining the maximal co-location patterns based on Fraction-Score. Section 5.2 details the supports computation algorithm. Section 5.3 analyzes the time complexity.

5.1 An Algorithm for Finding the Maximal Patterns

A straightforward solution to find all maximal patterns is to first find all co-location patterns using the algorithms proposed in Section 4, and then check the maximality of each pattern one by one. This method, however, incurs unnecessary computations as most of the patterns are not maximal and will not be in the result.

To this end, we propose an algorithm that generates the candidate maximal patterns and checks the maximality of each candidate, so it avoids those unnecessary computations as much as possible. It consists of the following steps.

- **Step 1. (Finding size-2 patterns).** We find the size-2 patterns using the algorithms discussed in Section 4, denoted by L_2.
- **Step 2. (Generating Candidate Maximal Patterns).** Inspired by [38], we construct a graph G from L_2, and find the maximal clique to generate the candidate maximal patterns. In particular, each label t correspond to a vertex v in G. If the labels form a size-2 pattern (i.e., can be found in L_2), each pair of vertices is connected by an edge in G. We then find the set CMP of all maximal cliques in G by utilizing the Bron-Kerbosch algorithm [5]. Different from [38] that generate the candidate patterns from size-2 instance table, we do not need to materialize the instances.
- **Step 3. (Finding Maximal Patterns).** We find the maximal patterns MP from the candidate set CMP. The major idea is to iteratively verify the candidate patterns in a descending order of their sizes. If a candidate pattern C is not maximal (i.e., $sup(C) <$ *min-sup*), all its subsets C' with $|C'| = |C| - 1$ are constructed as the candidate patterns to be checked. The iterations stop when $|C'| = m$.

Algorithm 4 shows the maximal pattern mining algorithm. It takes a set O of objects, a label set C as inputs, and finds all maximal patterns and stores in MP. Specifically, it first finds all size-2 patterns, denoted by L_2. Second, it constructs a graph by L_2, and find the set of maximal cliques in G to be the set of candidate maximal patterns CMP (lines 4-5). Third, it iteratively checks each label set C in CMP in descending order of their sizes j, where $2 \le j \le \max_{C \in CMP} |C|$ (lines 7-16). Consider an iteration it processes size j and label set C. If C is a subset of any pattern in the result, we can safely skip C. Otherwise, it invokes the procedure "SupportComputationMaximal" (to be discussed below), which takes a label set C and an object set O as inputs, and computes the support $sup(C)$. If $sup(C) >$ *min-sup*, C is added to MP. Otherwise, it constructs the subsets of C, denoted by C', with $|C'| = j - 1$, and inserts C' into CMP if C' does not exist in CMP. The iterations end when all candidates in CMP have been iterated. Finally, it returns MP as the result.

Algorithm 4 MaximalPatternMining(O, T)

Require: an object set O, a label set T
Ensure: the set of maximal patterns MP

```
1:  MP ← ∅
2:  L2 ← Patterns with size-2                          ▷ Step 1
3:  if L2 = ∅ then return ∅
4:  G ← Construct a graph by L2                        ▷ Step 2
5:  CMP ← Find the set of maximal cliques in G
6:  j ← |T|
7:  while j ≥ 2 do                                     ▷ Step 3
8:      for each candidate C ∈ CMP with |C| = j do
9:          if C is a subset of R ∈ MP then continue;
10:         sup(C) ← SupportComputationMaximal(C, O)
11:         if sup(C) > min-sup then
12:             MP ← MP ∪ {C}
13:         else
14:             for each label set C' ⊂ C with |C'| = j − 1 do
15:                 if C' ∉ CMP then CMP ← CMP ∪ {C'}
16:     j ← j − 1
    return MP
```

Theorem 2. *The MaximalPatternMining algorithm correctly finds all maximal co-location patterns.*

Proof. The completeness can be proven as follows. It is easy to see that all maximal patterns with size-2 can be found

TABLE 3
Datasets used in the experiments

Dataset	# of objects	# of labels	Weighted
UK (Real)	182,334	36	×
NeuroSynth (Real)	507,891	3,229	√
Unweighted Synthetic	94,028	462	×
Weighted Synthetic	94,028	462	√

in Step 1. For maximal patterns with size larger than 2, we show that they must exists in CMP. Specifically, we prove it by contradiction. Suppose there exists a maximal pattern C not in CMP. Then, either (1) there exists a superset of C is in CMP, or (2) there exists a subset $C' \subset C$ with $|C'| = 2$ that is not a pattern. In the former, C is not a maximal pattern by definition. In the latter, C can not be a pattern by the anti-monotonicty property. Both cases lead to contradictions. Thus, all maximal patterns C are in CMP. The correctness is guaranteed as the algorithm calculates the support of each candidate patterns. □

5.2 Algorithm SupportComputationMaximal

In fact, since the definition of $sup(C)$ does not change, we can reuse "SupportComputation" procedure (i.e., Algorithm 2) to calculate the support value of a label set C.

Nevertheless, to further improve the performance, we include an additional filter in the filtering phase of "SupportComputation". The resulting procedure is called "SupportComputationMaximal". In particular, the additional filter takes advantage of the top-down approach in our maximal pattern mining algorithm to reuse information from previous checking. It is inserted after Filter 2, and is as follows.

Filter 2'. We check if there exists a row instance of $C'' \supset C$ involving o for the label set C'' that satisfies $|C''| = |C| + 1$. If so, we return "yes" (since o must also be involved in a row instance of C). To support this checking, we maintain the objects involved in the row instances of each label set with size $(k + 1)$ when we process the label set with size k.

