# Efficiently Computing Weighted Proximity Relationships in Spatial Databases

Xuemin Lin[1] Xiaomei Zhou[1] Chengfei Liu[2] Xiaofang Zhou[3]

[1] School of Computer Science and Engineering, University of New South Wales
Sydney, NSW 2052, Australia. *lxue@cse.unsw.edu.au*
[2] *School of Computer and Information Science, University of South Australia
Adelaide, SA 5095, Australia.* Liu@cs.unisa.edu.au
[3] Department of Computer Science and Electrical Engineering
University of Queensland, QLD 4072, Australia. *zxf@csee.uq.edu.au*

**Abstract.** Spatial data mining recently emerges from a number of real applications, such as real-estate marketing, urban planning, weather forecasting, medical image analysis, road traffic accident analysis, etc. It demands for efficient solutions for many new, expensive, and complicated problems. In this paper, we investigate the problem of evaluating the top $k$ distinguished "features" for a "cluster" based on weighted proximity relationships between the cluster and features. We measure proximity in an average fashion to address possible nonuniform data distribution in a cluster. Combining a standard multi-step paradigm with new lower and upper proximity bounds, we presented an efficient algorithm to solve the problem. The algorithm is implemented in several different modes. Our experiment results not only give a comparison among them but also illustrate the efficiency of the algorithm.

*Keywords*: Spatial query processing and data mining.

## 1 Introduction

Spatial data mining is to discover and understand non-trivial, implicit, and previously unknown knowledge in large spatial databases. It has a wide range of applications, such as demographic analysis, weather pattern analysis, urban planning, transportation management, etc. While processing of typical spatial queries (such as joins, nearest neighbouring, KNN, and map overlays) has been received a great deal of attention for years [2, 3, 20], spatial data mining, viewed as advanced spatial queries, demands for efficient solutions for many newly proposed, expensive, complicated, and sometimes ad-hoc spatial queries.

Inspired by a success in advanced spatial query processing techniques [2, 3, 8, 20], relational data mining [1, 18], machine learning [7], computational geometry [19], and statistics analysis [11, 21], many research results and system prototypes in spatial data mining have been recently reported [2, 4, 5, 10, 12, 14, 16]. The existing research not only tends to provide system solutions but also covers quite a number of special purpose solutions to ad-hoc mining tasks. The results include efficiently computing spatial association rules [13], spatial data classification and generalization [10, 14, 16], spatial prediction and trend analysis [5], clustering and cluster analysis [4, 12, 17, 23], mining in image and raster databases [6], etc.

Clustering has been shown one of the most useful tools to partition and categorize spatial data into *clusters* for the purpose of knowledge discovery. A number of efficient algorithms [4, 17, 22, 23] have been proposed. While most existing clustering algorithms are effective to construct "clusters", they are ineffective to provide the reasons why the clusters are there spatially. Moreover, in many applications clusters are naturally existing; for example, a cluster could be a residential area. Therefore, it is equally important, if not more, to find out spatial properties of clusters based on their surrounding "features". The proximity is one of the most common measurements to represent the relationship between clusters and features. In [12], the problem of computing $k$ closest features surrounding a set (cluster) of points in two dimensional space based on average/aggregate proximity relationships was investigated.

The problem of computing $k$ closest features has a number of useful real applications. For instance, in a spatial database for the real-estate information, a set of points represents a residential area where each point represents a house/land parcel. A polygon corresponds to a vector representation of feature, such as a lake, golf course, school, motor way, etc. In this application, buyers or developers may want to know why some residential area is so expensive. Consequently they may want to know the $k$ closest features. As pointed in [12], in such an application it is better to measure the proximity relationship in an average fashion to address a possible nonuniform data distribution. Furthermore, in such an application it may be more important to know that a residential area is only 5 kilometres away from the downtown area rather than 500 meters away from a swimming pool. This suggests that we should put weights when evaluate features, so that the obtained top $k$ features are also based on their importance. In this paper, we investigate the problem of computing the *weighted $k$ closest features* (WK); that is, each feature is associated with a weight. We will formally define the WK problem in the next section.

