Cohesion Based Co-location Pattern Mining

Cheng Zhou, Boris Cule, and Bart Goethals
University of Antwerp, Belgium
Email: {cheng.zhou, boris.cule, bart.goethals}@uantwerpen.be

***Abstract*—Because of a wide range of applications, e.g., GPS applications and location based services, spatial pattern discovery is an important task in data mining. A co-location pattern is defined as a subset of spatial items whose instances are often located together in spatial proximity. Current co-location mining algorithms are unable to quantify the spatial proximity of a co-location pattern. We propose a co-location pattern miner aiming to discover co-location patterns in a multidimensional spatial structure by measuring the cohesion of a pattern. We present two ways to build the co-location pattern miner, *FromOne* and *FromAll*, in an attempt to find a balance between accuracy and runtime. Additionally, we propose a method named *Fre-ball* to transform a structure into a transaction database, after which any existing itemset mining algorithm can be used to find the co-location patterns. An experimental evaluation shows that *FromOne* and *Fre-ball* are more efficient than existing methods. The usefulness of our methods is demonstrated by applying them on the publicly available geographical data of the city of Antwerp in Belgium.**

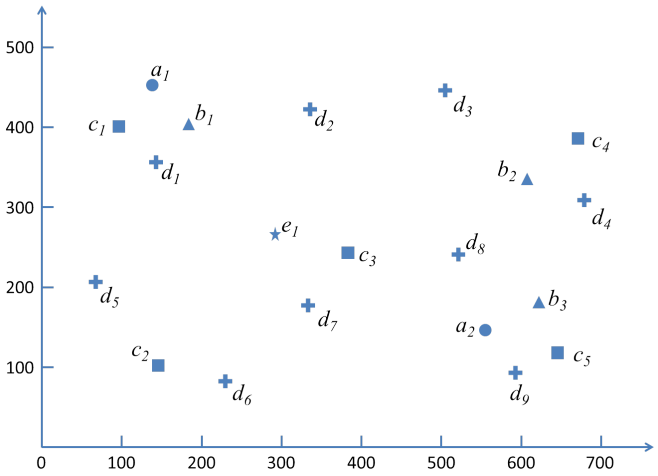

Figure 1. An example of a 2-dimensional structure.

I. Introduction

With the boom in the availability of spatial data, spatial data mining, i.e., discovering interesting and previously unknown but potentially useful patterns from large spatial datasets, has become a popular field. A co-location pattern represents a subset of spatial items that frequently appear together in spatial proximity. Spatial co-location patterns may yield important insights for many applications, including city planning, mobile commerce, earth science, biology, transportation, etc. For example, location based service providers are very eager to know what services are requested frequently together and located in spatial proximity. This information can help them improve the effectiveness of their location based recommendation system where users request a service in a nearby location and enable the use of pre-fetching to speed up service delivery. Therefore, spatial co-location pattern mining is one of the most crucial spatial data mining tasks.

Figure 1 shows a 2-dimensional structure consisting of four different items, a, b, c, and d. Each item is represented by a distinct shape. The subscript indexes next to the items are only used to identify individual instances of each item, to avoid having to explicitly refer to their coordinates.

A typical approach for mining co-location patterns is proposed by Shekhar and Huang [1], [2]. This method first identifies co-location patterns of size 1 and 2. In our example, given a neighbourhood distance threshold of 100, the approach identifies the neighbourhood relationship of all point pairs, as depicted by solid lines in Figure 2. A clique represents an instance of a set of items located in the same neighbourhood. For example, in Figure 2, $\{a_2, b_3, d_9\}$ is an instance of itemset $\{a, b, d\}$, but $\{a_2, b_3, d_8\}$ is not. Next, the approach generates candidates of size $n+1$ ($n \geq 2$) and tests the prevalence of each candidate to identify co-location patterns of size $n+1$. For each candidate of size $n+1$, the method first joins the instances of two of its subset co-location patterns of size n that share the first $n-1$ items to identify its instances. The proposed algorithm then computes the prevalence of the candidate based on the identified instances. If the prevalence of the candidate is not lower than a user-defined prevalence threshold, it is identified as a co-location pattern. The prevalence $prev(P)$ of a pattern P is defined as $prev(P) = \min\{pr(t_i, P), t_i \in P\}$, where $pr(t_i, P)$ is the participation ratio of an item t_i in pattern P, which is defined as

$$pr(t_i, P) = \frac{\text{\# instances of } t_i \text{ in any instance of } P}{\text{\# instances of } t_i}.$$

For example, as shown in Figure 2, $pr(a, \{a, d\}) = \frac{2}{2} = 1$, which reflects the fact that 100% of instances of a (i.e., a_1 and a_2) participate in some instances of pattern $\{a, d\}$ (i.e., $\{a_1, d_1\}$, $\{a_2, d_8\}$ and $\{a_2, d_9\}$). The prevalence of pattern $\{a, b, c\}$ is 0.4, since $pr(a, \{a, b, c\}) = 1$, $pr(b, \{a, b, c\}) = \frac{2}{3}$ and $pr(c, \{a, b, c\}) = 0.4$.

A number of more efficient co-location mining algorithms [3], [4], [5] using the same prevalence measure have been proposed afterwards. However, it has been shown that a prevalence threshold based mining approach may fail to find true patterns and may even report meaningless patterns [6]. The prevalence measure value of a set of items can be low, if one participating item has a high participation ratio, but other participating items have low participating ratios due to their large number of ocurrences. Overall, the minimum

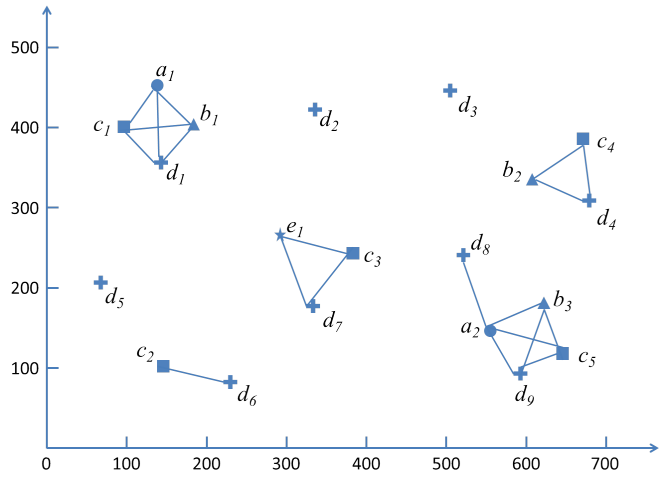

Figure 2. The neighbourhood relationship.

participation value of such patterns will be low and they will be ignored by the existing co-location mining approaches.

Furthermore, the method proposed by Barua and Sander [6], as well the prevalence based methods [1], [2], [3], [4], [5] search for meaningful patterns for a given proximity neighbourhood. Neighbourhood information is given in the form a neighbourhood threshold which is the maximum distance between instances of any two participating items of a pattern. Therefore, there are actually two user-defined thresholds, i.e., a prevalence threshold and a neighbourhood threshold. As a result, a random pattern can attain a high prevalence value, when a large neighbourhood threshold is used by the existing algorithms and be reported as prevalent. A random pattern may also have a high prevalence value with a smaller neighbourhood threshold if the participating items are abundant.