5.3 Time Complexity Analysis

It is easy to see that the time complexity of SupportComputationMaximal is same as that of SupportComputation, denoted by θ. We analyze the time complexity of Algorithm 4 as follows. The complexity of MaximalPatternMining is dominated by Step 3. The complexity is $O((|L_2|+|CMP|)\cdot\theta)$, since it need to compute the supports of at most $(|L_2| + |CMP|)$ label sets.

6 Empirical Studies

Section 6.1 details the experimental set-up. Section 6.2 reports the results on co-location pattern mining, and Section 6.3 presents the results on maximal pattern mining.

6.1 Experimental Set-up

Datasets. We use both real and synthetic datasets, as shown in Table 3. The first real dataset UK is the set of POIs of the United Kingdom[2]. 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 second real dataset NeuroSynth [39] was developed as an automated brain mapping framework that uses text mining to generate a large database of mappings between neural and cognitive states. The database contains a mapping between terms (e.g., "depression" and "anxiety") and the activated locations in the brain (3D coordinates in the MNI stereotaxic space, which we mapped to 3D Euclidean space). It contains 507,891 locations (i.e., objects) with 3,229 terms (i.e., labels). The object weights, obtained from text-mining, are relevance scores between the labels and the locations.

TABLE 4
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

The synthetic datasets are generated by following existing studies [18], [28] 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_loc}, 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 [18], [28], we set the other parameters as shown in Table 4 (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. In addition to the unweighted datasets, we further assign weights to generate weighted datasets. Specifically, we assign each object a weight picked uniformly at random in the range $[0, 1]$ to form the weighted datasets.

Algorithms. For the co-location pattern mining problem, we test our Filtering-and-Verification approach. For comparison, we adapt the Join-less algorithm from [44] 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

2. http://www.pocketgpsworld.com

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.

For the maximal pattern mining problem, we test our MaximalPatternMining algorithm. For comparison, we adapt the SGCT algorithm from [38], which is the state-of-the-art algorithm for maximal co-location pattern mining. Similar to the above, though it is originally designed for participation-based measure, we adapt it for our Fraction-Score measure. Specifically, the adapted algorithm works as follows. First, it finds the size-2 patterns and candidate maximal patterns. Second, for each candidate C, it generates all row instances and stores them in a condensed instance tree. Third, to check whether an object o is involved in C, it checks whether o exists in the tree.

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.

6.2 Experiment Results on Co-location Pattern Mining

6.2.1 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 2. Besides, we use the unweighted synthetic datasets only for the study here since it allows the flexibility to generate the datasets where the ground-truth 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 it by the maximum number of objects that have a specific label in T.

Figure 4 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 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.

We also studied how the fractions in Fraction-Score are distributed. The results showed that only around one-fifth of the patterns have their fractions equal to 1. Due to page limit, please refer to our previous work [8].

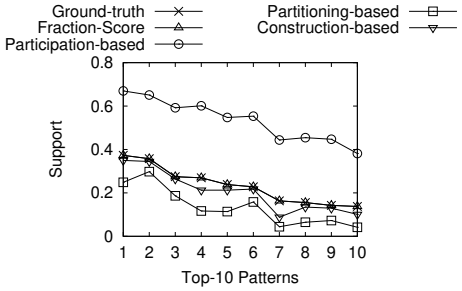


Fig. 4. Support value comparison (unweighted synthetic)

6.2.2 Effectiveness Results on the UK Dataset

We study the effectiveness of different support measures on the UK dataset. Specifically, we ran our algorithm and found the co-location patterns with top-5 supports (with the setting of $d = 1000m$). Table 5 presents the patterns, each with its supports computed by 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 partitioning-based and construction-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).

We also visualized the objects involving the labels in two different patterns. The distributions shown in the visualizations are consistent with our computation results. Due to page limit, please refer to [8].

TABLE 5
Patterns in UK 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

6.2.3 Effectiveness Results on the NeuroSynth Dataset

We further study the effectiveness of our Fraction-Score on the NeuroSynth dataset. Specifically, in the 3D space with $x, y, z \in [-100, 100]$ mapped from the MNI space, we found the co-location patterns by setting $d = 20$. We selected four interesting patterns, and computed their supports with different approaches. Note that the baseline approaches originally do not support weighted dataset, and we adapted them to handle the case with weights. For the adaption details, please refer to Appendix A.

The results are shown in Table 6. According to the results, we found that autism spectrum disorder (ASD) is often correlated to pain, speech and working memory (WM), which conforms with the findings in existing studies [34], [16]. We found that ASD and Parkinson's disease (PD) have similar activated locations in the brain, which is also an ongoing research direction in the medical field [14], [17]. In addition, we have observations on the supports by other approaches similar to above. The supports by the participation-based approach are very close to 1, which decreased the ability to distinguish patterns from label sets.

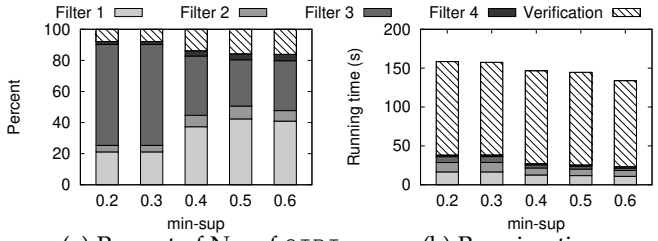
The supports by partitioning-based and construction-based approaches are smaller than those by Fraction-Score.