In WK, we assume that the "proximity value" between a cluster and feature has not been pre-computed, nor stored in the database. A naive way to solve WK is to first precisely compute the proximity value between each feature and a given cluster, and then to solve the WK problem. However, in practice there may be many features far from being part of solution to WK; and thus they should be ruled out by a fast filtering technique. The filtering technique developed in [12] for unweighted problem is not quite suitable to WK, because it specifically designed to solve unweighted problem. Motivated by these, our algorithm adopts a standard multi-step technique [2, 12, 13] in combining with novel and powerful pruning conditions to filter out uninvolved features. The algorithm has been implemented in several different modes for performance evaluation. Our experiments clearly demonstrate the efficiency of the algorithm.

The rest of the paper is organized as follows. In section 2, we present a precise definition of WK as well as a brief introduction of an adopted spatial database architecture. Section 3 presents our algorithm for solving WK. Section 4 reports our experimental results. In section 5, a discussion is presented regarding various modifications of our algorithm; this will be together with the conclusions. Note, due to the space limitation we do not provide the proofs of the theorems in this paper; the interested readers may refer to a long version [15] of the paper for these.

## 2 Preliminary

In this section we precisely define the WK problem. A *feature* $F$ is a *simple* and *closed* polygon [19] in the 2-dimensional space. A set $C$ of points in the two dimensional space is called *cluster* for notation simplicity. Following [12], we assume that in WK a cluster is always *outside* [19] a feature. Note that this assumption may support many real applications. For instance, in real-estate data, a cluster represents a set of land parcels, and a feature represents a man-made or natural place of interest, such as lake, shopping center, school, park, and entertainment center. Such data can be found in many electronic maps in a digital library.

To efficiently access large spatial data (usually tera-bytes), in this paper we adopt an extended-relational and a SAND (spatial-and-non-spatial database) architecture [3]; that is, a spatial database consisting of a set of spatial objects and a relational database describing non-spatial properties of these objects.

Below we formally define WK. In WK, the input consists of: 1) a cluster $C_0$; 2) a set $\Pi = \{\pi_j : 1 \leq j \leq m\}$ of groups of features (i.e., each $\pi_j$ is a group of features); and 3) $k$ to indicate the number of features to be found.

Given a feature $F$ and a point $p$ outside $F$, the length of the actual (working or driving) shortest path from $p$ to $F$ is too expensive to compute in the presence of tens of thousands of different roads. In WK, we use the shortest Euclidean distance from $p$ to a point in the boundary of $F$, denoted by $d(p, F)$, to reflect the geographic proximity relationship between $p$ and $F$. We believe that on average, the length of an actual shortest path can be reflected by $d(p, F)$. We call $d(p, F)$ the *distance* between $p$ and $F$. Note that $F$ may degenerate to a line or a point. Moreover, for the purpose of computing lower and upper proximity bounds in Section 3, the definition of $d(p, F)$ should be extended to cover the case when $p$ is inside or on the boundary of $F$; that is, $d(p, F) = 0$ if $p$ is inside of or on the boundary of $F$.

To address a possible arbitrary distribution of the points in $C$, we use the following *average proximity* value to quantitatively model the proximity relationship between $F$ and $C$:

$$AP(C, F) = \frac{1}{|C|} \sum_{p \in C} d(p, F).$$

As mentioned earlier, in WK we will rank the importance of a feature by a positive value. More important a feature is, smaller its weight is. A weight can be assigned to a group of features by either a user or the system default. A set

$\{w_j : 1 \leq j \leq m\}$ of positive values is also part of the input of WK. WK can now be modelled to find the $k$ features in $\sum_{j=1}^{m} \pi_j$ such that those $k$ features lead to the $k$ smallest values of the following function:

$$WAP(C_0, F) = W_F AP(C_0, F) \text{ where } F \in \sum_{j=1}^{m} \pi_j.$$

Here, $W_F$ is the weight of $F$; that is, $W_F = w_j$ for $F \in \pi_j$. Note that in [12], the proximity between a cluster and a feature is measured in an aggregate fashion; that is, $\sum_{p \in C} d(p, F)$ is used there instead of using $\frac{1}{|C|} \sum_{p \in C} d(p, F)$. However, it should be clear that to compute the $k$ closest features to a given cluster these two measurements are equivalent. This means that the top $k$ feature problem in [12] is a special case of WK where all the weights are 1.

