

SpaRClus: Spatial Relationship Pattern-Based Hierarchical Clustering*

Sangkyum Kim Xin Jin Jiawei Han
University of Illinois at Urbana-Champaign
kim71@uiuc.edu, xinjin3@uiuc.edu, hanj@cs.uiuc.edu

January 25, 2008

Abstract

For the past decade, the need of multimedia mining has increased tremendously, especially in image data due to inexpensive digital technologies and fast mounting of image data. In this paper, we, first, show an algorithm, *SpIBag* (**S**patial **I**tem **B**ag Mining), which discovers frequent spatial patterns in images. Due to the properties of image data, *SpIBag* considers a bag of items together with a spatial information as a pattern which persists over geometrical transformations, such as scaling, translation, and rotation. Then, based on *SpIBag*, we propose *SpaRClus* (**S**patial **R**elationship Pattern-Based Hierarchical **C**lustering) to cluster image data. Our performance study shows that the method is effective and efficient.

1 Introduction

For the past decade, there has been a big boom in using multimedia data due to cheap prices of digital equipments and storages, fast network environments, and exciting web applications like *Facebook* and *YouTube*. Moreover, the scientific usage of multimedia data has been demanding various powerful data mining tools to analyze and retrieve useful information from a tremendous amount of data which is itself in a big size. For example, pictures taken from thousands of satellites and abruptly increasing number of bio image data are now far above the capacity of labeling or creating meta-data manually by human being itself.

In this paper, we focus on clustering image data which is one of the fundamental step of image data mining. Traditionally, most clustering approaches adopted a framework on a pairwise similarity measure between two images. For this kind of frameworks, meaningful feature extraction was the most important part of the research. But most images contain small amount of sev-

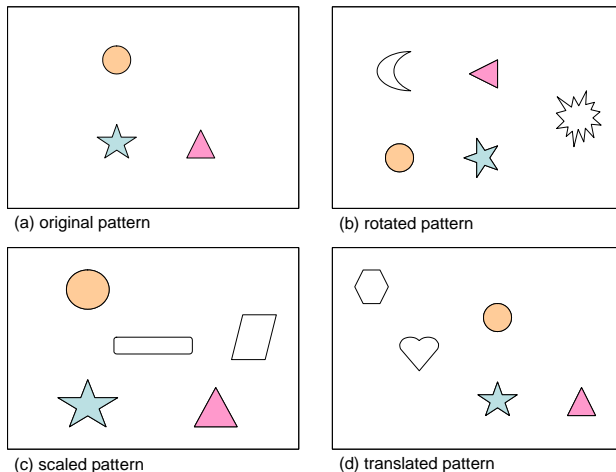


Figure 1: Examples of a pattern which is invariant to semi-affine transformations

eral key parts and a big portion of background. We want to group images which share lots of important common key parts as a cluster, without assuming those features being near each other.

For that reason, we developed an algorithm, *SpIBag* (**S**patial **I**tem **B**ag Mining), which discovers frequent spatial item bags in images. *SpIBag* is able to mine scale-, translation-, and rotation-invariant frequent substructures. We say that *SpIBag* is invariant on *semi-affine transformations*. For example, suppose a photographer takes several pictures of a room in a row. They need not be identical, but at least they preserve similar shape among items. Semi-affine transformation is a way to express or detect shape preserving images. By use of *SpIBag*, we propose a meaningful clustering algorithm which considers semi-affine transformations.

DEFINITION 1.1. *We define an affine transformation to be a composition of one or several linear transformations (rotation, scaling or shear) and translation (shift). We define semi-affine transformation to be a composition of one or several scaling, translation, or rotation*

*Supported in part by U.S. Office of Naval Research, U.S. National Science Foundation NSF N00014-06-1-1108 and NSF IIS-0513678. Any opinions, findings, and conclusions or recommendations expressed here are those of the authors and do not necessarily reflect the views of the funding agencies.

transformations. We say two patterns are similar if a pattern is semi-affine transformable from the other pattern.

EXAMPLE 1. Look at Fig.1. (a) is the original pattern with a star, a triangle, and a circle. (b) is an image containing a pattern in (a) which is rotated, (c) is an image containing a pattern in (a) which is scaled, and (d) is an image containing a pattern in (a) which is translated. We consider patterns of a star, a triangle, and a circle in images (a) \sim (d) to be all similar.

There have been quite a few document clustering algorithms based on frequent item sets [6], or closed frequent item sets [19], but there are two big differences between document clustering and image clustering. First, documents only consider item sets while images consider item bags. For example, using a term *wheel* one time or two times in a document doesn't make a big difference. It merely affects a scoring function which uses *TF-IDF* formula. In image domain, it is different. There is a big difference between a bike using one wheel and a bike using two wheels. Second, documents do not consider locations of terms which are very important in image data. In a document, there is no difference in the order of using two terms *mouth* and *nose*, but, in an image, relative/absolute locations of a mouth and a nose are significantly important. Therefore, we developed an image clustering algorithm based on the frequent substructures discovered by *SpIBag*. We use location information to detect similar bags of items.

DEFINITION 1.2. We define a bag to be a collection of items which allows replicates. A set is a collection of items which are all unique. For example, $I = \{a, a, a, b, c\}$ is a bag not a set since there are three *a*'s.

Based on frequent spatial patterns found from *SpIBag*, we are able to group images into several clusters. As in document clustering, each frequent spatial pattern becomes a representative of a cluster. Since frequent spatial patterns form a hierarchical graph, we can construct a similar graph of clusters based on the support image sets of frequent spatial patterns. Using *Apriori* property, we construct it from top to bottom manner, which becomes a divisive hierarchical clustering algorithm. To decide whether to divide current cluster, we use an entropy based score function to measure its cohesiveness. Once the graph is completed, we find similar clusters among its leaves and merge them. If a user wants clusters to be disjoint, we will use another score function for each common image to decide where it belongs to. Score functions defined in this paper are all based on spatial information, which are also different from traditional document clustering algorithms.

In this paper, we assume all objects are already identified. Our property that *SpIBag* is invariant on scaling, translation, and rotation assumes that the inputs are already identified on scaling, translation and rotation configurations. Image object identification is a branch of image processing field which have been producing lots of works [3, 27, 30]. But, in fact, our framework allows not only identified objects but also features like *GIST* [22] or *SIFT* [17] as inputs. We can quantize feature vectors and use them as our input data. From now on, we will use the term *item* that can be interpreted either as a real identified image object or a quantized feature vector. Finding a better algorithm of object identification or feature selection is beyond the scope of this paper.

In summary, the contributions of this paper are as follows:

- We propose a framework of image clustering that utilizes bags of features paradigm together with the spatial information.
- We define a spatial pattern in image data.
- We present *SpIBag*, a frequent spatial item bag mining algorithm, which persists over scaling, translation, and rotation.
- We present *SpaRClus* which is a divisive hierarchical clustering algorithm for image data. Using an entropy concept on a well-defined scoring function, we can decide whether to divide current cluster further, which enables *SpaRClus* to be efficient.
- By use of various data configuration, we demonstrate that *SpaRClus* is effective and efficient.