TABLE 6
Patterns in NeuroSynth dataset

Pattern	Fraction-Score	Partici-pation	Partit-ioning	Constr-uction
{ASD, pain, WM}	0.3466	0.9982	0.1897	0.3206
{ASD, pain, speech}	0.3159	0.9956	0.1535	0.2944
{ASD, PD}	0.3100	0.9975	0.1728	0.2836
{PD, pain, reward, speech, WM}	0.2021	0.9923	0.0965	0.1728

6.2.4 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 5(a), where we vary *min-sup* and measure the percentage of OIRI instances that are found by each of the four filters in the filtering phase and also that by the verification phase. These results show that more than 80% of OIRI instances could be found in the filtering phase, and thus less than 20% OIRI instances would be left in the verification phase. Besides, we notice that when *min-sup* increases, the filtering powers of Filters 1 and 2 increase while that of Filter 3 decreases. The former is because the number of large co-location patterns decreases when *min-sup* increases and as a consequence, it is more likely that size-2 patterns have a larger portion, which benefits Filter 1, and it is easier to find a row instance of a label set, which benefits Filter 2. The latter is because when *min-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 3 decreases. The results on the other datasets provide similar clues and thus they are omitted.



(a) Percent of No. of OIRI (b) Running time

Fig. 5. Effectiveness of the filtering phase (unweighted synthetic)

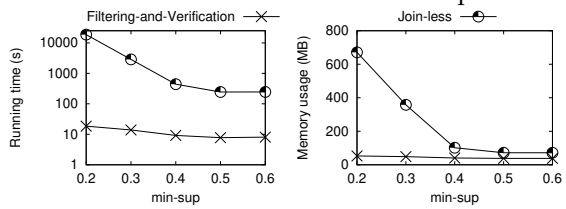
Verification phase. We conducted experiments on both real and synthetic datasets for studying the performance of the three methods proposed for the verification phase. The results can be found in Appendix C due to page limit. According to the results, Combinatorial-Search runs the fastest consistently 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 OIRI 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 5(b).

We also compared the overall improvement of the Filtering-and-Verification approach to the one proposed in [8]. The results can be found in Appendix D. According to the results, the updated Filtering-and-Verification algorithm runs faster and uses fewer memory in most cases, which demonstrates the effectiveness of the additional filtering and pruning steps and the strategy to reduce memory usage.

6.2.5 Filtering-and-Verification vs State-of-the-art

In this part, we compare the performance between Filtering-and-Verification and Join-less [44], in terms of running time and memory consumption.

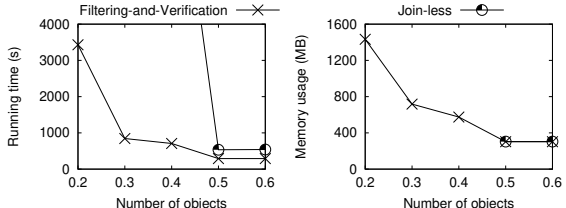
Effect of *min-sup*. Figure 6 shows the results on the real dataset where we vary *min-sup*. According to Figure 6(a), the running times of both algorithms decrease when *min-sup* increases. This is because fewer co-location patterns would be found when *min-sup* increases. Besides, our Filtering-and-Verification 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 6(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. 6. Effect of *min-sup* (UK)

Figure 7 shows the results on the NeuroSynth dataset where we vary *min-sup*, where the results for Join-less with *min-sup* ≤ 0.4 are not shown because it takes more than 1 day to run. According to Figure 7(a), the running times of both algorithms decrease when *min-sup* increases. Our Filtering-and-Verification approach runs faster than the Join-less method, which is because we only check if the objects are involved in any row instances, while Join-less finds all row instance for each pattern. According to Figure 7(b), our Filtering-and-Verification approach consumes less memory than the Join-less method, since the Join-less method needs to store all row instances of the patterns. The results on the synthetic datasets, where we vary other parameter settings, can be found in Appendix E.



(a) Running time (b) Memory usage

Fig. 7. Effect of *min-sup* (NeuroSynth)

6.2.6 Scalability Test

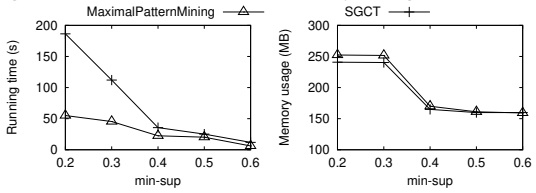
We further generated 5 synthetic datasets with sizes $\{180k, 360k, 540k, 720k, 900k\}$ from the real dataset for scalability test. According to the results, our Filtering-and-Verification method could scale up on large datasets of

size 1M, while the Join-less method cannot scale to large datasets, e.g., it ran for more than 2 days on dataset of size about 180k. The results can be found in Appendix F.

6.3 Experiment Results on Maximal Pattern Mining

In this part, we compare the performance between our MaximalPatternMining algorithm and SGCT [38], in both running time and memory consumption.

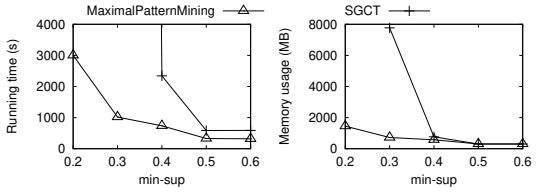
Figure 8 shows the results on the weighted synthetic dataset where we vary *min-sup*. According to Figure 8(a), the runnning times of both algorithms decrease when *min-sup* increases. This is because fewer co-location pattern exists and thus the sizes of the maximal patterns would decrease. Besides, our MaximalPatternMining runs much faster than the SGCT method, which is because (1) our two-phases approaches prune more non-promising candidates, and (2) we do not need to generate and store all row instances, while SGCT materializes all of them. According to Figure 8(b), the two algorithms have similar memory usage.