## 3 Algorithms for Solving WK

In this section, we present an efficient algorithm for solving WK. The algorithm is denoted by CWK, which stands for **C**omputing the **W**eighted $K$ closest features.

An immediate way (brute-force) to solve WK is to 1) compute $WAP(C_0, F)$ for each pair of a given cluster $C_0$ and a feature $F$, and then to 2) compute the $k$ closest features based on their WAP values. Note that $WAP(C_0, F)$ can be easily computed in $O(|C_0||F|)$ according to the definition of $WAP(C, F)$. Consequently, the brute-force approach runs in time $O(|C_0| \sum_{j=1}^{m} \sum_{F \in \pi_j} |F|)$. Practically, there may be many features involved in the computation; each feature may have many edges; and the given cluster may have many points. These make the brute-force approach probably computational prohibitive in practice. Our experiment results in Section 4.3 confirm this.

One possible way to resolve this problem is to adopt a multi-step paradigm [2, 12, 13]. First, we use a filtering step to filter out the features, and then do precise computation of $WAP$ values for those candidate features only. A simple and very effective technique was proposed in [12] to solve WK with all weight equal to 1: *draw a circle encompassing the cluster and then keep the features with an intersection to the circle as the candidates.*

If we want to apply the above technique generally to WK with different weights, we may have to draw different circles for different groups of features due to different weights. Because for each group of features we have to keep at least $k$ features, there may be too many candidates left and the filter may not be powerful enough,

In this paper, we propose a filtering technique based on the lower and upper bound for WAP values. Instead of computing the actual value of $WAP(C_0, F)$ in quadratic time $O(|C||F|)$, we may compute a lower bound and an upper bound for $WAP(C_0, F)$ in a linear time $O(|F|)$ with respect to the size of $F$. By these bounds, for the given cluster $C_0$, we can rule out the features, which are definitely not closest to $C_0$ in a weighted sense; Thus we do not have to precisely compute the weighted average proximity values between these eliminated features and $C_0$. This is the basic idea of our algorithm. In our algorithm CWK, we have not integrated our algorithm into a particular spatial index, such as $R$-trees, $R^+$-trees, etc, due to the following reasons.

- There may be no spatial index built.
- The WK problem may involve many features from different tables/electronic thematic maps; and thus, spatial index built for each thematic map may be different. This brings another difficulty when making use of spatial indices.
- A feature or a cluster, which is qualified in WK, may be only a part of a stored spatial object; for instance, a user may be interested only in certain part of a park. This makes a possible existing index based on the stored spatial objects not quite applicable.
- The paper [12] indicates the existing spatial indexing techniques do not necessarily support well the computation of *aggregate* distances; the argument should be also applied to average distance computation.

The algorithm CWK consists of the following 3 steps:

**Step 1:** Read the cluster $C_0$ into buffer.
**Step 2:** Read features batch by batch into buffer, compute lower and upper bounds of $WAP(C_0, F)$ for the given cluster $C_0$. Then determine whether or not $F$ should be kept for the computation of $WAP(C_0, F)$.
**Step 3:** Apply the above brute-force method to the remaining features to solve problem WK.

In the next several subsections we detail the algorithm step by step. Clearly, a success of the algorithm CWK largely relies on how good the lower and upper bounds of $WAP$ are. The goodness of lower and upper bounds means two things: 1) the bounds should be reasonably tight, and 2) the corresponding computation should be fast. We first present the lower and upper bounds.

Note that for presentation simplicity, the algorithms presented in the paper are restricted to the case when features and clusters qualified in WK are stored spatial objects in the database. However, they can be immediately extended to cover the case when a feature or a cluster is a part of a stored object.