The rest of the paper is organized as follows: Section 2 presents a brief overview of related works. In Section 3, various notations and definitions together with a basic approach on this problem are introduced. Section 4.1 proposes *SpIBag*, a frequent spatial pattern mining algorithm that is invariant on scaling, translation, and rotation. Section 5.1 proposes *SpaRClus*, an image clustering algorithm, which is divisive hierarchical considering spatial information. In Section 6, we report our experimental results which show the effectiveness and efficiency of our framework. We discuss several issues of *SpaRClus* in Section 7. Finally, we conclude the paper in Section 8.

2 Related Works

In this section, we briefly introduce previous works in computer vision and data mining fields.

2.1 Image Retrieval and Computer Vision Image retrieval has been received a lot of attention from research community for the past decades. Efficient browsing, searching, and retrieval were main research issues to store and access the image data effectively. Starting from late 1970s, text-based image retrieval [4, 18, 29] after annotating every image with text was used. However, with huge increase of the number of images, annotating all image data has become very difficult. Furthermore, the quality of image retrieval depended heavily on the annotated text. To overcome this deficiency, quite a few content-based image retrieval systems such as QBIC [18], Virage [11] and RetrievalWare [5] were proposed. These systems first extract features such as color, shape and texture histograms from images and use them to retrieve similar images. Separate distance functions of color, shape, and texture are defined and subsequently combined to derive the overall result. The quality of image retrieval depends on the characteristics of features [2] and thus researchers have investigated to develop effective features. Color histograms are one of the most popular features that has been used to measure the similarity between images [25]. Color histograms might be effective for image retrieval when the uniqueness in the color pattern held against the pattern in the rest of the entire data set is assured. However, when large sized images are at stake, the performance suffers because color histogram comparison saturates the discrimination [24]. Wavelet has been studied as an alternate feature. In [12, 28], the dominant *wavelet* coefficients of an image are used as its signature – since wavelets capture shape, texture and location information in a single unified framework, their use ameliorates some of the problems with earlier algorithms.

Recently, a new paradigm appeared in computer vision field, to use dictionary of visual words with the *bag-of-words* methodology, which was adapted by computer vision applications [14, 16]. Basically, they all share the following steps: First, detect regions or points of interests, and then compute quantized feature vectors over them to form a visual vocabulary. After that, they find the occurrences in the image of each word of the vocabulary to build the bag of words. Similar to document clustering frameworks, they do not consider spatial information which might lead to a large number of false positives. For example, they consider an image with a mouth beside a nose and another image with a mouth below a nose to be the same. Only in [14], they apply a well-known pyramid matching scheme [7] to use a spatial information indirectly. They repeatedly subdivide the image and compute histograms of local features at increasingly fine resolutions. This scheme is not durable on scaling, translation, and rotation.

Our framework can use the same input as *bag-of-words* methodology, and it persists over semi-affine transformations.

2.2 Data Mining Traditionally, spatial data mining played an important role in GIS field. They mainly focused on finding relationships between objects in spatial database by use of spatial association rule mining [13, 31]. These works need predefined predicates of relationships between objects such as *overlap* and *next-to* which are in elementary topological forms that define relationships between two objects. There have been other approaches in spatial data mining area like mining spatial collocations [10, 32]. They tried to find co-existing patterns of non-spatial features in a spatial neighborhood, but they did not consider spatial patterns from separate neighborhoods.

On the other part of data mining field, image data mining appeared due to the abrupt increase of image data. Works in this field usually assumed that image data can be preprocessed using the existing image processing techniques. [20] proposed association rule mining algorithms which revealed patterns of objects, but they did not consider spatial information. In a similar direction, a viewpoint pattern mining in image database was proposed in [8] which used spatial information using a relative distance idea which also found patterns invariant in translation and partially in rotation with predefined fixed angles. Afterwards, there have been image data mining works that use a grid based approach to explicitly code the location of objects in each image [9, 15]. They could generate spatial association rules based on the grid information, but they were not able to consider shape preserving properties.

3 Preliminary

In this section, we define several terminologies that would be used for the rest of the paper and define our problem setting by use of those terminologies. Also, we briefly introduce traditional method that has been used to mine frequent item set, that will give a background knowledge for *SpIBag* algorithm.

3.1 Problem Statement In [31], an image I was defined as a bag of items:

$$I = \{a_1, a_2, \dots, a_n\}$$

where replicated items are allowed in I as mentioned in Def 1.2. But in the real life, we also have to consider spatial information of items in image data, since shapes which several items in an image form together are also important to define an image. Even the items are the same, they might form different shapes which leads to

a totally different image.

DEFINITION 3.1. We say an item a to be an identified object or a quantized feature vector of the image data. We define an image I to be a bag of items together with spatial information.

To get the exact image, we need location information like 2-dimensional coordinates which would be useful for pairwise matching between images. In our case, since we are interested in more realistic situations, we want to consider semi-affine transformation invariant spatial information which preserves shapes among items. For this reason, we consider relative location between items instead of absolute location of each item. We define a spatial pattern in an image which contains the required spatial information, and then consider an image as a bag of patterns.

We say p is a spatial pattern, or in short a pattern, of an image I when p contains a bag of items, $B \in I$, with spatial information. To define spatial information, it first occurs in mind to use distances of all pairs of items in B . In that case, we need $O(n^2)$ distances to express spatial information of a pattern p .

DEFINITION 3.2. We define p to be a pattern in an image I , as

$$p = (B, S)$$

where p contains a bag of items $B = \langle a_1, a_2, \dots, a_n \rangle$ with spatial information $S = \{d(a_i, a_j) | i, j = 1, \dots, n\}$ where $d(a_i, a_j)$ is the Euclidean distance between two items a_i and a_j . Here, the spatial information is the distances between all pairs of items in B . We denote B_p to be the bag of items in p and S_p to be the spatial information of p . We say n is the size of p and denote as $|p| = n$. We say p is an n -pattern.

In principle, to check whether two n -patterns p_1 and p_2 are similar, we have to check if their corresponding distances are in the same rate, since we have to consider the scaled pattern also. Sometimes we need to check all possibilities of correspondences between same items, since we allow duplicate items in B_p . So, following the definition directly, we need $O(n^2)$ computations to check if two n -patterns p_1 and p_2 are similar.

We get an idea for an efficient computation from the following propositions.

DEFINITION 3.3. We define q to be a subpattern of p if q is a pattern and satisfies both $B_q \subset B_p$ and $S_q \subset S_p$ conditions. We say q is an m -subpattern of p if q is a subpattern of p and $|q| = m$.

PROPOSITION 3.1. Two n -patterns p_1 and p_2 are similar if and only if all corresponding 3-subpatterns of p_1 and p_2 are similar.