(a) Running time (b) Memory usage

Fig. 8. Effect of *min-sup* on maximal pattern mining (weighted synthetic)

Figure 9 shows the results on the NeuroSynth dataset, where the results for SGCT with *min-sup* < 0.4 are not shown because it takes more than 1 day to run. According to Figure 9(a), our MaximalPatternMining runs consistently faster than SGCT, which is because of MaximalPatternMining has more effective prunings to reduce the number of candidate patterns. The results for the unweighted synthetic and UK datasets can be found in Appendix G.



(a) Running time (b) Memory usage

Fig. 9. Effect of *min-sup* on maximal pattern mining (NeuroSynth)

Summary of Results. Our Fraction-Score metric 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% `OIRI` 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. Moreover, our MaximalPatternMining algorithm runs faster than the state-of-the-art.

7 Conclusion

In this paper, we studied the co-location pattern mining problem. We showed the weaknesses of the existing support measures, and proposed Fraction-Score which quantifies the prevalence properly. We proposed an Apriori-like algorithm for mining co-location patterns based on Fraction-Score. We developed a filtering-and-verification algorithm for an operation of deciding whether an object is involved in a row instance of a label set, which is proved to be NP-hard. We also studied the maximal co-location pattern mining problem based on Fraction-Score, and develop an efficient method for the mining task. We conducted experiments on both real and synthetic datasets, which verified that Fraction-Score measures the prevalence better than existing approaches and our algorithms run significantly faster than the adaption of state-of-the-arts.

In the future, we plan to study the co-location pattern mining problem on spatio-temporal data, where a time dimension is taken into consideration. This problem is interesting since some patterns occur only at certain time stamps. It is also interesting to develop parallel algorithms on GPU for mining co-location patterns based on Fraction-Score.

Acknowledgements: The research of Raymond Chi-Wing Wong is supported by GZSTI16EG24. The research of Da Yan is supported by Alabama Research and Development Enhancement Fund 1ARDEF21 03 and NSF OAC-2106461. This research is also supported by the Ministry of Education, Singapore, under its Academic Research Fund (Tier 2 Award MOE-T2EP20221-0013 and Tier 2 Award MOE-T2EP20220-0011). Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not reflect the views of the Ministry of Education, Singapore.

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. *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] C. Bron and J. Kerbosch. Algorithm 457: finding all cliques of an undirected graph. *Communications of the ACM*, 16(9):575–577, 1973.
[6] M. Celik, J. M. Kang, and S. Shekhar. Zonal co-location pattern discovery with dynamic parameters. In *ICDM*, pages 433–438. IEEE, 2007.
[7] H. K.-H. Chan, C. Long, and R. C.-W. Wong. On generalizing collective spatial keyword queries. *TKDE*, 30(9):1712–1726, 2018.
[8] H. K.-H. Chan, C. Long, D. Yan, and R. C.-W. Wong. Fraction-score: a new support measure for co-location pattern mining. In *ICDE*, pages 1514–1525. IEEE, 2019.
[9] N. Cressie. Statistics for spatial data. *Terra Nova*, 4(5):613–617, 1992.
[10] 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.
[11] 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.
[12] 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.
[13] V. Estivill-Castrol and A. T. Murray. Discovering associations in spatial data—an efficient medoid based approach. In *PAKDD*, pages 110–121. Springer, 1998.
[14] H. M. Geurts, G. A. McQuaid, S. Begeer, and G. L. Wallace. Self-reported parkinsonism features in older autistic adults: A descriptive study. *Autism*, 26(1):217–229, 2022.
[15] T. Guo, X. Cao, and G. Cong. Efficient algorithms for answering the m-closest keywords query. In *SIGMOD*, pages 405–418. ACM, 2015.

[16] A. Habib, L. Harris, F. Pollick, and C. Melville. A meta-analysis of working memory in individuals with autism spectrum disorders. *PloS one*, 14(4):e0216198, 2019.

[17] B. N. Hand, A. M. Angell, L. Harris, and L. A. Carpenter. Prevalence of physical and mental health conditions in medicare-enrolled, autistic older adults. *Autism*, 24(3):755–764, 2020.

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

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

[20] Y. Huang, L. Zhang, and P. Yu. Can we apply projection based frequent pattern mining paradigm to spatial co-location mining? In *PAKDD*, pages 719–725. Springer, 2005.

[21] 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.

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

[23] 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.

[24] 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.

[25] y. Long, P. Yang, and L. Wang. Mining significant co-location patterns from spatial regional objects. In *MDM*, pages 479–484. IEEE, 2019.

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

[27] 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.

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

[29] 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.

[30] T. Uno, M. Kiyomi, and H. Arimura. Lcm ver. 3: collaboration of array, bitmap and prefix tree for frequent itemset mining. In *OSDM*, pages 77–86. ACM, 2005.

[31] L. Wang, X. Bao, and L. Zhou. Redundancy reduction for prevalent co-location patterns. *TKDE*, 30(1):142–155, 2017.

[32] 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.

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

[34] Y. Wang, Y.-b. Zhang, L.-l. Liu, J.-f. Cui, J. Wang, D. H. Shum, T. van Amelsvoort, and R. C. Chan. A meta-analysis of working memory impairments in autism spectrum disorders. *Neuropsychology review*, 27(1):46–61, 2017.

[35] 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.

[36] P. Yang, L. Wang, X. Wang, and D. Fang. An effective approach on mining co-location patterns from spatial databases with rare features. In *MDM*, pages 53–62. IEEE, 2019.