### 3.1 Lower and Upper Bounds for Average Proximity

In this subsection, we recall first some useful notation. The *barycenter* (*centroid*) of a cluster $C$ is denoted by $b(C)$. A *convex* [19] polygon encompassing a feature $F$ is called a bounding convex polygon of $F$. The smallest bounding convex polygon of $F$ is called the *convex hull* [19] of $F$ and is denoted by $P_F$. An *isothetic* [19] rectangle is orthogonal to the coordinate axis. The minimum bounding rectangle of $F$ refers to the minimum isothetic bounding rectangle of $F$ and is denoted by $R_F$.

Given two minimum bounding rectangles $R_C$ and $R_F$ respectively for a cluster $C$ and a feature $F$, an immediate idea is to use the shortest distance and the longest distance between $R_C$ and $R_F$ to respectively represent a lower bound and an upper bound of $AP(C, F)$. However, this immediate idea has two problems.

The first problem is that when two rectangles intersect with each other (note that in this case $C$ and $F$ do not necessarily have an intersection), the shortest and longest distances between $R_C$ and $R_F$ are not well defined. The second problem is that the bounds may not be very tight even if the two rectangles do not intersect. These also happen similarly for convex hulls. Below, we present new and tighter bounds.

Our lower bound computation is based on the following Lemma.

**Lemma 1.** $\sum_{i=1}^{K} \sqrt{x_i^2 + y_i^2} \geq \sqrt{(\sum_{i=1}^{K} x_i)^2 + (\sum_{i=1}^{K} y_i)^2}$

**Theorem 2.** *Suppose that $C$ is a cluster, $F$ is a feature, and $P$ is either the convex hull or the minimum bounding rectangle of $F$. Then, $AP(C,F) \geq d(b(C),P)$; in other words $d(b(C),P)$ is a lower bound of $AP(C,F)$.*

Clearly, the above lower bound is tighter than the shortest distance between two bounding rectangles.

Suppose that $p$ is an arbitrary point. For a cluster, we use $\lambda(p,C)$ to denote the maximum distance between $p$ to a point in $C$. Below is an upper bound of $AP$.

**Theorem 3.** *Suppose that $p$ is an arbitrary point, $C$ is a cluster, and $F$ is a feature. Then $AP(C,F) \leq d(p,F) + \lambda(p,C)$.*

It is clear the right hand side in the inequality of Theorem 3 can be used as an upper bound of $AP$; and can be computed in time $O(|F|)$ once the computation of $\lambda(p,C)$ is done. We believe that this bound may be tighter than the longest distance between the minimum bounding rectangles. However, we cannot generally prove this because the tightness of the upper bound depends on the choice of $p$. In our algorithm, we will choose the *centroid* of a cluster $C$ in the upper bound computation since it has to be used to obtain a lower bound. Note that generally $d(b(C),P_F)$ and $d(b(C),F)$ respectively in the lower and upper bounds cannot replace each other if $F$ is not a convex polygon.

### 3.2 Read in the relevant cluster

In Step 1, we first read in the cluster $C_0$, which are specified by a user, into buffer. Then, we compute the centroid $b(C_0)$ and the maximal distance $\lambda(b(C_0),C_0)$. Clearly, this step takes linear time with respect to the size of $C_0$.

### 3.3 Read in and Filter out Features

This subsection presents Step 2. Consider that the total size of features to be processed may be too large to fit in buffer simultaneously. Features should be read into buffer batch by batch.

Note that in WK, our primary goal is to make a candidate set as close as possible to the actual solution; this means a small set of candidates will be expected to be left in the main memory after filtering step. Therefore, we may store the detail of the candidates in the main memory for a further processing. Particularly, the filtering step developed in CWK is to:

1. initialize the candidate set by assigning the first $k$ features, and then,
2. examine the rest of the features one by one to determine whether or not they are candidates. In case that a new candidate is added, we should also check whether or not certain features included in the candidate set should be removed.

The following lemma can be immediately verified.

**Lemma 4.** *For a cluster $C$ and a feature $F$ with weight $W_F$,*

$$W_F d(b(C), P_F) \leq WAP(C, F) \leq W_F(d(p, F) + \lambda(p, C)).$$

Lemma 4 means that $W_F d(b(C), P_F)$ is the lower bound of $WAP(C, F)$ denoted by $LB_F$, while $W_F(d(p, F) + \lambda(p, C))$ is the upper bound of $WAP(C, F)$ denoted by $UB_F$.