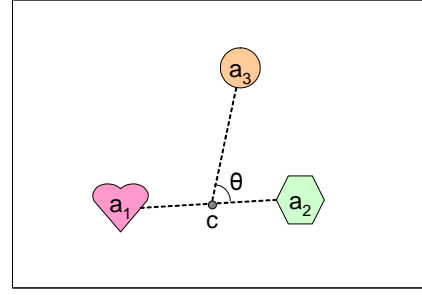


Figure 2: Example of a 3-pattern.

Based on Prop 3.1, we can express a pattern with a group of 3-subpatterns. A 3-pattern is the basic unit of an image pattern that preserves shape information. For later use, based on properties of triangle similarity, we first redefine these basic patterns, and express a general n -pattern as a union of basic patterns.

DEFINITION 3.4. Let p be a 3-pattern, i.e. $p = (\langle a_1, a_2, a_3 \rangle, S)$, with a predefined order among items in B_p . Let c be the center of two items a_1 and a_2 . We denote p as $p = (\langle a_1, a_2, a_3 \rangle, \theta, r)$ where $\theta = \angle a_3ca_2$ and $r = d(c, a_3)/d(a_1, a_2)$.

PROPOSITION 3.2. Let p be an n -pattern, for $n \geq 4$. Then, we can express p as a bag of 3-patterns using the expression in Def 3.4.

PROPOSITION 3.3. Let p be an n -pattern, i.e. $p = (\langle a_1, a_2, \dots, a_n \rangle, S)$, with a predefined order among items, for $n \geq 4$. Then, we can express p as a set of $(n - 2)$ 3-patterns using the expression in Prop 3.4 with the first two elements being a_1 and a_2 :

$$\{(\langle a_1, a_2, a_3 \rangle, \theta_1, r_1), \dots, (\langle a_1, a_2, a_n \rangle, \theta_{n-2}, r_{n-2})\}$$

Based on Prop 3.3, since a line $\overline{a_1a_2}$ is fixed for 3-patterns, we need $O(n)$ 3-patterns to express an n -pattern p . This means that we only need $O(n)$ computation to decide whether two n -patterns are similar or not. Compared to $O(n^2)$ computation which is required when we follow the formula in Def 3.2 directly, this new expression allows a faster computation, and thus, will be used for the rest of the paper.

So far, we defined a *pattern* in an image. Now, we redefine an *image* using this new definition of patterns.

DEFINITION 3.5. We define an image I as a union of all patterns in I .

DEFINITION 3.6. Let \tilde{I} be a set of all images, i.e. $\tilde{I} = \{I_1, I_2, \dots, I_l\}$. We define the support of a pattern p ,

support(p), by the size of the set $\tilde{I}_p = \{I | I \in \tilde{I}, p \in I\}$ of images which contain p ,

$$\text{support}(p) = |\tilde{I}_p|$$

We say a pattern p is frequent if $\text{support}(p) \geq \delta_f$ for a given threshold δ_f .

We define the support of a pattern p using the number of images that contain p . Now let's define our problem in this paper.

Problem Definition Given a set of preprocessed images $\tilde{I} = \{I_1, I_2, \dots, I_l\}$ and a minimum support threshold δ_f ,

1. how to mine frequent patterns of \tilde{I} .
2. how to use mined patterns to find clusters of images.

3.2 Frequent Item Set Mining In this subsection, we briefly introduce a data mining algorithm that finds frequent item sets, called *Apriori* algorithm [1]. The main idea is based on the following property:

Apriori property: All nonempty subsets of a frequent itemset must also be frequent.

In other words, if an itemset is not frequent, then its superset can not be frequent. This property is useful when it is used as a strategy for pruning. *Apriori* method increases the size of candidate frequent itemsets by one for each iteration, and if a candidate itemset turns out not to be frequent then its supersets need not be generated since they can not be frequent because of the *Apriori* property. The basic procedure of *Apriori* algorithm is based on the following two steps, *joining* and *pruning*:

1. **Join step:** To find a candidate frequent $(n + 1)$ -itemset, we join two frequent n -itemsets. We can join two n -frequent itemsets if their first $n - 1$ items are the same and the last items are different. Note that, in *Apriori*, an order among items is assumed. For example, suppose $f_1 = \{a_1, a_2, \dots, a_n\}$ and $f_2 = \{a_1, a_2, \dots, a_{n-1}, a_{n+1}\}$ are frequent n -itemsets. Since the first $n - 1$ items, a_1, a_2, \dots, a_{n-1} , are common and the last items, a_n of f_1 and a_{n+1} of f_2 , are different, we can join itemsets f_1 and f_2 and get a candidate $(n + 1)$ -itemset $\{a_1, a_2, \dots, a_n, a_{n+1}\}$.
2. **Prune step:** Candidate frequent itemsets found in join step might not be frequent. We need to examine their support and prune the ones which has

support lower than the given minimum threshold. For this reason, we have to scan the whole database to count the support of candidates for each size. Before the scanning, we can reduce the number of candidates by checking if their subsets are all frequent. For example, if we got $(n + 1)$ -itemset $\{a_1, a_2, \dots, a_n, a_{n+1}\}$ as a candidate, then we check whether its $(n+1)$ n -subitemsets $\{a_1, a_2, \dots, a_n\}, \dots, \{a_2, \dots, a_{n+1}\}$ are all frequent. This checking can be done efficiently by use of a hash tree of all frequent itemsets.

There are lots of variations of *Apriori* algorithm to improve its computation efficiency, such as using hash-based technique [21], transaction reduction [1, 21], partitioning [23], sampling [26], and so on, but we do not discuss them in detail since they are not our focus in this paper.

4 Spatial Relationship Pattern Mining

In this section, we explain *SpIBag* (**S**patial **I**tem **B**ag Mining) in detail. Here, we use a pattern formula based on Prop 3.3, not from the Def 3.2 directly. There are two important features in *SpIBag*.

- *SpIBag* adapts *Apriori* property together with a pruning strategy based on the spatial information.
- Considering the real situations, we define approximately similar patterns, and allow those patterns to be in the same pattern group.

In Sec 4.1, we explain of main procedures in *SpIBag*. In Sec 4.2, we show in detail how to do grouping among frequent 3-patterns.

ALGORITHM 4.1. The *SpIBag* Algorithm

Procedure *SpIBag* Algorithm

```

begin
1.  $F_3 =$  find frequent 3-pattern groups of  $\tilde{I}$ 
2. for( $k = 4; F_{k-1} \neq \emptyset; k++$ ) {
3.    $C_k =$  candidate_generator( $F_{k-1}$ )
4.   for( $I \in \tilde{I}$ ) {
5.     for( $p \in C_k$ ) {
6.       if  $p \in I$  then
7.         p.count++
8.       }
9.     }
10.     $F_k = \{p \in C_k, p.count \geq min\_sup\}$ 
11.  }
12. return  $F = \bigcup_k F_k$ 

```

end

4.1 The *SpIBag* Algorithm The whole procedure of *SpIBag* seems quite similar to the *Apriori* algorithm explained in Sec 3.2. But, looking into details, we see several big differences.