Motivated by the above observations, we propose a cohesion based co-location pattern miner. When looking for co-location patterns, the spatial proximity of the items making up the pattern is important. We measure the spatial proximity of a pattern by defining the cohesive radius of a pattern. The cohesive radius measures the average size of the spatial areas in which the minimal occurrences of the pattern are located. The main benefit of this method is that it allows us to quantify the spatial proximity of a pattern, requiring the user to specify only a cohesive radius threshold. We develop a cohesive itemset miner and explore its potential to find co-location patterns. Additionally, we propose a model for transforming a spatial structure into a transaction database, then any existing itemset mining algorithm can be used to find the co-location patterns.

The rest of the paper is organised as follows. We formally describe the problem setting for finding co-location patterns in Section II. In Section III, we present our algorithm for generating co-location patterns. In Section IV, we propose a frequent itemset mining based method to find co-location patterns. Section V demonstrates the effectiveness of our algorithm applied on the open data of Antwerp and Section VI provides an overview of the existing related work. We end the paper with a summary of our conclusions in Section VII.

II. Problem Setting

We consider an *n*-dimensional structure S as a set of points where a point v is a pair (t, c) consisting of an item $t \in I$, and an *n*-dimensional coordinate $c \in \mathbb{R}^n$, where I is the set of all possible items in S and $n \geq 1$. As can be seen in Figure 1, an item t may occur many times at different positions in a structure S. Thus there may be many points containing t in S and we denote such points as V_t, e.g., $V_a = \{a_1, a_2\}$. We denote the frequency of item t as $Fre(t) = |V_t|$. We denote the structure by $S = \{v_1, \ldots, v_l\}$, where $|S| = l$ is the number of points in the structure, i.e., the size of the structure.

We base our work on an earlier work on mining spatially cohesive itemsets in a dataset consisting of many three-dimensional protein structures [7], we propose a new co-location pattern mining algorithm to find cohesive itemsets from a single structure. We are specifically investigating patterns of items occurring spatially in close proximity. To do this, we will define co-location patterns in terms of cohesion making it possible to find itemsets consisting of items that, on average, appear close to each other.

A. Cohesive Radius

Given a set of points $V = v_1, \ldots, v_q$, let $MiniBall(V)$ denote the ball with the smallest radius that contains V, namely the *smallest enclosing ball*. It has been shown that $MiniBall(V)$ always exists and is unique [8]. Intuitively, we consider the points V in *n*-dimensional space cohesive if the radius of $MiniBall(V)$ is small enough. Therefore, we will measure the *cohesion* of an itemset X in a given structure S by computing the average radius of the smallest enclosing balls containing X in S. We call this computed average the *cohesive radius* of X in S.

Given an itemset $X = \{t_1, \ldots, t_m\}$, assume that each item t_i occurs n_i times in structure S. In order to compute the exact average radius of the smallest enclosing balls containing X in S, we need to find such smallest balls for each occurrence of an item in X. In other words, for an item t_i, we need to find n_i smallest balls. To do this, for each occurrence of t_i we need to examine each possible ball (i.e., the smallest ball for each of the possible combinations of occurrences of all other items in X) in order to find the one with the minimal radius. Repeating this for every item in X, requires finding $m \prod_{i=1}^{m} n_i$ balls, which is computationally expensive. As a result, we propose two different ways to approximate this process, *FromOne* and *FromAll*.

Intuitively, points that occur near to each other are more likely to produce the smallest enclosing ball than those far apart. Therefore, rather than looking at all possible combinations, we limit our search to a selection of points. We propose an algorithm (as described in Algorithm 1) to first find a nearest combination B_v containing a point $v = (t_i, c)$ (c is the *n*-dimensional coordinate of the point and $i \in \{1, \ldots, m\}$) and occurrences of all other items of X nearest to v, and then find the smallest enclosing ball of this nearest combination.

Lemma 1. *Given itemset $X = \{t_1, \ldots, t_m\}$, and a point $v = (t_i, c)$, with $1 \leq i \leq m$, the radius of the smallest enclosing ball around v containing all items of X as found*

Algorithm 1: Finding the smallest enclosing ball of a nearest combination

Input : itemset $X = \{t_1, \ldots, t_m\}$, point $v = (t_i, c)$ with $1 \leq i \leq m$
Output: a smallest enclosing ball $MiniBall(B_v)$

```
B_v = ∅;
for j = 1, ..., m do
    B_v = B_v ∪ arg min_{w ∈ V_{t_j}} D(w, v)
return MiniBall(B_v);
// D(w, v) is the Euclidean distance between w and v
```

by Algorithm 1, can never be more than twice as large as the radius of the exact smallest enclosing ball containing v and all items in X.

Proof: Denote the radius of the exact smallest enclosing ball containing v and all items in X with $R_v(X)$. Given a data point v, Algorithm 1 will approximate this value by computing the radius of the smallest ball enclosing v and the occurrences of all other items in X closest to v. Since we know that there exists a ball with radius $R_v(X)$, containing v and all items in X, we can conclude that, for any item t_j in X, the maximal distance from v to the nearest occurrence of t_j cannot be larger than $2 \times R_v(X)$. Therefore, a ball of radius $2 \times R_v(X)$ with v as its centre will contain at least one occurrence of each item in X. It follows that the smallest ball containing v and the nearest occurrences of each item in X must have a radius that is at most $2 \times R_v(X)$. ■

1) FromOne: In our first approach, we approximate the process of finding the smallest enclosing ball of an itemset as follows:

1. select item t_1 from $X = \{t_1, \ldots, t_m\}$ (items can be sorted by different strategies, as will be discussed later), and for each point $v_j \in V_{t_1},\ j = 1, 2, \ldots, |V_{t_1}|$, we find $MiniBall(B_{v_j})$ as described in Algorithm 1 and get its radius $r_j(X)$.

2. we denote the cohesive radius of X in a structure S as

$$R'(X) = \frac{\sum_{v_j \in V_{t_1}} r_j(X)}{|V_{t_1}|}. \tag{1}$$

There are only $|V_{t_1}|$ smallest enclosing balls to find in a structure S, much fewer than if we tried to find the exact smallest enclosing ball for each occurrence of t_1 in S, resulting in a considerable reduction in time complexity.

However, this procedure inevitably results in approximation errors. For example, as illustrated in Figure 3, assume we are evaluating itemset abc, and we picked item a as the first item. We look for the nearest b and the nearest c, and find b_1 and c_1, which are closer to a_1 than b_2 and c_2, respectively, resulting in the ball drawn with a dashed line. However, the smallest possible ball containing a, b and c is much smaller, and is depicted using a solid line. In practice, such cases are rarely encountered. Therefore, this approximate method is capable of producing reasonably accurate results, as will be demonstrated in Section V.

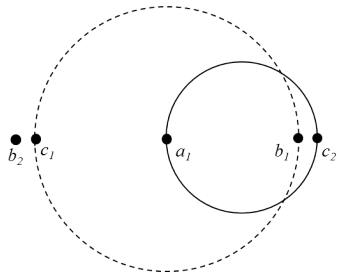

Figure 3. Approximation error made by *FromOne*.