**Lemma 5.** *Suppose $\pi$ is a set of candidate features, $Y$ is the k'th smallest $UB_F$ for every $F \in \pi$. Then, $F'$ is a feature in the solution of WK only if $WAP(C, F') \leq Y$. Thus, $LB_{F'} \leq Y$.*

Suppose that a new feature $F$ is processed, Lemma 5 says that if $LB_F$ for $F$ is greater than $Y$, than $F$ should not be considered as part of the solution of WK.

Consider the situation that a new $F$ is added to $\pi$ and the $Y$ is updated (changed smaller). Then, a feature $F'$ in $\pi$ may no longer be a candidate if $LB_{F'} > Y$; and thus we have to remove $F'$ from $\pi$. A naive way to do this is to scan the whole $\pi$ to determine whether or not there is feature to be removed. To speed up this process, we can divided $\pi$ into two parts $A$ and $B$, where $B$ is the set of candidates to be removed. We make $A$ and $B$ two ordered linked lists to store the candidates, let $A$ has the fixed length of $k$, and use $X$ to store the current largest lower bound of $F$ in $A$. Each element $F$ in $A$ or $B$ stores the identifier $FID$, $LB_F$ and $UB_F$, and the spatial description of $F$. $A$ and $B$ are initially set to empty. The elements in $A$ and $B$ are sorted on the lexicographic order of $(LB_F, UB_F)$ for each $F$.

Specifically, to process a $F$, CWK first computes the values of $LB_F$ and $UB_F$ for $WAP(C, F)$. Then, add the first $k$ features to $A$ according to a lexicographic order of $(LB_F, UB_F)$. For the rest of the features, if $LB_F \leq Y$, it will be a candidate and kept in $A$ or $B$. More specifically, $F$ will be added to $A$ in a proper position if $(LB_F, UB_F) < (LB_{F'}, UB_{F'})$, where $F'$ is the last element of $A$; and then $F'$ will be moved to $B$. When $(LB_F, UB_F) \geq (LB_{F'}, UB_{F'})$, $F$ will be added to $B$. Once a new $F$ is added to the $A$ or $B$, $X$ and $Y$ should be updated. Each time after $Y$ is reduced, we also need to check $B$ to remove those features whose lower bounds are bigger than $Y$. Below is a precise description of Step 2.

**Step 2 in CWK:**
$A \leftarrow \emptyset$; $B \leftarrow \emptyset$;
Read in features;
**for** the first $k$ features in $\pi$ **do**
  { compute lower and upper bounds $LB_F$ and $UB_F$ for each feature $F$;
  keep them in $A$ and compute $X$ and $Y$; }
$\pi = \pi$ - { the first $k$ features in $\pi$ };
**for** each $F \in \pi$ **do**
  { compute $LB_F$ and $UB_F$;
  **if** $LB_F < Y$ **then**
    **if** $LB_F > X$ or $(LB_F = X$ and $UB_F \geq Y)$ **then** $B \leftarrow F$ **else**
    { move last element of $A$ into $B$;
    $A \leftarrow F$;
    Update $X$ and $Y$;
    **if** $Y$ reduced **then** remove features $F \in B$ with $UB_F > Y$; }
  }

Once a batch of features are processed by the above procedure, we do not keep them in buffer except the candidate features in $A$ and $B$.

### 3.4 Precise Computation

After filtering process in Step 2, the full information of remaining features for solving WK is kept in $A$ and $B$. Then, we apply this information to perform a precise computation of $WAP$ values using brute-force method.

### 3.5 Complexity of CWK

In Step 1, we read in the user specified cluster $C_0$ and do some relevant computation, this step takes linear time with respect to the size of $C_0$.

In Step 2, we do the filtering process to read in and filter out features, the main expenses here are to compute the lower and upper bounds of $WAP$ for each $F$, and to possibly insert a $F$ to $A$ or $B$. These together run in $O(\sum_{i=1}^{n} |F_i| + n\log(|A| + |B|))$, here $n$ is the number of total features.