At line 1 in Algorithm 4.1, we start from finding frequent 3-patterns which are the smallest meaningful units for semi-affine transformations. As for 1-patterns and 2-patterns, we simply say two patterns are similar if they contain the same items. In those cases, not considering spatial information means we consider semi-affine transformations since all images with the same items are all similar under semi-affine transformation. For these reasons, we only consider the case where each image has at least three frequent patterns for the rest of the paper.

Again at line 1 in Algorithm 4.1, we use a term *group*. Since we consider approximately similar patterns, we do the grouping among frequent 3-patterns to consider patterns in each group to be all similar. Based on those similar groups, we proceed *SpIBag* algorithm. We explain how to perform the grouping on 3-patterns in Sec 4.2.

Using frequent 3-patterns F_3 found at line 1, we start joining and pruning process which is quite similar with what is explained in Sec 3.2, except that *SpIBag* considers spatial information which enables a further pruning at joining step.

At line 3 in Algorithm 4.1, to find a candidate frequent $(n + 1)$ -pattern, we join two frequent n -patterns. We can join two n -frequent patterns if

- in case $n = 3$, their first two items are the same and the last items are different.
- in case $n \geq 4$, their first $(n - 3)$ 3-subpatterns are the same and the last subpatterns are different.

Note that, similar to *Apriori* algorithm, an order among items is assumed in *SpIBag*.

EXAMPLE 2. Suppose there are two frequent n -patterns:

$$\begin{aligned} p_1 &= \{(\langle a_1, a_2, a_3 \rangle, \theta_1, r_1), \dots, \\ &\quad (\langle a_1, a_2, a_n \rangle, \theta_{n-2}, r_{n-2})\} \\ p_2 &= \{(\langle a_1, a_2, a_3 \rangle, \theta_1, r_1), \dots, \\ &\quad (\langle a_1, a_2, a_{n-1} \rangle, \theta_{n-3}, r_{n-3}), \\ &\quad (\langle a_1, a_2, a_{n+1} \rangle, \theta_{n-1}, r_{n-1})\}. \end{aligned}$$

Since the first $(n - 3)$ 3-subpatterns, $(\langle a_1, a_2, a_3 \rangle, \theta_1, r_1), \dots, (\langle a_1, a_2, a_{n-1} \rangle, \theta_{n-3}, r_{n-3})$, are common and the last items, $(\langle a_1, a_2, a_n \rangle, \theta_{n-2}, r_{n-2})$ of p_1 and $(\langle a_1, a_2, a_{n+1} \rangle, \theta_{n-1}, r_{n-1})$ of p_2 , are different, we can join p_1 and p_2 and get a candidate $(n + 1)$ -pattern $\{(\langle a_1, a_2, a_3 \rangle, \theta_1, r_1), \dots, (\langle a_1, a_2, a_{n+1} \rangle, \theta_{n-1}, r_{n-1})\}$.

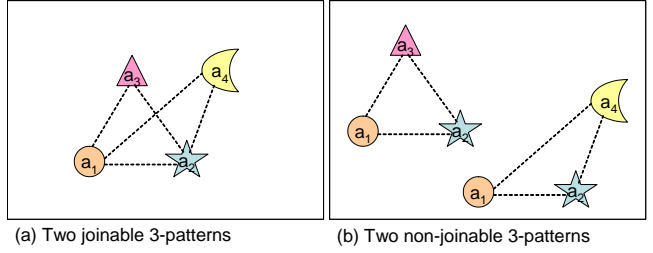


Figure 3: Joinable and non-joinable 3-patterns.

But, there is one more thing to consider, the location of patterns. There might be several duplicated patterns in the same image since image data allows replicated items. To join two frequent n -patterns p_1 and p_2 correctly and produce a candidate $(n + 1)$ -pattern, we have to make it sure that the first $(n - 3)$ 3-subpatterns of p_1 and p_2 are the same even in their locations. We can assure it when we check the identity or location of the first two items of p_1 and p_2 . For this reason, we allocate each item of every image an identification number (ID) and use that attribute to check whether we can join the patterns or not in the given image. If the first two items of p_1 and p_2 have the same IDs correspondingly and satisfies the join condition mentioned above, then we are able to join p_1 and p_2 . In Fig.3, we show two patterns that are joinable and non-joinable. In Fig.3(b), even if two 3-patterns have items a_1 and a_2 in common, they cannot be joined since the a_1 and a_2 of the left pattern have different IDs from the right ones.

PROPOSITION 4.1. Let p_1 and p_2 be two n -patterns defined in Example 2 which are in the same image I . Then, patterns p_1 and p_2 are the same if and only if (ID of p_1 's $a_1 = \text{ID of } p_2$'s a_1) and (ID of p_1 's $a_2 = \text{ID of } p_2$'s a_2) and (all 3-subpatterns of $p_1 = \text{all 3-subpatterns of } p_2$).

At lines 4 ~ 10 in Algorithm 4.1, we count the support of each candidate pattern and prune the ones whose support are below minimum threshold δ_f . This pruning part is similar with the one described in Sec 3.2, so we skip the details in this section.

4.2 Grouping frequent 3-patterns For further discussions, we formally define approximately similar patterns as a neighborhood of a pattern. As discussed in Sec 4.1, we use pattern expressions based on Prop 3.3.

DEFINITION 4.1. We say two 3-patterns $p_1 = \{\langle a_1, a_2, a_3 \rangle, \theta_1, r_1\}$ and $p_2 = \{\langle a_1, a_2, a_3 \rangle, \theta_2, r_2\}$ are approximately similar if $|\theta_1 - \theta_2| < \delta_\theta$ and $|r_1 - r_2| < \delta_r$ for user specified thresholds δ_θ and δ_r .

We define p_2 to be in a $(\delta_\theta, \delta_r)$ -neighborhood of p_1 , or vice versa, if p_1 and p_2 are approximately similar.

DEFINITION 4.2. We say a 3-pattern $p = \{a_1, a_2, a_3, \theta, r\}$ is a core pattern if $(\delta_\theta, \delta_r)$ -neighborhood of p , say $N(p)$, contains at least a minimum number of 3-patterns, $MinPats$, which is a user-specified number.

Of course, for a meaningful threshold, the following inequality must be hold:

$$MinPats \geq \delta_f$$

PROPOSITION 4.2. Let C be a set of core patterns. Let \tilde{I} be the union of neighborhoods of core patterns.

$$\tilde{I} = \bigcup_{p \in C} N(p)$$

Then, it is NP-complete to find the minimum number of core points in C whose neighborhoods cover \tilde{I} .

We can reduce from maximum covering problem to show that Prop 4.2 is an NP-hard problem. For the efficiency reason, we propose a greedy algorithm in Algorithm 4.2 to find a subset of C to cover \tilde{I} . At line 6 in Algorithm 4.2, we greedily choose the point whose neighborhood has the maximum size. We update the remaining patterns and their neighborhoods based on the remaining images at lines 9 ~ 11.