2) FromAll: As can be seen above, *FromOne* can, in theory, result in relatively large approximation errors. However, in Figure 3, if we also try to find the smallest enclosing balls for occurrences of other items in itemset abc (and not just a), the average error will be greatly reduced. We therefore develop another way to decrease the approximation error as follows.

1. for each point $v_j \in V(X),\ j = 1, 2, \ldots, N$, we find $MiniBall(B_{v_j})$ as illustrated in Algorithm 1 and get its radius $r_j(X)$, where $V(X) = \bigcup_{i=1,\ldots,m} V_{t_i}$ and $N = \sum_{i=1,\ldots,m} |V_{t_i}|$.

2. denote the cohesive radius of X in the structure S as

$$R''(X) = \frac{\sum_{j=1}^{N} r_j(X)}{N}. \tag{2}$$

There are now N smallest enclosing balls to find in S, which will further limit the approximation error, but the time complexity is now higher than that of *FromOne*.

Figure 4 shows an example of a large approximation error for an itemset of size three, where the smallest possible ball containing a, b and c is depicted using a solid line. However, searching from any given point, we will find a ball with a radius nearly $\sqrt{3}$ times as large as the exact radius of the smallest possible ball. For example, starting off from point a_1, we find the nearest combination a_1, b_1, c_2, resulting in the ball drawn with a dashed line at the top left of Figure 4. Similar results are obtained for the other points.

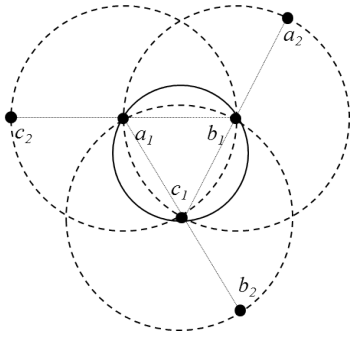

Figure 4. Approximation error made by *FromAll*.

In the rest of the paper, whenever the distinction does not matter, we denote the cohesive radius of X in S as $R(X)$, which is either $R'(X)$ or $R''(X)$, depending on whether we use *FromOne* or *FromAll*.

B. Co-location Pattern

Given a maximum cohesive radius threshold *max_rad*, X is a *co-location pattern* or a *cohesive itemset* if $R(X) \leq max\_rad$. In this case, we say that X is cohesive. Note that the smaller the radius $R(X)$ the higher the cohesion of X. A single item will always be cohesive since the cohesive radius of a singleton is always equal to 0.

The constraint for *FromOne* gives a guarantee that when the first item from a co-location pattern is encountered, the remainder of the set is likely to be found nearby. *FromAll*, meanwhile, gives a guarantee that all the items from a co-location pattern, on average, appear close to each other.

III. Colocation Pattern Mining Algorithm

In this section we present an algorithm for mining co-location patterns in a single structure containing a number of multidimensional points.

A. Search For Itemsets

Figure 5 shows the process of enumerating the frequent itemsets, given that the items $\{a, b, c, d\}$ are sorted, e.g., by ascending or descending frequency. Our method enumerates itemsets in a depth-first manner, i.e., we will process itemsets $\{ab, ac, ad\}$, followed by $\{abc, abd, abcd\}$, and finally acd, before moving on to itemsets whose first item is not a.

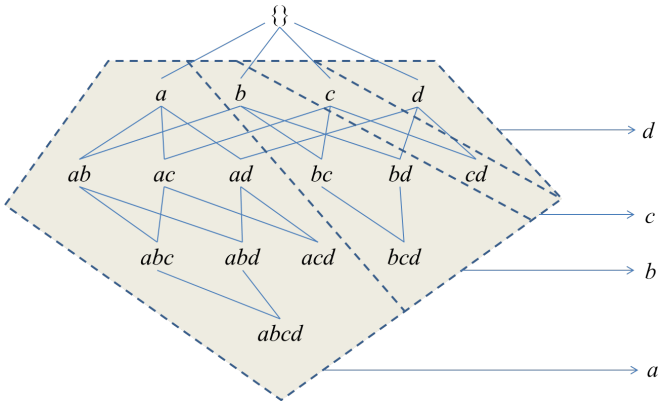

Figure 5. Depth-first search.

B. Pruning

The cohesive radius of an itemset is not a monotonic measure. In other words, it is possible for the cohesive radius of a smaller itemset to be greater than the cohesive radius of one of its supersets. For example, if we are using *FromOne*, $R'(bc)$ may turn out to be larger than $R'(abc)$ as we are computing the smallest balls around different data points. As shown in Figure 6, $R'(bc) > R'(abc)$ since the average of $R'(b_1c_1)$ and $R'(b_2c_1)$is much larger than $R'(a_1b_1c_2)$.

When using *FromAll*, $R''(ab)$ may turn out to be larger than $R''(abc)$ if the smallest balls around the occurrences of c are on average smaller than those around a and b. For example, as illustrated in Figure 7, $R''(ab) > R''(abc)$ since

$$R''(ab) = \frac{2 \times R''(a_1b_1) + R''(a_2b_1)}{3},$$

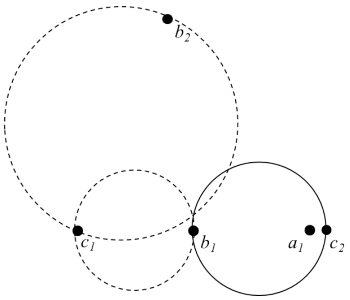

Figure 6. An example resulting in a larger itemset having a smaller cohesive radius using *FromOne*.

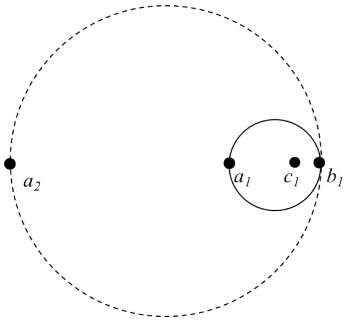

Figure 7. An example resulting in a larger itemset having a smaller cohesive radius using *FromAll*.

and

$$R''(abc) = \frac{3 \times R''(c_1a_1b_1) + R''(a_2b_1c_1)}{4},$$

which is considerably smaller.

As a result, we are unable to use the standard itemset mining pruning techniques that rely on the quality measure (typically frequency) being anti-monotonic. However, we here present alternative pruning techniques that are applicable in our methods, which allow us to develop efficient algorithms for the cohesion based co-location pattern mining task.

1) FromOne: Although the cohesive radius of an itemset is not monotonic, we can still use its properties for pruning certain candidates from the search space. Our pruning method for *FromOne* is based on the following lemma:

Lemma 2. *Assume itemset X is a subset of itemset Y, and all the items of X and Y are sorted by the same order. If they share the same first item, then $R'(X) \leq R'(Y)$.*

Proof: Denote the first item in X and Y with t. Given an occurrence v of t, *FromOne* finds the smallest ball containing v and the nearest occurrences of all other items in X. Clearly, this ball cannot be smaller if we insist that it also contains the nearest occurrences of all items in $Y \setminus X$. *FromOne* finds such smallest balls for each occurrence of t, and then computes the average radius of these balls. Given that X and Y share the first item t, the number of balls will be the same for X and Y, and each ball for Y will be at least as large as the corresponding ball for X. Therefore, $R'(Y)$ (the average radius of such balls for Y) will be at least as large as $R'(X)$. ■

Therefore, we can prune itemsets which are not cohesive when we generate itemsets in the depth-first way as shown in

Figure 5.