[37] 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.

[38] X. Yao, L. Peng, L. Yang, and T. Chi. A fast space-saving algorithm for maximal co-location pattern mining. *Expert Systems with Applications*, 63:310–323, 2016.

[39] T. Yarkoni, R. A. Poldrack, T. E. Nichols, D. C. Van Essen, and T. D. Wager. Large-scale automated synthesis of human functional neuroimaging data. *Nature methods*, 8(8):665–670, 2011.

[40] J. S. Yoo and D. Boulware. Incremental and parallel spatial association mining. In *Big Data*, pages 75–76. IEEE, 2014.

[41] 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.

[42] J. S. Yoo and M. Bow. Mining maximal co-located event sets. In *PAKDD*, pages 351–362. Springer, 2011.

[43] J. S. Yoo and M. Bow. Mining top-k closed co-location patterns. In *ICSDM*, pages 100–105. IEEE, 2011.

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

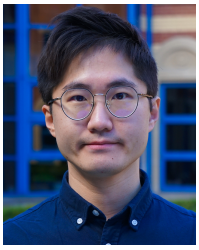
[45] J. S. Yoo, S. Shekhar, and M. Celik. A join-less approach for co-location pattern mining: A summary of results. In *ICDM*. IEEE, 2005.

[46] 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.

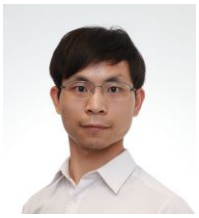
[47] D. Zhang, Y. M. Chee, A. Mondal, A. K. Tung, and M. Kitsuregawa. Keyword search in spatial databases: Towards searching by document. In *ICDE*, pages 688–699. IEEE, 2009.

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

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



Harry Kai-Ho Chan received the BEng, MPhil and PhD degrees in computer science and engineering from the Hong Kong University of Science and Technology. He is a Lecturer in Data Science with Information School, University of Sheffield, United Kingdom. His research interests include spatio-temporal data management, data mining, and data science. He has served on the program committees for conferences such as KDD, ICDE, CIKM and ACM SIGSPATIAL.



Cheng Long (S'11-M'15-SM'22) is currently an Assistant Professor at the School of Computer Science and Engineering, Nanyang Technological University. He received his PhD degree from the Hong Kong University of Science and Technology, Hong Kong, in 2015, and his BEng degree from South China University of Technology, China, in 2010. His research interests are broadly in data management, data mining and big data analytics.

Da Yan is currently an Assistant Professor of the Department of Computer Sciences at the University of Alabama at Birmingham (UAB). His research interests include parallel and distributed systems for big data analytics, graph data management, geo-spatial data management, data mining and machine learning. Dr. Yan regularly publishes in first-tier venues such as SIGMOD, VLDB, KDD, ICDE, VLDB Journal, TKDE, TPDS, etc., where he also regularly serves as reviewers. Dr. Yan was the sole winner of Hong Kong 2015 Young Scientist Award in Physical/Mathematical Science, and a senior member of ACM and IEEE.

Raymond Chi-Wing Wong received the BSc, MPhil and PhD degrees in computer science and engineering from the Chinese University of Hong Kong (CUHK) in 2002, 2004, and 2008, respectively. He is a professor of the Department of Computer Science and Engineering, the Hong Kong University of Science and Technology. His research interests include database and data mining.

Hua Lu is a Professor of Computer Science in the Department of People and Technology, Roskilde University, Denmark. He received the BSc and MSc degrees from Peking University, China, and the PhD degree in computer science from National University of Singapore. His research interests include data management, spatial data, location-based services, data science and GIS. He has served as PC cochair or vice chair for ISA 2011, MUE 2011, MDM 2012, NDBC 2019 and IEEE BigData 2022, demo chair for SSDBM 2014, and PhD forum cochair for MDM 2016 and MDM 2022. He has served on the program committees for conferences such as VLDB, ICDE, KDD, WWW, CIKM, DASFAA, ACM SIGSPATIAL, SSTD and MDM. He received the Best Vision Paper Award at SSTD 2019. He is a senior member of the IEEE.

Fraction-Score: A Generalized Support Measure for Weighted and Maximal Co-location Pattern Mining (Appendix)

APPENDIX A
ADAPTION OF EXISTING SUPPORT MEASURES FOR WEIGHTED DATASET

Given a label set C, we calculate the support of C by different approaches as follows.

Participation-based Approach. Instead of counting each group as 1 as in the unweighted version, we use the following adapted equation for participation ratio of a label t in C.

$$PR(t|C) = \frac{\sum_{o \in Obj(t,C)} o.w}{\sum_{o \in O_t} o.w} \quad (6)$$

where $Obj(t, C)$ is the set of objects o which has the label t and there are some row instances of C involving o.

Correspondingly, we utilize the same definition of participation index as follows.

$$PI(C) = \min_{t \in C} PR(t|C) \quad (7)$$

Partitioning-based Approach. We partition the space by $d \times d$ grids. For each grid, we calculate the *count* of C in the grid by the minimum weighted object that contains any label in C, multiplied by the minimum count of the number of objects for each label in C. The sum of these counts from all grids are used as the support of C.

Construction-based Approach. We arbitrary construct row instances for C. For each constructed row instance, we use the minimum weight among the objects in the row instance as the *contribution* of this row instance, such that it can capture the worse-case scenario. The sum of these contributions among all row instances are used as the support of C.