ALGORITHM 4.2. Grouping Frequent 3-Patterns

Procedure Grouping Frequent 3-Patterns

begin

1. $F =$ find frequent 3-patterns
2. $C =$ find core patterns in F
3. $\tilde{C}, N = \emptyset$
4. $\tilde{I} = \bigcup_{p \in C} N(p)$
5. **while** ($\tilde{I} \neq \emptyset$) {
6. Choose a pattern $p \in C$ whose $|N(p)|$ is maximum
7. $\tilde{C} = \tilde{C} \cup \{p\}$
8. $N = N \cup \{N(p)\}$
9. $C = C - \{p\}$
10. $\tilde{I} = \tilde{I} - N(p)$
11. Update neighborhoods of the core patterns in C
12. }
13. return \tilde{C} and N

end

Using Algorithm 4.2, we can do the grouping of frequent 3-patterns, and express each frequent 3-pattern p by a pattern q in \tilde{C} which is the representative pattern of the neighborhood of p . Based on them we do the joining and pruning to find all frequent patterns in \tilde{I} .

5 Spatial Pattern-Based Clustering

In this section, we propose a clustering algorithm *SpaRClus* (**S**patial **R**elationship **P**attern-Based **H**ierarchical **C**lustering) based on a spatial pattern mining algorithm *SpIBag*. In Sec 5.1, we explain the main procedures of *SpaRClus*. In Sec 5.2, we show in detail how to perform postprocessing to the acquired clusters.

5.1 The *SpaRClus* Algorithm

The main idea of *SpaRClus* is that the support subset of each pattern forms a cover of the whole image set \tilde{I} . Since frequent patterns play an important role to characterize each image, we only use the support of frequent patterns. *SpaRClus* uses internally *SpIBag* algorithm to mine frequent patterns, and generates a hierarchical structure of image clusters based on their representative frequent patterns. When *SpIBag* algorithm generates a frequent n -pattern p , *SpaRClus* computes a scoring function of its support image set \tilde{I}_p and decides if p will be used or not to join with other n -patterns, which enables more pruning power than using *SpIBag* alone.

We first define several notations and a score function for further discussions.

DEFINITION 5.1. We denote $n(p, C)$ to be the size of the support image set of a pattern p in a cluster C . We denote $|C|$ to be the number of images in a cluster C . We denote $|I|$ to be the number of frequent 3-patterns in an image I . We define a score, or a probability, of an image I being in a cluster C by

$$(5.1) \quad Score(C, I) = \frac{\sum_{p: \text{freq 3-pattern in } I} n(p, C)}{|C| \times |I|}.$$

Note that the *Score* function ranges between 0 and 1. If no pattern in I appears in a cluster C , then the *Score* becomes 0. If all frequent 3-patterns in I appears in C , then the *Score* becomes 1. That is, an image gets a high score if most of its patterns are common through the images of C . We use this *Score* function to allocate an image I to the cluster with a highest score if I happens to be in several clusters. Also, we use this function later in Sec 5.2 to make clusters disjoint for the postprocessing part of *SpaRClus* framework.

DEFINITION 5.2. We define an entropy function E of a cluster C by

$$(5.2) \quad E(C) = \sum_{I \in C} -Score(C, I) \ln(Score(C, I)).$$

We say a cluster C is tight if $E(C) \leq \delta_e$ for a given threshold δ_e .

The entropy function $E(C)$ becomes 0 when all $Score$ functions are 1, in other words when all images in a cluster C contains only common frequent patterns. It increases monotonically when $Score(C, I)$ values decrease. In general, an entropy function tends to be low if a cluster C is tight. Based on this property, we use the entropy function E to see whether the current cluster, a support image set \tilde{I}_p of a pattern p , needs to be divided or not. This is a powerful tool to prevent us from generating explosive number of patterns. It enables us to reduce the number of patterns that will be required to make clusters of the image set \tilde{I} .

ALGORITHM 5.1. The *SpaRClus* Algorithm

Procedure *SpaRClus Algorithm*

begin

1. $\tilde{C} = \emptyset$
2. $F_3 = \text{find frequent 3-pattern groups of } \tilde{I}$
3. **for** ($k = 4; F_{k-1} \neq \emptyset; k++$) {
4. $F_k = \text{perform join in } SpIBag \text{ based on } (F_{k-1})$
5. **for** ($p \in F_k$) {
6. **if** ($E(\tilde{I}_p) \leq \delta_E$) **then** {
7. $\tilde{C} = \tilde{C} \cup \{\tilde{I}_p\}$
8. $F_k = F_k - \{p\}$
9. }
10. }
11. }
12. $\tilde{C} = \tilde{C} \cup \bigcup_{p \in F_{k-2}} \{\tilde{I}_p\}$
13. $\tilde{C} = \text{postprocessing}(\tilde{C})$
14. **return** \tilde{C}

end

At lines 2 ~ 4 in Algorithm 5.1, we adapt *SpIBag* to generate $(n + 1)$ frequent patterns. At lines 5 ~ 9, we perform another pruning step based on the entropy function E defined in Equation 5.2. If an entropy of a cluster C is below a given threshold δ_e , then we stop dividing C and prevent it from joining with other frequent patterns.

Once we finish the adapted joining and pruning steps, we do the postprocessing of the resulted clusters. We first perform merging similar clusters, and then make them disjoint. This process will be explained in the following subsection.

5.2 Postprocessing Clusters The clusters we get when the joining and pruning of *SpaRClus* is finished form leaves of a hierarchical graph structure. We call them *leaves of clusters*. Among these leaves, there might be similar clusters or even the same ones, since we did

not force the intermediate clusters to be disjoint to keep their characteristics consistent. Now, since we arrived at the last step, we do the postprocessing to find and merge similar clusters and make them disjoint.

Merging Similar Clusters From the *leaves of clusters*, all pairs of them are measured to see their similarity. The pair of clusters which are the most similar will be merged. When merging two clusters, we also union their representative patterns. This merging process looks like an agglomerative hierarchical clustering algorithm. To compute the similarity score between clusters, we define a formula based on an entropy function previously defined in Equation 5.2.

DEFINITION 5.3. We define a similarity measure between two clusters C and C' by

$$(5.3) \quad Sim(C, C') = E(C \cup C').$$

We merge two clusters C and C' when $Sim(C, C') \leq \delta_s$ for a given threshold δ_s .