2) FromAll: Similarly, we also find a lower bound for the cohesive radius of a superset using the *FromAll* method.

Given a structure S, and itemsets X and Y, such that $X \subset Y$, if both X and Y occur in S, it does not necessarily hold that $R''(X) \leq R''(Y)$, not even if they share the first item, as was the case for *FromOne*. The cohesive radius can now actually get smaller if an item is added to the itemset that allows us to find a smaller smallest ball. Consider the example given in Figure 4. If we add a point d inside the solid circle, $R''(abcd)$ found by *FromAll* will be smaller than $R''(abc)$ (radius of the dashed circles) since the nearest combination from point d gets the radius of the solid circle which is much smaller. However, we have the following lemma:

Lemma 3. *Given itemsets $X = \{t_1, \ldots, t_m\}$ and $Y = \{t_1, \ldots, t_m, t_{m+1}, \ldots, t_n\}$ (i.e., $X \subset Y$), the cohesive radius of Y will always satisfy the inequality*

$$R''(Y) \geq \frac{\sum_{j=1}^{M} r_j(X) + (N-M)\frac{\min_{j=1,\ldots,M} r_j(X)}{2}}{N} = LB(X),$$

where $M = \sum_{i=1}^{m} |V_{t_i}|$ and $N = \sum_{i=1}^{n} |V_{t_i}|$.

Proof: We begin the proof by noting that the cohesive radius of Y is the average radius of N balls, one ball for each occurrence of an item in Y. We denote these radii as $r_j(Y)$, with $j = 1, \ldots, N$. For the first M occurrences (i.e., $j = 1, \ldots, M$), we have already discovered the smallest balls containing all items in X. The radii of those balls are $r_j(X)$, with $j = 1, \ldots, M$. Clearly, by adding the nearest occurrences of items in $Y \setminus X$, these balls can only grow larger. Therefore, we have $r_j(Y) \geq r_j(X)$ for $j = 1, \ldots, M$. We now consider the new balls generated for occurrences of items in $Y \setminus X$. There are $N-M$ such occurrences, for each of which we need to find the smallest ball containing the nearest occurrences of all items in Y. Clearly, the radius of any such ball is larger than the radius of the exact smallest ball containing only items of X in the entire structure (we denote this ball with MB_{XS}). Pick any point in $v \in MB_{XS}$ labelled by an item $t_i \in X$. It follows directly from Lemma 1 that the radius of MB_{XS}, denoted $r(MB_{XS})$, can never be smaller than half the radius of the smallest ball containing v and all items of X that we have discovered using Algorithm 1. Therefore, it follows that $r_k(Y) \geq r(MB_{XS}) \geq \frac{\min_{j=1,\ldots,M} r_j(X)}{2}$ for $k = M+1, \ldots, N$. Since

$$R''(Y) = \frac{\sum_{j=1}^{M} r_j(Y)}{N} = \frac{\sum_{j=1}^{M} r_j(Y) + \sum_{k=M+1}^{N} r_k(Y)}{N},$$

this completes the proof. ∎

From the above proof, it is clear that the cohesive radius of a superset Y can be smaller than that of its subset X only if the new balls added into the average are smaller than the previously discovered balls for the subset. This effect can be maximised if there are as many as possible of such new, smaller, balls. Therefore, to materialise the worst case, in our implementation we take X to be the current candidate, and Y to contain X and all other items yet to be enumerated (i.e., Y is the largest possible itemset that can be generated in the current branch of the depth-first search tree). For example, as can be seen in Figure 5, if $X = \{a\}$, Y will be $\{a, b, c, d\}$, and when $X = \{b, c\}$, Y will be $\{b, c, d\}$. Lemma 3 therefore allows us to define $LB(X)$ as the lower bound of the cohesive radius of any itemset Z, where $X \subseteq Z \subseteq Y$. As can be seen, the pruning bound for *FromAll* is not as strict as that of *FromOne*. Indeed, our experiments show that *FromAll* prunes a lot less, but it does achieve more accurate results.

C. Algorithm

After choosing the way of enumerating the itemsets and the pruning method, we design the algorithm to generate all co-location patterns. Our algorithm generates all co-location patterns in two steps. In the first step, we use the depth-first search method to generate candidate itemsets. In the second step, we determine which of the itemsets are actually spatially cohesive and utilise the observations above to prune the itemsets that cannot be cohesive.

Let *n-itemset* denote an itemset of size n. Let F_n denote the set of *n-itemset*s and T_n be the set of cohesive *n-itemset*s. Algorithms 2 and 3 show the process of generating all co-location patterns. Frequency constraints *min_fre* and *max_fre* can be used to filter out items which are not interesting, due to being either too frequent or not frequent enough. For example, in our *city* dataset (see Section V for details), there are trash cans on every corner, and patterns including trash cans are therefore of little value to the user. Optional parameters, *min_size* and *max_size*, can be used to limit the output only to co-location patterns with a size bigger than or equal to *min_size* and smaller than or equal to *max_size*.

Algorithm 2: GENERATINGCOLOCATIONPATTERNS. An algorithm for generating all co-location patterns in a structure.

Input : structure S, frequency constraints *min_fre*, *max_fre*, maximum cohesive radius threshold *max_rad*, pattern size constraints *min_size*, *max_size*.
Output: all co-location patterns T.

```
1 F_1 = {t | t ∈ I, min_fre ≤ Fre(t) ≤ max_fre};
2 if 1 ≥ min_size then
3     T_1 = F_1;
4 sort(F_1);
5 Depth-First-Search(F_1);
6 T = ⋃ T_i;
7 return T;
```

In Algorithm 2, lines 1-3 count the frequency of all the items to determine the cohesive 1-itemsets. Line 4 sorts the items in F_1 by an order (we later examine which type of ordering produces the best results). Line 5 calls Algorithm 3 to get cohesive n-itemsets (*max_size* $\geq n \geq 2$). Finally, we get the complete set of co-location patterns T (lines 6-7).

In Algorithm 3, given any two n-itemsets α_i and α_j that share the same first $n-1$ items, we generate a candidate itemset X of length $n+1$ by adding the last item in α_j to α_i (line 4). Lines 5-9 show the *FromOne* method for finding cohesive

Algorithm 3: Depth-First-Search(Q)

```
   Input : a set of itemsets Q sharing all but the last item
1  foreach α_i in Q do
2      F_i = ∅;
3      foreach α_j in Q, with j > i do
4          X = α_i + last_item(α_j);
5          if use FromOne then
6              if |X| ≤ max_size and R'(X) ≤ max_rad
               then
7                  F_i = F_i ∪ {X};
8                  if |X| ≥ min_size then
9                      T_|X| = T_|X| ∪ {X};
10         else if use FromAll then
11             if |X| ≤ max_size and LB(X) ≤ max_rad
               then
12                 F_i = F_i ∪ {X};
13                 if |X| ≥ min_size and R''(X) ≤
                   max_rad then
14                     T_|X| = T_|X| ∪ {X};
15     Depth-First-Search(F_i);
```

itemsets while lines 10-14 show the *FromAll* method. In lines 6-7 and 11-12, we prune the candidates that cannot be cohesive by the properties of *FromOne* and *FromAll*, respectively. Then in lines 8-9 and 13-14, we store the cohesive itemsets into T_n.