Step 3 takes time $O(\sum_{F \in (A \cup B)} |C||F|)$ to compute $WAP$ for the remaining features.

Note that the brute-force algorithm runs in time $O(\sum_{i=1}^{n} |C||F_i|)$. Clearly, in practice, the time complexity of the brute-force method is much higher than that of CWK, because $|A \cup B|$ is much smaller than $n$. This is confirmed by our experiment results in Section 4.

### 3.6 Variations of CWK

We implement the algorithm CWK in three different modes in order to evaluate the performance. The differences are the bounding shapes of a feature to be taken while doing the filtering process. The first mode is denoted by CWK-R, which uses only minimal bounding rectangles of features to compute the lower and upper bounds of $WAP(C, F)$. An alternative mode to CWK-R is to use the minimum bounding convex hulls instead of the minimal bounding rectangles, we denote this mode by CWK-P. The third mode, denoted by CWK-RP, adopts a multiple-filtering technique: 1) minimal bounding rectangles are firstly used to obtain the ordered linked list $A$ and $B$, and then, 2) the minimum bounding convex hulls are computed for features stored in $A$ and $B$ to repeat Step 2 to get two new ordered linked list $A'$ and $B'$ before processing Step 3.

In next section, we will report our experiment results regarding the performance of the brute-force method, CWK-P, CWK-R, and CWK-RP.

## 4 Implementation and Experiment Results

The brute-force method and the three modes of CWK algorithm have been implemented by C++ on a Pentium I/200 with 128 MB of main memory, running Window-NT 4.0. In our experiments, we evaluated the algorithms for efficiency and scalability. Our performance evaluation is basically focused on Step 2 onwards, because the methods [3] of reading in clusters and features are not our contribution. Therefore, in our experiment we record only the CPU time but exclude I/O costs.

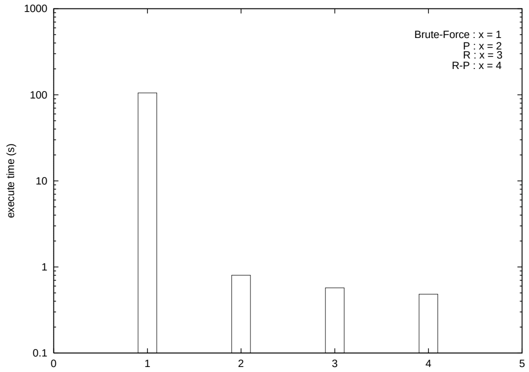
In the experiments below, we adopt a common set of parameters: 1) a feature has 30 edges on average, 2) a cluster has 200 points on average, and 3) the features are grouped into 20 groups.

In our first experiment, we generate a database with 50 clusters and 5000 features, and $k = 5$. We implement our algorithm for each cluster. The experiment results depicted in Figure 1 are the average time. Note the algorithm CWK-P, CWK-R, CWK-RP are respective abbreviated to "P", "R", "R-P" in the diagram.

From the first experiment, we can conclude that the brute-force method is practically very slow. Another appearance is that CWK-P is slower than CWK-R. Intuitively this is because that in CWK-P, the computation of a lower bound for each feature is more expensive than that in CWK-R. We also observed that CWK-RP appears the most efficient. This because that the second filtering phase in CWK can still filter out some features; and thus the number of features left for the precise computation is reduced. In short, we believe that CWK-P should be intuitively and significantly slower than CWK-R and CWK-RP when the number of features increases; this has been confirmed by the second experiment.

The second and third experiments have been undertaken through two dimensions. In the second experiment, we fix the $k$ to be 15 while the number of features varies from 5000 to 50,000. Again, we run the 3 algorithms for each cluster and record the average time. The results are depicted in Figure 2.

In the third experiment, we fix the number of features to be $50,000$ while $k$ varies from 5 to 75. Each algorithm has been run against each cluster and the average time is recorded. The experiment results are depicted in Figure 3.