The similarity function Sim in Equation 5.3 measures how tight it would be if two clusters are merged, recalling the fact that entropy function E in Equation 5.2 measured how tight a cluster is. It is quite natural to use the threshold δ_s to be the same with entropy threshold δ_e defined in Def 5.2. Once a cluster C is divided into two smaller clusters C_1 and C_2 by a threshold δ_e , they can not be merged to C again if we apply those thresholds to be the same ($\delta_s = \delta_e$), since

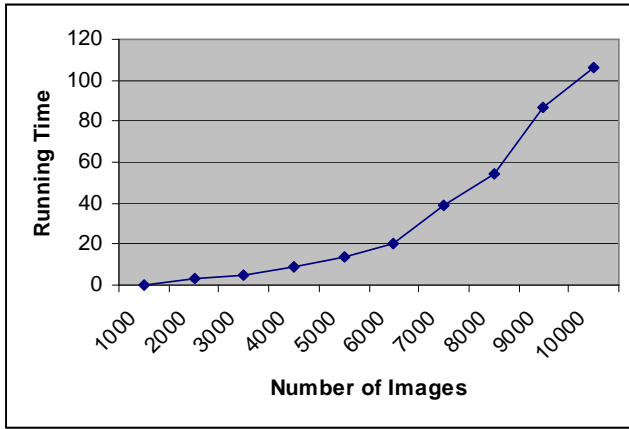
$$Sim(C_1, C_2) = E(C_1 \cup C_2) = E(C) > \delta_e.$$

Note that the two input clusters of Sim measure are already tight.

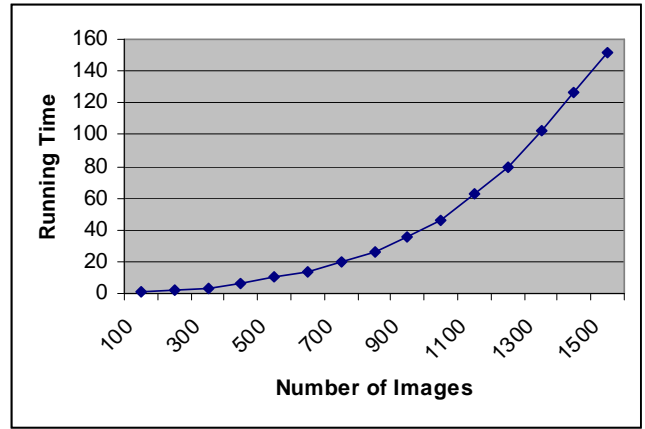
Making Clusters Disjoint In this step, we assume all clusters are from leaves and finished merging process. Thus, we can make clusters disjoint, since we do not need each cluster's preserved characteristics any more. For an image in several clusters, we compute its score of being in each cluster by the formula defined in Equation 5.1 and allocate it to the highest scored cluster. Once this process is done, we get rid of the clusters whose supports become lower than the minimum support δ_f .

6 Experiment

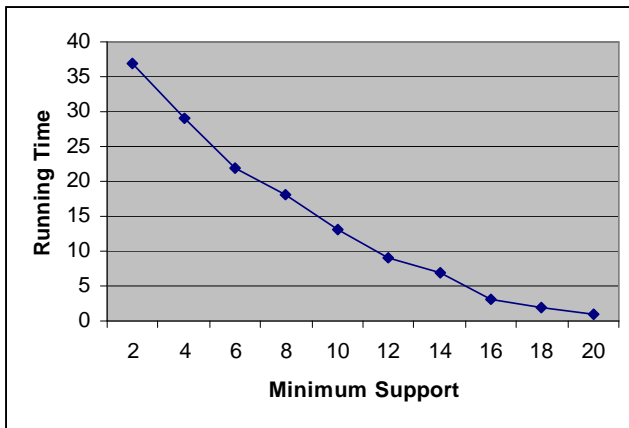
This section describes our experiments to show that *SpaRClus* is effective and efficient. We performed experiments on a various configuration of synthetic dataset to measure the efficiency, and generated 30 images with 5 clusters to measure the effectiveness of our algorithm. Experiments were conducted on a PC with a single 3.4GHz Pentium D CPU and 1GB RAM.



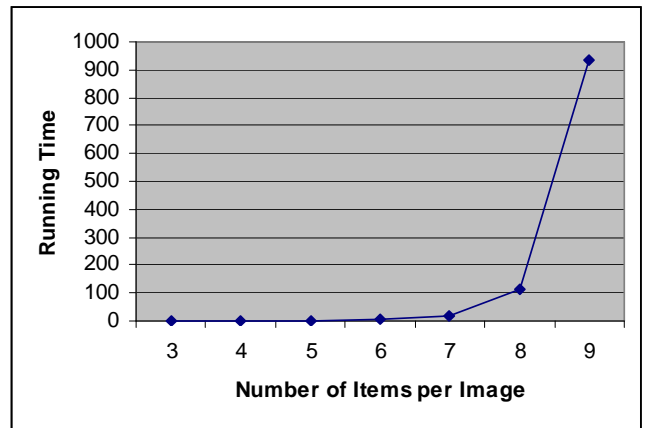
(a) Running time of *SpIBag* w.r.t the number of images when the minimum support is 20



(b) Running time of *SpIBag* w.r.t the number of images when the minimum support is 2



(c) Running time of *SpIBag* w.r.t the minimum support



(d) Running time of *SpIBag* w.r.t the number of items per image

Figure 4: Performance evaluations of *SpIBag* algorithm

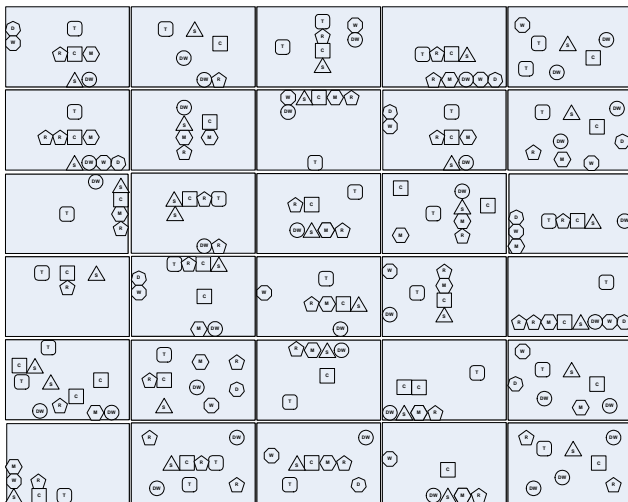
6.1 Experiments on Synthetic Data In this subsection, we explain several experiments to see the efficiency of the algorithm. Since the main computation of *SpaRClus* occurs in *SpIBag* procedure, we measured its running time in the following several experiments. We used thresholds $\delta_\theta = 50$ and $\delta_r = 1$ for experiments described in this subsection. We used degree unit instead of radian unit to measure angles.