The two most time consuming steps are the candidate generation and the evaluation of the cohesive radius. For these two steps we use the depth-first search algorithm to generate candidates, and an existing implementation[1] of the algorithm for computing the smallest enclosing ball [8], respectively. The time complexity of these algorithms has been extensively analysed in the papers that originally proposed them. Since the smallest enclosing ball must be computed only for the candidate itemsets that have been generated, the runtime will be proportional to the number of generated candidate itemsets.

IV. Frequent Itemsets Method

In Section III, we presented a method that finds cohesive spatial patterns. If all the items of an itemset regularly occur in a ball with a small enough radius, we consider the itemset cohesive, and report it as a co-location pattern. We now present an alternative method for finding such patterns. Rather than computing the cohesive radius of an itemset, we define a neighbourhood relationship between items and convert the multidimensional structure S into a transaction database $D(S)$.

A. Model

In essence, we are looking for items that often appear near each other in the structure, so we propose to create a transaction database in which each transaction would correspond to the neighbourhood of a single point in the structure. For each point $v \in S$, we draw a ball with v as its centre and radius r. The items covered by the ball make up a transaction corresponding to point v. Therefore, for a point $v \in S$, we define the corresponding transaction as $T(v) = \{item(w) | w \in S, d(v,w) \leq r\}$, where $item(w)$ is the item of point w, $d(v,w)$ is the Euclidean distance from point v to point w and r is the neighbourhood threshold, a user-defined parameter.

We now define the transaction database of S as $D(S) = \{T(v) | v \in S\}$. Once we have converted structure S in this way, we can use a number of existing frequent itemsets mining algorithms, e.g., Apriori [9] or Eclat [10], to find the frequent itemsets in $D(S)$ with a user-defined support threshold *min_sup*. In this paper, we use the Eclat algorithm to mine the frequent itemsets since Eclat is able to handle lower support thresholds in dense datasets [10].

B. Algorithm

The frequent itemsets method generates all interesting itemsets in two steps. In the first step, we use the model introduced above to convert the multidimensional structure S into a transaction database. In the second step, we use the Eclat algorithm to find the frequent itemsets. The algorithm for generating the co-location patterns based on frequent itemset mining is shown in Algorithm 4.

Algorithm 4: *Fre-ball*. A frequent itemset mining based algorithm for discovering co-location patterns in S.

```
   Input : structure S, neighbourhood threshold r,
           support threshold min_sup, frequency
           constraints min_fre, max_fre, pattern size
           constraints min_size, max_size.
   Output: co-location patterns X
1  F_1 = {t | t ∈ I, min_fre ≤ Fre(t) ≤ max_fre};
2  S' = {v | v ∈ S and item(v) ∈ F_1};
3  D(S') = {T(v) | v ∈ S'};
4  X = Eclat(D(S'), min_sup, min_size, max_size);
5  return X;
```

Line 1 counts the the frequency of each item to filter out the items that are either not frequent enough or too frequent. Line 2 creates a new structure S' from S by removing all data points carrying such an item. Line 3 converts the new structure S' into a transaction database. Line 4 uses the Eclat algorithm to find the frequent itemsets in the transaction database $D(S')$ using parameters *min_sup*, *min_size*, and *max_size*. Pattern size constraints *min_size* and *max_size* can be used to limit the output only to frequent itemsets with a size bigger than or equal to *min_size* and smaller than or equal to *max_size*. In line 5, we produce the frequent itemsets discovered by Eclat as output.

The time cost of generating frequent itemsets is equal to that of Eclat, which has been extensively analysed when the method was first proposed [10]. We will now analyse the time needed to convert the structure into a transaction database. To get a transaction $T(v)$, we first need to find the nearest neighbours of point v. In the implementation, we use a k-d tree (short for k-dimensional tree) to store the new structure S'. Theoretically, the time complexity of building the k-d tree is $O(|S'| \times log|S'|)$. In the worst case, each transaction contains all the items in S', so the time complexity of converting the structure into a transaction database is $O(|S'|^2)$. Experiments

[1] http://www.inf.ethz.ch/personal/gaertner/miniball.html

confirm that this method works efficiently in practice since the worst case only occurs when the neighbourhood threshold is large enough to cover the whole structure.

C. Example

Let us now return to our example in Figure 1. Using a neighbourhood threshold of 100 will result in the neighbourhoods shown in Figure 8 (to make the figure clearer, we only draw the neighbourhoods for occurrences of item c). The generated transaction database is shown in Table I. We can see that itemset $abcd$ will be discovered as a frequent itemset if $min\_sup = 0.4$.

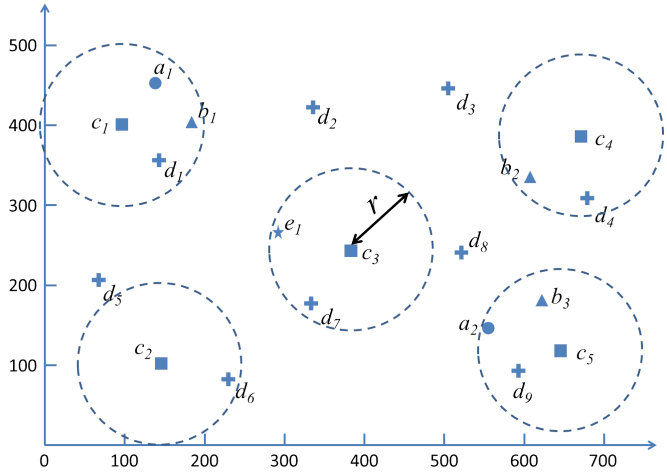

Figure 8. An illustration of the frequent itemsets method.

Table I. A TRANSACTION DATABASE OBTAINED FROM THE STRUCTURE SHOWN IN FIGURE 1.

TID	v	$T(v)$	*TID*	v	$T(v)$
1	a_1	$abcd$	2	a_2	$abcd$
3	b_1	$abcd$	4	b_2	bcd
5	b_3	$abcd$	6	c_1	$abcd$
7	c_2	cd	8	c_3	cde
9	c_4	bcd	10	c_5	$abcd$
11	d_1	$abcd$	12	d_2	d
13	d_3	d	14	d_4	cd
15	d_5	cd	16	d_6	cd
17	d_7	cde	18	d_8	abd
19	d_9	$abcd$	20	e_1	cde

V. EXPERIMENTS

We compared our pattern miners *1-ascend* (use *FromOne* and sort items by ascending frequency), *1-descend* (use *FromOne* and sort items by descending frequency), *FromAll* as described in Section III-C, and *Fre-ball* as described in Section IV, with the method proposed by Huang et al. [2] (which we call *LW-prev* since the method mines prevalent patterns by a "level-wise" approach). Note that the output of *FromAll* will be independent of the order in which the items are sorted. However, we sorted the items by descending frequency in order to reduce the search space and runtime. When using *FromOne*, we sorted the items by both ascending and descending frequency to check the effect of the sorting. We implemented the methods in Java and all experiments were performed on a 2.90GHz Ubuntu machine with 2GB memory.