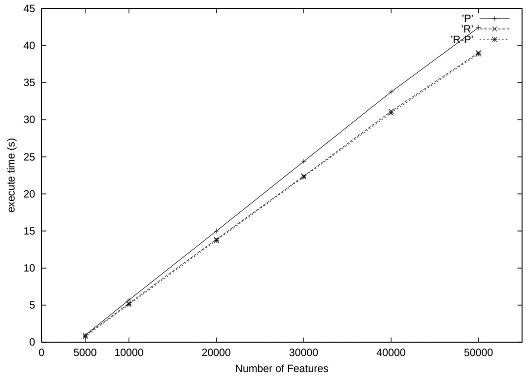



(a) histogram presentation

| Algorithm | Average Run Time (s) |
|---|---|
| Brute-force | 105.055 |
| P | 0.802 |
| R | 0.572 |
| R-P | 0.483 |

(b) table presentation

**Fig. 1.** Average execution time for four algorithms




(a) graphic presentation

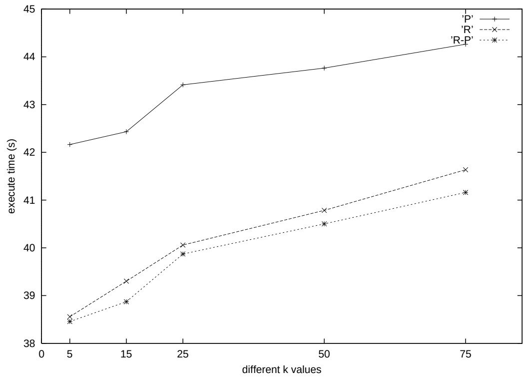
| #FE | Algorithm | Average Run Time (s) |
|---|---|---|
| 5000 | P | 0.975 |
| | R | 0.906 |
| | R-P | 0.760 |
| 10000 | P | 5.697 |
| | R | 5.244 |
| | R-P | 5.097 |
| 20000 | P | 14.988 |
| | R | 13.881 |
| | R-P | 13.720 |
| 30000 | P | 24.375 |
| | R | 22.400 |
| | R-P | 22.291 |
| 40000 | P | 33.747 |
| | R | 31.140 |
| | R-P | 30.922 |
| 50000 | P | 42.433 |
| | R | 39.302 |
| | R-P | 38.871 |

(b) table presentation

**Fig. 2.** Average execution time for three algorithms by different DB sizes

These three experiments suggest that CWK-RP and CWK-R is faster than CWK-P on average, and the performance of CWK-RP is faster than CWK-R. From the second and third experiment results, we see that the choices of different $k$ values and different DB sizes will not change the above observation. The second experiment also shows the scalability of algorithm CWK.

The three conducted experiments suggest that our algorithm is efficient and scalable. Furthermore, we can see that although an application of convex hulls to




(a) graphic presentation

| K | Algorithm | Average Run Time (s) |
|---|---|---|
| 5 | P | 42.163 |
| | R | 38.556 |
| | R-P | 38.456 |
| 15 | P | 42.432 |
| | R | 39.032 |
| | R-P | 38.871 |
| 25 | P | 43.413 |
| | R | 40.058 |
| | R-P | 39.871 |
| 50 | P | 43.764 |
| | R | 40.784 |
| | R-P | 40.502 |
| 75 | P | 44.264 |
| | R | 41.635 |
| | R-P | 41.161 |

(b) table presentation

**Fig. 3.** Average execution time for three algorithms by different K

the filtering procedures is more accurate than an application of minimal bounding rectangles, it is more expensive to use directly. Thus, the best use of convex hulls should follow an application of minimal bounding rectangles; that is, it is better to apply the CWK-RP mode.

## 5 Conclusion and Remarks

In this paper, we investigated the weighted top $k$ features problem regarding average/aggregate proximity relationships. We presented an efficient algorithm based on several new pruning conditions, as well as various different modes of the algorithms. Our experiment results showed that the algorithm is quite efficient.

The problem WK, as well as the algorithm CWK, may be either generalized or constrained according to various applications. The interested readers may refer to our full paper [15] for details.