In Fig.4(a), we measured the running time of *SpIBag* with respect to the number of images. To generate different number of images, we fixed the number of items for each image to be 5, the average number of images for each cluster to be 10. To change the total number of images, we fixed the average number of images per cluster and only changed the number of clusters. In this experiment, we set the minimum support to be 20, that is each cluster contains at least 20 images. It is $0.2 \sim 2\%$ of the total generated images which is a

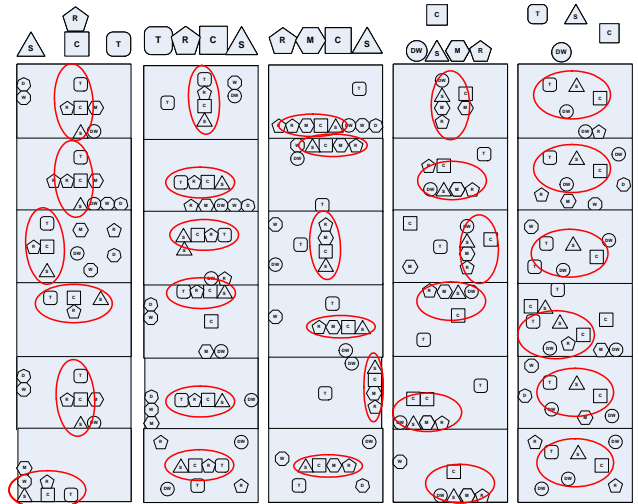
reasonable number to find meaningful clusters. We see in Fig.4(a) that the running time is less than 2 minutes for 10K images.

In Fig.4(b), we performed a similar experiment with Fig.4(a). We only changed the minimum number of supports to be 2 instead of 20 that required huge computational power and space to compute and store all possible frequent patterns. The line is showing an exponential growth of running time, as expected. This explosiveness happens to all frequent item set mining algorithms. We can delay the point of explosion by using entropy function of *SpaRClus* to prune generating clusters in the subgraph.

Fig.4(c) shows the running time of *SpIBag* with respect to the minimum supports. We generated 1,000 images with 100 clusters. Each cluster contained 10 images in average, and each image contained 5 items. The minimum support varied from 0.2 to 2 percent of



(a) Kitchen plan images



(b) Five clusters of kitchen plan images with their representative patterns

Figure 5: Clustering kitchen plan images using *SpaRClus* algorithm

the total images. To generate all possible patterns and their corresponding support images, it took less than 40 seconds.

In Fig.4(d), we performed an experiment to measure the running time of *SpIBag* with respect to the number of items per image. We used 100 images with 10 clusters. Each cluster had 10 images in average. Even with this small number of images, it started requiring explosive computational power and space, when there were 9 items in each image. The reason of this explosion is because we need to get all 3-patterns for each image. If there are n items in an image, it has to calculate $O(n^3)$ number of patterns. With the help of pruning power using an entropy function E defined in Equation 5.2, we can do the clustering more efficiently.

Algorithm *SpIBag* performed reasonably well enough to be applied to our clustering algorithm *SpaRClus*. We show the effectiveness of *SpaRClus* in Sec 6.2.

6.2 Experiments on Kitchen Plan Images

In this subsection, we explain an experiment to see the effectiveness of *SpaRClus*. We generated 30 images with 5 clusters based on *kitchen plan images* which were originally proposed in [8]. We defined 8 items including a cooktop (C), a sink(S), a refrigerator(R), a microwave(M), a dishwasher(DW), a table(T), a washer(W), and a dryer(D). Using this items, we designed 5 seed patterns, and created 6 images for each pattern. We allowed replicates of an item, and generated a semi-affine transformed patterns to see whether *SpaRClus* can correctly cluster images based on their

spatial information. We used thresholds $\delta_\theta = 20$, $\delta_r = 1$, $\delta_e = 1.5$ and minimum support to be 6.

In Fig.5(a), we show kitchen plan images used for the experiment. They are quite simple images, but it is really hard to cluster them manually. Even worse, since we are considering semi-affine transformations, it becomes almost impossible to cluster images by hands.

We applied *SpIBag* on kitchen plan images not using entropy pruning to compare it with *SpaRClus*. *SpIBag* generated 26 3-patterns, 13 4-patterns, and 5 5-patterns, which was total 44 different patterns. Then, we ran *SpaRClus* to get the clusters in Fig.5(b) and it generated 10 different patterns before the postprocessing step. We could prune generating lots of patterns by use of entropy function, but still generated more than 5 clusters, which is a good reason to do the postprocessing step. Since we did not make intermediate clusters be disjoint to fully utilize the characteristic of each cluster, *SpaRClus* might produce similar or even the same clusters for a different patterns. For this reason, even though we do not produce a larger pattern once we get a tight cluster, the leaves of the cluster graph might contain similar clusters.

Note that the first two clusters in Fig.5(b) had the same item bag of pattern but different spatial information, which was successfully detected by *SpaRClus* unlike previous approaches.

In this experiment, *SpaRClus* found all correct clusters, by which we can claim that *SpaRClus* really does the clustering on images which preserve the shapes under semi-affine transformations.

7 Discussion

In this paper, we assumed that images are already pre-processed and transformed into a well-defined input data of our *SpaRClus* framework. In fact, this preprocessing part is itself a hot research topic, and it would be another good project to apply *SpaRClus* to real image dataset.

As for the implementation part, we used *Apriori* algorithm which makes it easy to understand such a rather complicated concept of pattern mining in image data. But in the performance viewpoint to deal with massive image data, we need to develop a realistic and more efficient algorithm than merely using *Apriori* property. For example, to enable a new version of powerful image search engine, we need to have a real-time application.

The efficiency can be improved if we only consider neighborhoods of items, or k -nearest neighborhoods of items, to detect spatial patterns. But since we wanted to find similar spatial patterns under the scaling transformation also, we discarded the *neighborhood* idea. Sometimes when an image is zoomed in, new items might be detected between original items, which might lead to miss important patterns when we only consider neighborhoods of items. Moreover, we wanted to consider patterns across neighborhoods which might have important meanings to represent clusters. But still, there are rooms to adapt *neighborhood* idea which might be applied to some other areas including object identification.

There are still lots of works to do. We leave further use of *SpIBag* for indexing, searching and classifying image data for future works.

8 Conclusion

In this paper, we propose two algorithms *SpIBag* (**S**patial **I**tem **B**ag Mining) and *SpaRClus* (**S**patial **R**elationship **P**attern-Based **H**ierarchical **C**lustering). *SpIBag* discovers frequent spatial patterns invariant of semi-affine transformations. It utilizes two basic properties in image data: allowance of replicates of items in an image, and spatial information of a pattern. Based on this frequent pattern mining approach, we develop an image clustering algorithm *SpaRClus*. *SpaRClus* is a divisive hierarchical algorithm that performs clustering by dividing a large cluster to smaller ones. We contrive an entropy function that measures the tightness of a cluster. By use of this function, *SpaRClus* can decide where to stop dividing which enabled to prune further joining processes which makes the algorithm efficient. An extension version of this entropy function is also used for the postprocessing part, to measure the similarity of two clusters. We used various parameter settings of syn-

thetic data to show the efficiency of the algorithm, and proved the effectiveness by use of kitchen plan images.