The *city* dataset we used is one 2-dimensional structure obtained from the open data of the city of Antwerp in Belgium[2]. We first downloaded the datasets containing coordinates of different infrastructure objects with locations, e.g., schools, kindergartens, city offices, playgrounds, cultural institutions, public toilets, recycling centres, trash cans, waste recycling bins, glass recycling bins, hospitals, and so on. We expanded the dataset by adding some data about the city neighbourhoods[3] from 2009, i.e., average age, percentage of immigrant population and average income per person, all of which are numeric attributes. Therefore, we first discretised such numbers into different levels based on the information given on the website and used the coordinate of the centroid of a neighbourhood as its location. Table II shows a few examples of the items generated in this way. Finally, we merged the datasets together, and thus obtained a 2-dimensional structure containing 6424 points carrying 33 different items.

Table II. EXAMPLES OF ITEMS GENERATED FROM THE DEMOGRAPHIC DATA.

Attribute	Item Example	Meaning
average age	age42-45	the average age in the neighbourhood is between 42 and 45
percentage of immigrant population	immigrant33-50	the percentage of immigrant population in the neighbourhood is between 33% and 50%
average income per person	income15k-30k	the average annual income (in euros) per person in the neighbourhood is between 15k and 30k

A. Comparison of the Discovered Patterns

The most cohesive itemsets turned out to be singletons, which was to be expected, since singletons always have a cohesive radius equal to 0. To obtain more meaningful results, we decided to look only for itemsets of size 2 or higher. Therefore, for all methods, *min_size* was set to 2 and *max_size* unlimited. We further found that most of the cohesive itemsets contained trash cans, glass recycling bins or recycling bins, which was not surprising, as there were 3750 trash cans, 551 glass recycling bins and 344 recycling bins in the dataset, making their frequencies a lot larger than that of any other item. As a result, we disregarded these items by setting *max_fre* to 340. We set *min_fre* to 5 to prevent items that hardly ever occur from becoming part of a pattern.

All the presented methods use some sort of a distance threshold, i.e., the maximum cohesive radius threshold *max_rad* for *1-ascend*, *1-descend* and *FromAll*, and the neighbourhood threshold for *Fre-ball* and *LW-prev*. We set the distance threshold to 300 metres for all methods.

We first tried to get the exact results on which we could base our comparisons. However, the exact method failed to produce results even after several days if we set *max_size* to unlimited. We therefore opted to set *max_size* to 5, which gave us sufficient output to be able to compare the remaining methods. We discovered the 27 patterns shown in Table III, where $N = \sum_{i=1,\ldots,m} |V_{t_i}|$. The runtime was 12170.619 seconds. Concrete examples of interesting patterns included

[2]http://opendata.antwerpen.be/
[3]http://www.antwerpen.buurtmonitor.be/

the fact that the higher the percentage of immigrant population, the lower the average income (patterns 6 and 14) as well as the average age (patterns 18 and 25) in a neighbourhood, or that given a school or a playground, there is likely to be a kindergarten nearby (patterns 1 and 21).

Table III. ITEMSETS FOUND BY THE EXACT METHOD.

NO.	Itemset	Cohesive radius	N
1	playground, kindergarten	189.18	390
2	age42-45, income15k-30k	198.49	167
3	dog walking area, playground	199.97	258
4	age<36, income<15k	203.65	168
5	public toilet, playground	217.35	305
6	immigrant33-50, income<15k	224.88	157
7	age39-42, income15k-30k	225.51	175
8	income15k-30k, playground	237.01	294
9	income<15k, kindergarten	242.21	310
10	income<15k, playground	245.53	288
11	renewal area, kindergarten	249.28	397
12	income15k-30k, kindergarten	249.49	316
13	public toilet, kindergarten	249.77	327
14	immigrant10-20, income15k-30k	254.97	170
15	playground, renewal area	256.03	375
16	public toilet, renewal area	256.19	312
17	age36-39, income<15k	260.32	143
18	age42-45, immigrant10-20	260.61	117
19	income<15k, public toilet	265.63	225
20	dog walking area, kindergarten	267.02	280
21	school, kindergarten	273.27	288
22	income15k-30k, public toilet	275.02	231
23	dog walking area, income<15k	278.58	178
24	income<15k, renewal area	278.65	295
25	immigrant33-50, age<36	279.20	117
26	immigrant33-50, school	291.92	135
27	dog walking area, playground, kindergarten	297.58	464

Then, we ran *FromAll*, discovering the 26 patterns shown in Table IV. The runtime was 2.052 seconds. It can be seen that *FromAll* discovered almost exactly the same patterns as the exact method. Pattern {*dog walking area, playground, kindergarten*} is the only one that *FromAll* failed to discover.

Table IV. ITEMSETS FOUND BY *FromAll*.

NO.	Itemset	Cohesive radius	N
1	playground, kindergarten	189.18	390
2	age42-45, income15k-30k	198.49	167
3	dog walking area, playground	199.97	258
4	age<36, income<15k	203.65	168
5	public toilet, playground	217.35	305
6	immigrant33-50, income<15k	224.88	157
7	age39-42, income15k-30k	225.51	175
8	income15k-30k, playground	237.01	294
9	income<15k, kindergarten	242.21	310
10	income<15k, playground	245.53	288
11	renewal area, kindergarten	249.28	397
12	income15k-30k, kindergarten	249.49	316
13	public toilet, kindergarten	249.77	327
14	immigrant10-20, income15k-30k	254.97	170
15	playground, renewal area	256.03	375
16	public toilet, renewal area	256.19	312
17	age36-39, income<15k	260.32	143
18	age42-45, immigrant10-20	260.61	117
19	income<15k, public toilet	265.63	225
20	dog walking area, kindergarten	267.02	280
21	school, kindergarten	273.27	288
22	income15k-30k, public toilet	275.02	231
23	dog walking area, income<15k	278.58	178
24	income<15k, renewal area	278.65	295
25	immigrant33-50, age<36	279.20	117
26	immigrant33-50, school	291.92	135

Thirdly, we ran *1-ascend*, which discovered 314 itemsets in 0.279 seconds, of which the 20 most cohesive ones are shown in Table V. *1-ascend* finds too many patterns since the first item is the least frequent item in the pattern. Pattern 8, for example, tells us that the average distance from a city office to a playground is relatively small, but we know nothing about the average distance from a playground to a city office, which is likely to be much higher, since there are only 7 city offices.

Table V. THE 20 MOST COHESIVE ITEMSETS FOUND BY *1-ascend*.

NO.	Itemset	Cohesive radius	$\lvert V_{t_1} \rvert$
1	immigrant>=50, income<15k	45.86	31
2	immigrant>=50, age<36	51.00	31
3	immigrant>=50, age<36, income<15k	69.84	31
4	immigrant33-50, income<15k	72.62	53
5	city office, renewal area	80.91	7
6	residential area, public toilet	89.71	10
7	age42-45, income15k-30k	102.46	57
8	city office, playground	105.34	7
9	immigrant10-20, income15k-30k	106.84	60
10	dog walking area, playground	110.98	74
11	city office, income<15k	112.59	7
12	immigrant20-33, income15k-30k	117.44	56
13	city office, kindergarten	118.55	7
14	immigrant<10, income15k-30k	118.79	39
15	school, kindergarten	131.00	82
16	immigrant33-50, kindergarten	131.11	53
17	residential area, kindergarten	134.95	10
18	residential area, playground	135.30	10
19	city office, public toilet	136.63	7
20	immigrant33-50, playground	136.93	53

Fourthly, we ran *1-descend*, discovering the 20 patterns shown in Table VI in 0.279 seconds. We note that all these patterns can be found among the 27 resulting patterns discovered by the exact method except patterns 11, 16 and 20. It can been seen that *1-descend* is a better approximation of the desired results than *1-ascend* since *1-descend* computes a more accurate average by sorting the items by frequency in descending order.