## References

1. R. Agrawal and R. Srikant, Fast Algorithms for Mining Association Rules, *Proceedings of the 20th VLDB Conference*, 487-499, 1994.
2. M. Ankerst, B. Braunmuller, H.-P. Kriegel, T. Seidl, Improving Adaptable Similarity Query Processing by Using Approximations, *Proceedings of the 24th VLDB Conference*, 206-217, 1998.
3. W. G. Aref and H. Samet, Optimization Strategies for Spatial Query Processing, *Proceedings of the 17th VLDB Conference*, 81-90, 1991.

4. M. Ester, H.-P. Kriegel, J. Sander and X. Xu, A density-based algorithm for discovering clusters in large spatial databases, *Proceedings of the Second International Conference on Data Mining KDD-96*, 226-231, 1996.
5. M. Este, H.-P. Kriegel, J. Sander, Spatial Data Mining: A Database Approach, *SSD'97*, LNCS 1262, 47-65, 1997.
6. U. M. Fayyad, S. G. Djorgovski, and N. Weir, Automating the analysis and cataloging of sky surveys, *Advances in Knowledge Discovery and Data Mining*, AAAI/MIT Press, 1996.
7. U. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy, Eds. *Advances in Knowledge Discovery and Data Mining*, AAAI/MIT Press, Menlo Park, CA, 1996.
8. R. H. Guting, An Introduction to Spatial Database Systems, *VLDB Journal*, 3(4), 357-400, 1994.
9. J. Han, Y. Cai, and N. Cercone, Dynamic Generation and Refinement of Concept Hierarchies for Knowledge Discovery in Databases, *IEEE Trans. knowledge and Data Engineering*, 5, 29-40, 1993.
10. J. Han, K. Koperski, and N. Stefanovic, GeoMiner: A System Prototype for Spatial Data Mining, *Proceedings of 1997 ACM-SIGMOD International Conference on Management*, 553-556, 1997.
11. L. Kaufman and P. J. Rousseeuw, *Finding Groups in Data: an Introduction to Cluster Analysis*, John Wiley & Sons, 1990.
12. E. M. Knorr and R. T. Ng, Finding Aggregate Proximity Relationships and Commonalities in Spatial Data Mining, *IEEE Transactions on Knowledge and Data Engineering*, 8(6), 884-897, 1996.
13. K. Koperski and J. Han, Discovery of Spatial Association Rules in Geographic Information Databases, *Advances in Spatial Databases*, Proceeding of 4th Symposium (SSD'95), 47-66, 1995.
14. K. Koperski, J. Han, and J. Adhikary, Mining Knowledge in Geographic Data, to appear in *Communications of ACM.*
15. X. Lin, X. Zhou, C. Liu, and X. Zhou, Efficiently Computing Weighted Proximity Relationships in Spatial Databases, (http://www.cse.unsw.edu.au/~lxue), 2001.
16. H. Lu, J. Han, and B. C. Ooi, Knowledge Discovery in Large Spatial Databases, *Proceedings of Far East Workshop on Geographic Information Systems*, 275-289, 1993.
17. N. Ng and J. Han, Efficient and Effective Clustering Method for Spatial Data Mining, *Proceeding of 1994 VLDB*, 144-155, 1994.
18. J. S. Park, M.-S. Chen, and P. S. Yu, An Effective Hash-Based Algorithm for Mining Association Rules, *Proceedings of 1995 ACM SIGMOD*, 175-186, 1995.
19. F. Preparata and M. Shamos, *Computational Geometry: An Introduction*, Springer-Verlag, New York, 1985.
20. H. Samet, *The Design and Analysis of Spatial Data Structures*, Addison-Wesley, 1990.
21. G. Shaw and D. Wheeler, *Statistical Techniques in Geographical Analysis*, London, David Fulton, 1994.
22. X. Xu, M. Ester, H.-P. Kriegel, Jorg Sander, A Distribution-Based Clustering Algorithm for Mining in Large Spatial Databases, *ICDE'98*, 324-331, 1998.
23. T. Zhang, R. Ramakrishnan and M. Livny, BIRCH: an efficient data clustering method for very large databases, *Proceeding of 1996 ACM-SIGMOD International Conference of Management of Data*, 103-114, 1996.

This article was processed using the LaTeX macro package with LLNCS style