References

- [1] Rakesh Agrawal and Ramakrishnan Srikant. Fast algorithms for mining association rules in large databases. In *VLDB'94, Proceedings of 20th International Conference on Very Large Data Bases, September 12-15, 1994, Santiago de Chile, Chile*, pages 487–499, 1994.
- [2] Simone Santini Amarnath Gupta Arnold W.M. Smeulders, Marcel Worring and Ramesh Jain. Content-based image retrieval at the end of the early years. *Pattern Analysis and Machine Intelligence, IEEE Transactions on Volume 22, Issue 12*, pages 1349–1380, December 2000.
- [3] Alexander C. Berg, Tamara L. Berg, and Jitendra Malik. Shape matching and object recognition using low distortion correspondences. In *CVPR '05: Proceedings of the 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05) - Volume 1*, pages 26–33, Washington, DC, USA, 2005.
- [4] N. S. Chang and K. S. Fu. Query-by pictorial-example. *IEEE Transactions on Software Engineering Volume 6, No.6*, pages 519–524, November 1980.
- [5] J. Dowe. Content-based retrieval in multimedia imaging. In *Proc. SPIE Storage and Retrieval for Image and Video Databases*, 1993.
- [6] Benjamin C. M. Fung, Ke Wang, and Martin Ester. Hierarchical document clustering using frequent itemsets. In *SDM*, 2003.
- [7] K. Grauman and T. Darrell. The pyramid match kernel: discriminative classification with sets of image features. In *ICCV 2005*, volume 2, pages 1458–1465 Vol. 2, 2005.
- [8] Wynne Hsu, Jing Dai, and Mong Li Lee. Mining viewpoint patterns in image databases. In *KDD '03: Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 553–558, New York, NY, USA, 2003. ACM Press.
- [9] Po-Whei Huang and Chu-Hui Lee. Image database design based on 9d-spa representation for spatial relations. *IEEE Transactions on Knowledge and Data Engineering*, 16(12):1486–1496, 2004.
- [10] Yan Huang, Hui Xiong, Shashi Shekhar, and Jian Pei. Mining confident co-location rules without a support threshold. In *SAC '03: Proceedings of the 2003 ACM symposium on Applied computing*, pages 497–501, New York, NY, USA, 2003. ACM Press.
- [11] A. Gupta A. Hampapur B. Horowitz R. Humphrey R. Jain J. Bach, C. Fuler and C. Shu. The virage image search engine: An open framework for image management. In *Proc. SPIE Conference on Storage and Retrieval for Image and Video Databases IV*, 1996.
- [12] C. E. Jacobs, A. Finkelstein, and D. H. Salesin. Fast multiresolution image querying. In *Proc. of SIGGRAPH 95, Annual Conference Series*, pages 277–286, August 1995.

- [13] Krzysztof Koperski and Jiawei Han. Discovery of spatial association rules in geographic information databases. In *Proc. 4th Int. Symp. Advances in Spatial Databases, SSD*, volume 951, pages 47–66. Springer-Verlag, 6–9 1995.
- [14] Svetlana Lazebnik, Cordelia Schmid, and Jean Ponce. Beyond bags of features: Spatial pyramid matching for recognizing natural scene categories. In *CVPR '06: Proceedings of the 2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 2169–2178, Washington, DC, USA, 2006.
- [15] Anthony J. T. Lee, Ruey-Wen Hong, Wei-Min Ko, Wen-Kwang Tsao, and Hsiu-Hui Lin. Mining spatial association rules in image databases. *Inf. Sci.*, 177(7):1593–1608, 2007.
- [16] Fei-Fei Li and Pietro Perona. A bayesian hierarchical model for learning natural scene categories. In *CVPR '05: Proceedings of the 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05) - Volume 2*, pages 524–531, Washington, DC, USA, 2005. IEEE Computer Society.
- [17] David G. Lowe. Object recognition from local scale-invariant features. In *ICCV '99: Proceedings of the International Conference on Computer Vision-Volume 2*, page 1150, Washington, DC, USA, 1999.
- [18] W. Niblack J. Ashley Q. Huang B. Dom M. Gorkani J. Hafner D. Lee D. Petkovic D. Steele M. Flicker, H. Sawhney and P. Yanker. Query by image and video content: the QBIC system. *IEEE Computer*, Volume 28, No.9, pages 23–32, September 1995.
- [19] Hassan H. Malik and John R. Kender. High quality, efficient hierarchical document clustering using closed interesting itemsets. *ICDM*, 0:991–996, 2006.
- [20] Carlos Ordonez and Edward Omiecinski. Discovering association rules based on image content. In *ADL '99: Proceedings of the IEEE Forum on Research and Technology Advances in Digital Libraries*, page 38, Washington, DC, USA, 1999.
- [21] Jong Soo Park, Ming-Syan Chen, and Philip S. Yu. An effective hash-based algorithm for mining association rules. In *SIGMOD '95: Proceedings of the 1995 ACM SIGMOD international conference on Management of data*, pages 175–186, New York, NY, USA, 1995.
- [22] M.A. Rubin, W.T. Freeman, K.P. Murphy, and A. Torralba. Context-based vision system for place and object recognition. In *ICCV*, 2003.
- [23] Ashoka Savasere, Edward Omiecinski, and Shamkant B. Navathe. An efficient algorithm for mining association rules in large databases. In *VLDB '95: Proceedings of the 21th International Conference on Very Large Data Bases*, pages 432–444, San Francisco, CA, USA, 1995.
- [24] M. Stricker and M. Swain. The capacity of color histogram indexing. In *Proc. Computer Vision and Pattern Recognition*, 1994.
- [25] M. Swain and D. Ballard. Color Indexing. *International Journal of Computer Vision*, Volume 7, No. 1, pages 11–32, November 1991.
- [26] Hannu Toivonen. Sampling large databases for association rules. In *VLDB '96: Proceedings of the 22th International Conference on Very Large Data Bases*, pages 134–145, San Francisco, CA, USA, 1996.
- [27] Michel Vidal-Naquet and Shimon Ullman. Object recognition with informative features and linear classification. In *ICCV '03: Proceedings of the Ninth IEEE International Conference on Computer Vision*, page 281, Washington, DC, USA, 2003.
- [28] James Ze Wang, Gio Wiederhold, Oscar Firschein, and Sha Xin Wei. Content-based image indexing and searching using daubechies' wavelets. *Intl. Journal of Digital Libraries (IJODL)*, 1(4):311–328, 1998.
- [29] Thomas S. Huang Yong Rui and Shih-Fu Chang. Image Retrieval: Current Techniques, Promising Directions, and Open Issues. *Journal of Visual Communication and Image Representation*, Volume 10, Issue 1, pages 39–62, March 1999.
- [30] Junsong Yuan, Ying Wu, and Ming Yang. From frequent itemsets to semantically meaningful visual patterns. In *KDD '07: Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 864–873, New York, NY, USA, 2007.
- [31] Osmar R. Zaiane, Jiawei Han, and Hua Zhu. Mining recurrent items in multimedia with progressive resolution refinement. *ICDE*, 00:461, 2000.
- [32] Xin Zhang, Nikos Mamoulis, David W. Cheung, and Yutao Shou. Fast mining of spatial collocations. In *KDD '04: Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 384–393, New York, NY, USA, 2004.