Table VI. ITEMSETS FOUND BY *1-descend*.

NO.	Itemset	Cohesive radius	$\lvert V_{t_1} \rvert$
1	kindergarten, playground	167.91	206
2	immigrant10-20, age42-45	209.55	60
3	income<15k, age<36	216.36	104
4	kindergarten, renewal area	226.26	206
5	playground, dog walking area	235.7	184
6	income15k-30k, age42-45	248.25	110
7	income15k-30k, age39-42	252.20	110
8	playground, public toilet	259.18	184
9	income<15k, dog walking area	259.91	104
10	public toilet, income15k-30k	260.82	121
11	immigrant33-50, age36-39	264.60	53
12	kindergarten, public toilet	270.56	206
13	playground, income15k-30k	272.37	184
14	kindergarten, income15k-30k	276.79	206
15	renewal area, playground	281.19	191
16	school, age39-42	286.77	82
17	kindergarten, income<15k	287.30	206
18	income<15k, age36-39	293.17	104
19	kindergarten, dog walking area	293.37	206
20	school, age42-45	298.40	82

We next ran *Fre-ball* with *min_sup* $= 0.1$, discovering 125 patterns in 0.76 seconds. The 20 most frequent itemsets are shown in Table VII. We find that all of these patterns can be found in the 27 resulting patterns discovered by the exact method except patterns 13, 14, 15 and 20. While this method produced satisfactory results, it should be noted that the frequency of a pattern can be boosted by occurrences of other, unrelated, items nearby. For example, pattern {*playground, kindergarten*} has a reported frequency of 637 even though there are just 206 kindergartens and 390 playgrounds in the city.

Finally, we ran *LW-prev* with the prevalence threshold set to 0.1 (as described in Section I), discovering 301 patterns in

Table VII. THE 20 MOST FREQUENT ITEMSETS FOUND BY *Fre-ball*.

NO.	Itemset	Frequency
1	playground, kindergarten	637
2	renewal area, kindergarten	589
3	income<15k, kindergarten	551
4	playground, renewal area	535
5	income<15k, playground	510
6	income<15k, renewal area	469
7	public toilet, playground	460
8	public toilet, kindergarten	457
9	school, kindergarten	445
10	public toilet, renewal area	439
11	income15k-30k, kindergarten	424
12	income15k-30k, playground	408
13	income<15k, playground, kindergarten	388
14	playground, renewal area, kindergarten	387
15	income15k-30k, renewal area	384
16	immigrant33-50, income<15k	369
17	age<36, income<15k	368
18	dog walking area, playground	365
19	income<15k, renewal area, kindergarten	365
20	income<15k, public toilet	360

0.11 seconds. The 20 most prevalent itemsets are shown in Table VIII, where # instances is the number of instances of a pattern. We note that five of these patterns (6, 15, 17, 18, 19) do not appear in the output of the exact algorithm. Most notably, pattern {fire station, city office} has a prevalence value of $3/7$, due to three out of seven fire stations being nearby city offices, but it has a low cohesion value because the remaining 4 fire stations are very far from any city office. We conclude that *FromAll* produces output closest to the exact algorithm, followed by *1-descend*. We next provide an analysis of the runtimes of all the presented methods.

Table VIII. THE 20 MOST PREVALENT ITEMSETS FOUND BY *LW-prev*.

NO.	Itemset	Prevalence	# instances
1	income<15k, kindergarten	0.5194	145
2	age<36, income<15k	0.4807	64
3	renewal area, kindergarten	0.4515	212
4	playground, kindergarten	0.4511	178
5	immigrant33-50, income<15k	0.4423	60
6	fire station, city office	0.4286	3
7	age42-45, immigrant10-20	0.4210	29
8	income<15k, renewal area	0.4188	109
9	school, kindergarten	0.4175	123
10	immigrant33-50, school	0.4146	46
11	age42-45, income15k-30k	0.4091	49
12	playground, renewal area	0.4076	149
13	income<15k, public toilet	0.3942	71
14	public toilet, renewal area	0.3927	149
15	age<36, immigrant>=50	0.3906	33
16	income<15k, playground	0.3859	93
17	age39-42, immigrant20-33	0.3846	31
18	age>=45, immigrant10-20	0.3833	25
19	income15k-30k, renewal area	0.3818	87
20	immigrant10-20, income15k-30k	0.3818	48

B. Impact of Distance Threshold on Runtime

Figure 9 shows the runtimes of the methods with various distance thresholds where other parameters are kept the same as before. The results show that, as the distance threshold increases, the runtimes of all methods increase. This is because a larger distance threshold will increase the number of candidate patterns that need to be processed. We find that the runtime of the exact method, even with *max_size* $= 5$, is prohibitive since it takes more than 10^5 seconds when the distance threshold is more than 350 metres. The runtimes of the exact method, *FromAll* and *LW-prev* increase too fast while the runtimes of other methods are acceptable. It can

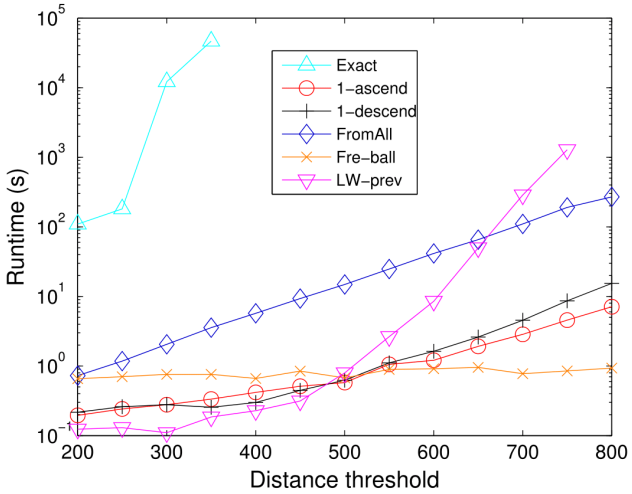

Figure 9. Impact of the distance threshold on the runtime.

be noted that the exact method is the most time consuming, while *LW-prev* runs out of memory when the distance threshold is 800 since the process of identifying prevalent co-location patterns needs more memory than the computations required by our algorithms. *Fre-ball* appears to be the most scalable method, but taking into account both efficiency and accuracy, we conclude that *1-descend* seems to be the best choice.