APPENDIX B
NP-HARDNESS PROOF OF THE OIRI PROBLEM

Proof. We prove by reduction from the existing Collective Spatial Keyword Query with the Diameter cost function (Dia-CoSKQ) problem [24], [7] which is NP-hard.

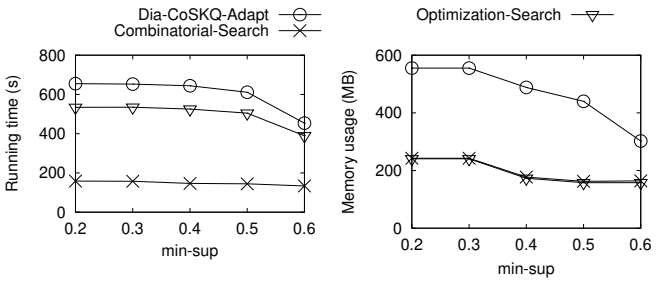
We first give the formal definition of the decision problem of Dia-CoSKQ. Given a set D of POIs where each POI p has a location $p.\lambda$ and a set of keywords $p.\psi$ and a query q with a query location $q.\lambda$, a set of query keywords $q.\psi$, and a real number c, the decision problem of Dia-CoSKQ 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.

We next transform a given decision problem instance of Dia-CoSKQ to a OIRI 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 creating another object o with its location as $q.\lambda$ and its label as a fictious one t. 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 OIRI problem is to decide whether there exists a row instance of C which involves q. Clearly, the above transformation step is in polynomial time.

It could be easily verified that the decision problem of Dia-CoSKQ is equivalent to that of OIRI. □

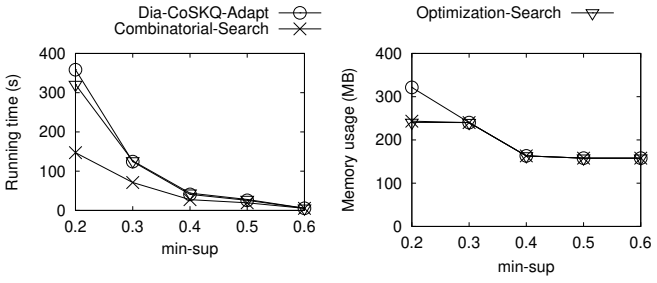
APPENDIX C
EXPERIMENTAL RESULTS ON VERIFICATION PHASE

In this part, we show the results reflecting the performance of three proposed methods for the verification phase. Figure 10 and Figure 11 show the results on the unweighted and weighted synthetic datasets, respectively, where we vary *min-sup*. Figure 12 and Figure 13 show the results on the UK and NeuroSynth dataset, respectively. According to the results, Combinatorial-Search runs the fastest consistently under all datasets. 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 problems (i.e., Dia-CoSKQ and mCK problems) while OIRI is a decision problem. These exact algorithms involve extra steps for finding an optimal solution and thus they take more time.



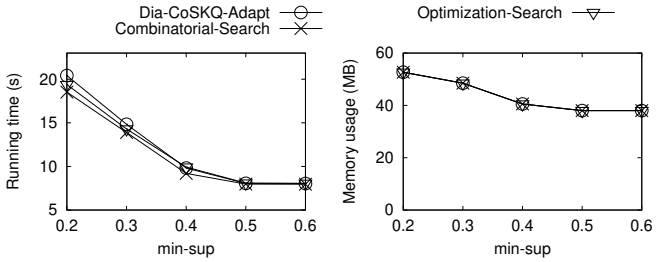
(a) Running time (b) Memory usage

Fig. 10. Effect of *min-sup* (unweighted synthetic)



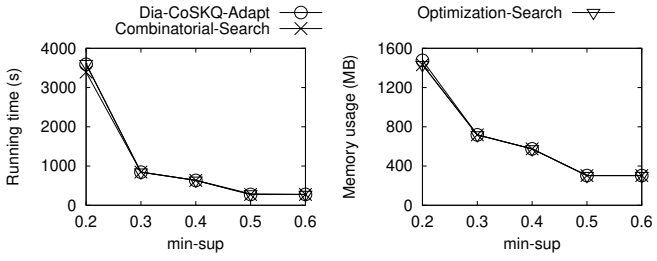
(a) Running time (b) Memory usage

Fig. 11. Effect of *min-sup* (weighted synthetic)



(a) Running time (b) Memory usage

Fig. 12. Effect of *min-sup* (UK)



(a) Running time (b) Memory usage

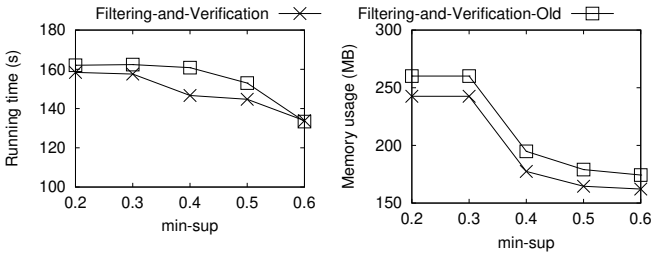
Fig. 13. Effect of *min-sup* (NeuroSynth)

APPENDIX D
EXPERIMENTAL RESULTS ON IMPROVED FILTERING-AND-VERIFICATION ALGORITHM

In this part, we show the results comparing the performance of the improved Filtering-and-Verification algorithm and the one developed in [8] (Filtering-and-Verification-Old). The Combinatorial-Search are used here as it is the fastest one among the three verification methods. Figure 14 and Figure 15 show the results on the unweighted and weighted synthetic datasets, respectively, where we vary *min-sup*. Figure 16 and Figure 17 show the results on the UK and NeuroSynth dataset, respectively. According to the results, the updated Filtering-and-Verification algorithm runs faster and uses fewer memory than the old one in most cases, which demonstrates the effectiveness of the additional filtering and pruning steps and the strategy to reduce memory usage.