C. Impact of Structure Size on Runtime

Figure 10 shows the runtimes of the methods with respect to different number of points (varying from 10% to 100% of the whole structure). In this experiment we use the whole structure without setting *min_fre* and *max_fre* for the items. We set *min_size* to 2, *max_size* unlimited, and the distance threshold to 300 metres for all methods. We ran Fre-ball with *min_sup* $= 0.1$ and LW-prev with the prevalence threshold set to 0.1. We repeated the experiment ten times, using ten different random permutations of the data points. The reported runtimes are therefore averages of the ten different runs. For all methods, the runtime grows with increasing number of points. The exact method seems to be prohibitive, while the two *FromOne* variants (*1-ascend* and *1-descend*) are the fastest. Comparing the resulting patterns, we find that *FromAll* gets the best approximation, but *1-descend* performs comparably in terms of accuracy while achieving much quicker runtimes.

VI. RELATED WORK

Co-location pattern mining approaches are mainly based on spatial relationship such as “close to” proposed by Koperski and Han [11]. Morimoto proposed a method to find groups of various service types originating from nearby locations and reported a group if its frequency of occurrences is above a given threshold [12]. The basic definitions of co-location pattern mining and the general framework was first proposed by Shekhar and Huang [1]. The paper proposes a prevalence measure with an anti-monotonic property, which helps to build an Apriori [9] based approach to mine prevalent co-location patterns in a reduced search space. This was later extended through the usage of a multi-resolution pruning technique to

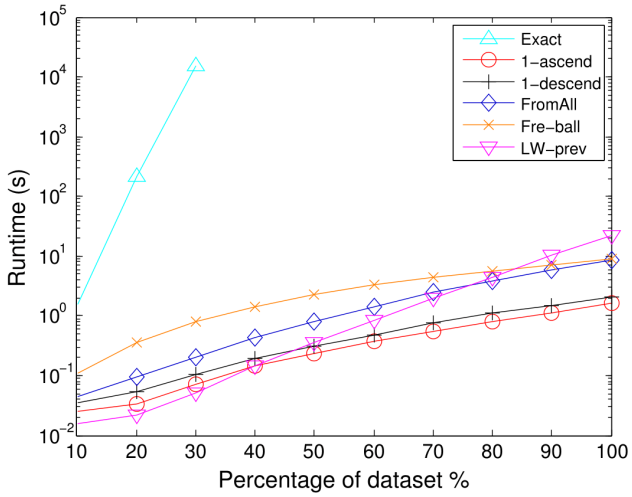

Figure 10. Impact of the structure size on the runtime.

prune non-prevalent co-locations at a reduced computational cost [2]. A faster method for the same problem setting was proposed by Zhang et al. [3], while the problem of mining co-location patterns with rare spatial items has been studied by Huang et al. [13].

The above methods [1], [2], [3], [13] have used spatial join approaches to identify patterns whose items are close to each other. A join operation, however, is expensive to compute when searching for patterns. The space required to store all pattern instances is large, and the instance search cost before joining will also increase significantly. To further improve the runtime, Yoo et al. proposed a new join-less approach where a neighbourhood relationship is materialized from a clique type neighbourhood and a star type neighbourhood, respectively [4]. Xiao et al. proposed a density based approach to improve the runtime [5]. From a dense region, the method counts the number of instances of a candidate co-location. Assuming all the remaining instances are in co-locations, the method then estimates an upper bound of the prevalence value and if it is below the threshold the candidate co-location is pruned.

However, all the above mentioned methods use a prevalence threshold and look for prevalent patterns based on a user-defined neighbourhood threshold. Therefore, if thresholds are not selected properly, meaningless co-location patterns could be reported, or meaningful co-location patterns could be missed when the prevalence threshold is too high. Barua and Sanders proposed a statistical approach to mine true patterns without using a prevalence measure threshold but this method, too, uses a given neighbourhood threshold [6].

VII. Conclusion

In this paper, we have presented two methods (*FromOne* and *FromAll*) to mine co-location patterns in a multidimensional structure. Based on the classical frequent itemset mining approach, we develop another complete algorithm *Fre-ball* to mine cohesive itemsets in the structure, by converting the spatial data into a transaction database. We applied the methods to find spatially cohesive patterns from the open data of a city and the resulting patterns demonstrated the efficiency and intuitiveness of the proposed methods. Through experimental evaluation, we confirmed that *FromOne* finds patterns much faster than *FromAll* while *FromAll* approximates better. *Fre-ball* appears to be the most scalable method, but the frequency of a pattern generated by this method may be boosted by occurrences of other items nearby. Meanwhile, *FromOne* only gives a guarantee that when the first item from a co-location pattern is encountered, the remainder of the set will be found nearby, while *FromAll* gives a guarantee that all the items from a co-location pattern, on average, appear close to each other. We further examined the order in which the items are enumerated when generating and evaluating candidate itemsets, and discovered that sorting them in the order of descending frequency allowed *FromOne* to produce satisfactory results requiring very little running time. Taking into account both efficiency and accuracy, we conclude that this variant of *FromOne* (which we call *1-descend*) does well on both counts, while all other methods (either proposed here or already existing in literature) do worse on at least one of the two criteria.

References

[1] S. Shekhar and Y. Huang, "Discovering spatial co-location patterns: A summary of results," in *Advances in Spatial and Temporal Databases*. Springer, 2001, pp. 236–256.

[2] Y. Huang, S. Shekhar, and H. Xiong, "Discovering colocation patterns from spatial data sets: a general approach," *IEEE Transactions on Knowledge and Data Engineering*, vol. 16, no. 12, pp. 1472–1485, 2004.

[3] X. Zhang, N. Mamoulis, D. W. Cheung, and Y. Shou, "Fast mining of spatial collocations," in *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2004, pp. 384–393.

[4] J. S. Yoo and S. Shekhar, "A joinless approach for mining spatial colocation patterns," *IEEE Transactions on Knowledge and Data Engineering*, vol. 18, no. 10, pp. 1323–1337, 2006.

[5] X. Xiao, X. Xie, Q. Luo, and W.-Y. Ma, "Density based co-location pattern discovery," in *Proceedings of the 16th ACM SIGSPATIAL international conference on Advances in geographic information systems*. ACM, 2008, p. 29.

[6] S. Barua and J. Sander, "Mining statistically significant co-location and segregation patterns," *IEEE Transactions on Knowledge and Data Engineering*, vol. 26, no. 5, pp. 1185–1199, 2014.

[7] C. Zhou, P. Meysman, B. Cule, K. Laukens, and B. Goethals, "Discovery of spatially cohesive itemsets in three-dimensional protein structures," *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, vol. 11, no. 5, pp. 814–825, 2014.

[8] B. Gärtner, "Fast and robust smallest enclosing balls," in *Algorithms-ESA'99*. Springer, 1999, pp. 325–338.

[9] R. Agrawal and R. Srikant, "Fast algorithms for mining association rules," in *VLDB'94*. Morgan Kaufmann Publishers, 1994, pp. 487–499.

[10] M. J. Zaki, "Scalable algorithms for association mining," *IEEE Transactions on Knowledge and Data Engineering*, vol. 12, no. 3, pp. 372–390, 2000.

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

[12] Y. Morimoto, "Mining frequent neighboring class sets in spatial databases," in *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2001, pp. 353–358.

[13] Y. Huang, J. Pei, and H. Xiong, "Mining co-location patterns with rare events from spatial data sets," *Geoinformatica*, vol. 10, no. 3, pp. 239–260, 2006.