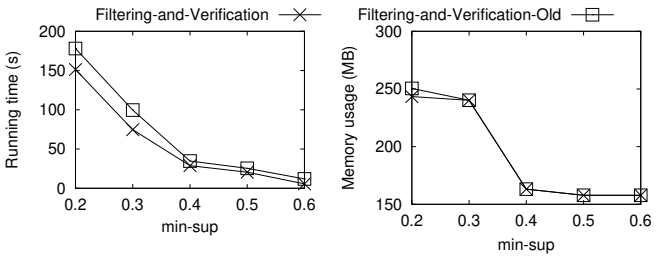
APPENDIX E
EXPERIMENTAL RESULTS ON UNWEIGHTED AND WEIGHTED SYNTHETIC DATASETS

Effect of λ_2. Figure 18 and Figure 19 show the results on the unweighted and weighted synthetic datasets, respectively, where we vary λ_2. According to the results, the algorithms have the same trends in both running time and memory consumptions for both datasets. Specifically, 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. Moreover, the memory consumptions of both algorithms increase with λ_2. This is because the datasets would involve more objects when λ_2 increases. The



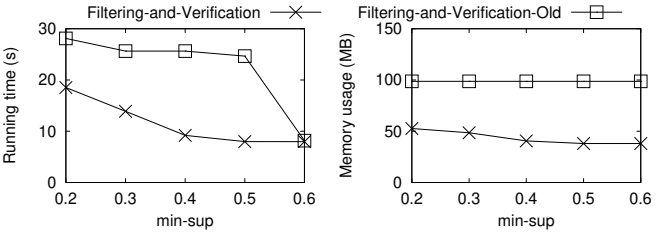
(a) Running time (b) Memory usage

Fig. 14. Effect of *min-sup* (unweighted synthetic)



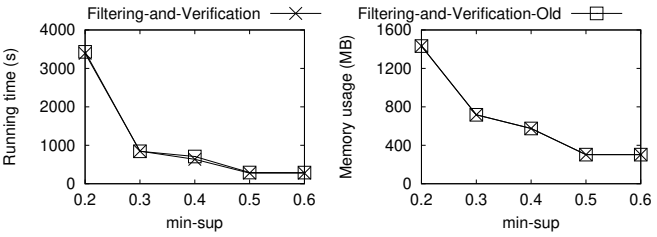
(a) Running time (b) Memory usage

Fig. 15. Effect of *min-sup* (weighted synthetic)



(a) Running time (b) Memory usage

Fig. 16. Effect of *min-sup* (UK)

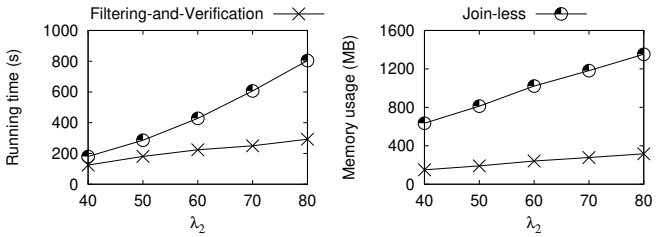


(a) Running time (b) Memory usage

Fig. 17. Effect of *min-sup* (NeuroSynth)

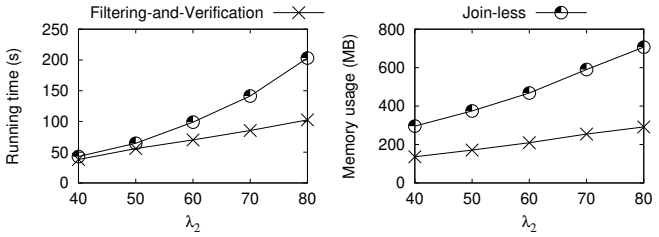
memory usage of our Filtering-and-Verification approach is much smaller than that of the Join-less method consistently. For example, when $\lambda_2 = 80$ in the unweighted dataset, filtering-and-verification approach only consumes less than 400MB of memory, while join-less method consumes more than 1200MB of memory, which is 3 times more.

Effect of m_{clump}. Figure 20 and Figure 21 show the results on the synthetic datasets where we vary m_{clump}. The results of the Join-less method for $m_{clump} = 5$ are missing which is



(a) Running time (b) Memory usage

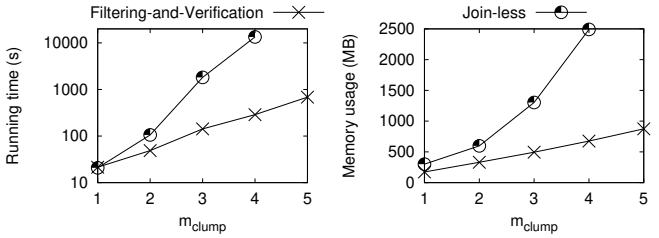
Fig. 18. Effect of λ_2 (unweighted synthetic)



(a) Running time (b) Memory usage

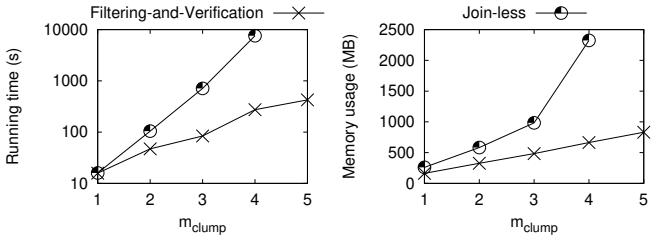
Fig. 19. Effect of λ_2 (weighted synthetic)

simply because it ran for a very long time, i.e., more than 6 hours (this time upper bound applies for all the following results). According to the results, 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. Moreover, 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

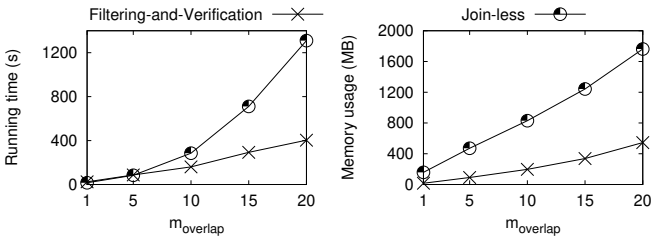
Fig. 20. Effect of m_{clump} (unweighted synthetic)



(a) Running time (b) Memory usage

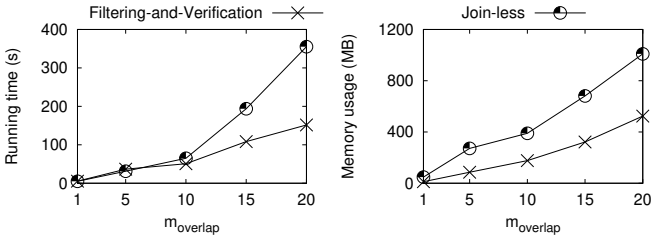
Fig. 21. Effect of m_{clump} (weighted synthetic)

Effect of $m_{overlap}$. Figure 22 and Figure 23 show the results on synthetic datasets when we vary $m_{overlap}$. According to the results, the running times of both algorithms increase when $m_{overlap}$ increases and our Filtering-and-Verification approach outperforms the Join-less method. In addition, 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. 22. Effect of $m_{overlap}$ (unweighted synthetic)

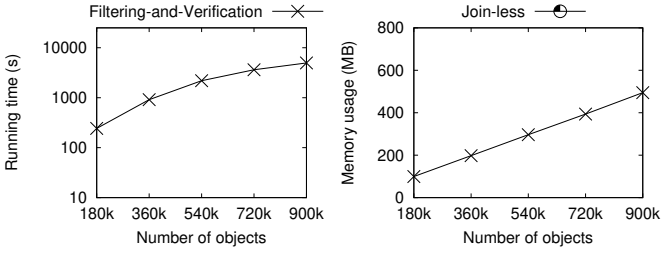


(a) Running time (b) Memory usage

Fig. 23. Effect of $m_{overlap}$ (weighted synthetic)

APPENDIX F
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 24 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.

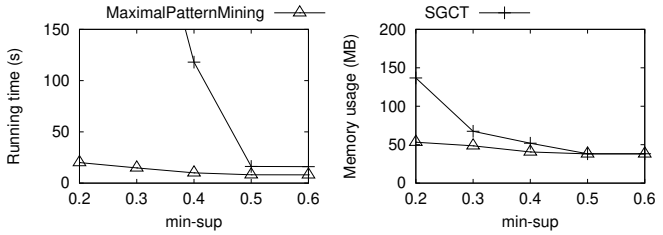


(a) Running time (b) Memory usage

Fig. 24. Scalability test

APPENDIX G
EXPERIMENTAL RESULTS ON MAXIMAL PATTERN MINING

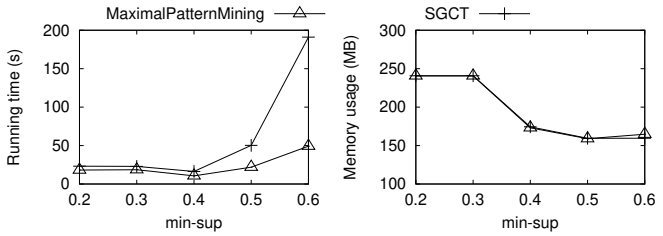
Figure 25 shows the results on the UK dataset, where we vary *min-sup*. According to Figure 25, the running times and memory consumptions of both algorithms decrease when *min-sup* increases. This is because fewer maximal co-location pattern candidates exists when *min-sup* is large, and thus fewer time and space are needed to process and store these candidates. Also, our MaximalPatternMining runs significantly faster than SGCT, especially when *min-sup* is small, since MaximalPatternMining has more aggressive prunings to remove those unpromising row instances for the candidates, and thus reducing the processing time needed.



(a) Running time (b) Memory usage

Fig. 25. Effect of *min-sup* on maximal pattern mining (UK)

Figure 26 shows the results on the unweighted synthetic dataset, where we vary *min-sup*. According to Figure 26(a), the running times of both algorithms increase when *min-sup* increases. We investigated the reason for this trend. It is because when *min-sup* increase, the number of maximal pattern candidates increases. To illustrate, consider a maximal pattern candidate C with size k. If C is not a maximal pattern (after checking by the algorithms), all its proper subsets that has a size $k-1$ will become the maximal pattern candidates. In this case, the algorithms need to check the maximality of each candidate, and thus resulting a larger running time. Note that this situation can happen recursively until $k = 2$. Still, our MaximalPatternMining performs much better than SGCT, which is becasuse MaximalPatternMining has the additional filter that utilizes the previous checking results to speed up the checking, and thus reducing the total processing time. Besides, the memory consumptions of both algorithms are similar, and decrease when *min-sup* increase.



(a) Running time (b) Memory usage

Fig. 26. Effect of *min-sup* on maximal pattern mining (unweighted synthetic)