Finding Aggregate Proximity Relationships and Commonalities in Spatial Data Mining

Edwin M. Knorr and Raymond T. Ng

Abstract—In this paper, we study two spatial knowledge discovery problems involving proximity relationships between *clusters* and *features*. The first problem is: Given a cluster of points, how can we efficiently find features (represented as polygons) that are closest to the majority of points in the cluster? We measure proximity in an aggregate sense due to the non-uniform distribution of points in a cluster (e.g., houses on a map), and the different shapes and sizes of features (e.g., natural or man-made geographic features). The second problem is: Given n clusters of points, how can we extract the aggregate proximity commonalities (i.e., features) that apply to most, if not all, of the n clusters? Regarding the first problem, the main contribution of the paper is the development of Algorithm CRH which uses geometric approximations (i.e., circles, rectangles, and convex hulls) to filter and select features. Highly scalable and incremental, Algorithm CRH can examine over 50,000 features and their spatial relationships with a given cluster in approximately one second of CPU time. Regarding the second problem, the key contribution is the development of Algorithm GenCom that makes use of concept generalization to effectively derive many meaningful commonalities that cannot be found otherwise.

Index Terms—spatial knowledge discovery, concept generalization, proximity relationships, geometric filtering, GIS

1 Introduction

In the past few years, many excellent studies on data mining have been conducted [1, 2, 8, 9, 12, 14]. The goal of these studies is to discover knowledge on hidden patterns that may exist in large databases. While most of the studies mentioned above focus on mining relational data, we are particularly interested in mining spatial data. Due to the ever-growing uses of spatial systems such as GIS's, there are already huge amounts of spatial data accumulated, presenting ample opportunities for data analysis and knowledge discovery. (The research described here is an example of such opportunities.) Furthermore, because of events such as the scheduled launch of more satellites in the next decade, it is widely expected that the amount of spatial data collected will soon be so enormous that it becomes unrealistic for human users to examine the data in detail. Spatial data mining, then, aims to automate as much as possible the task of extracting interesting information from the spatial data.

To the best of our knowledge, the first paper that studies spatial data mining is the one by Lu *et al.* [12]. It proposes two algorithms that extract high-level relationships between spatial and non-spatial data stored in a spatial database. However, for those two algorithms to work, spatial hierarchies must be provided as input to the algorithms. For most applications, it is almost impossible to know *a priori* which hierarchies will be the most appropriate. To overcome this problem, Ng and Han [14] develop, among other things, the CLARANS algorithm which can find clustering structures that may exist in the data. Without relying on any spatial hierarchy, CLARANS may discover, for example, that the most expensive housing units in a city can be grouped spatially into a few clusters.

While CLARANS is effective to the extent of answering the question of *what* the clusters are, it fails to answer the question, which is perhaps more interesting from a knowledge discovery point of view, of *why* the clusters are there spatially. Since we believe that in general it may be too much to ask a computer to pinpoint exactly why a cluster is there, we aim to develop algorithms that can answer a weaker form of the question—namely, what the characteristics of the clusters are. In this paper, we focus on characteristics expressed in terms of *features* that are close to the clusters. In abstract terms, a feature can be any simple polygon. Since we use geographic data as our testbed, we define a feature as a closed curve (polygon) describing any natural or man-made place of interest such as a lake, park, school, golf course, shopping centre, etc. Such features can be found in map libraries. For example, the GIS's of the City of Vancouver may have thousands of maps, in a wide variety of scales, containing and describing all kinds of features in Vancouver. Specifically, we aim to provide the following two operations:

1. Finding aggregate proximity relationships —
   For a given cluster of points, find the "top-k" features that are "closest" to the cluster. This problem is not as simple as it may seem, for three reasons: (i) the sizes and shapes of the cluster and the features may vary greatly, (ii) there may be a very large number of features to examine, and (iii) even if a suitable polygon is found to describe the "shape" of the cluster of points, it is inappropriate to simply report those features whose boundaries are closest to the cluster's boundary, because the distribution of points in a cluster may not be uniform. To illustrate the latter point, consider Fig. 1 which shows a cluster of houses (each house denoted by an "x") and a number of features. In terms of boundary to boundary distance, the closest two features to the cluster are the garbage dump and the fire hall; however, the vast majority of the houses are located close to the golf course, lake, mall, and sports complex. Thus, it is more meaningful to find the features that are closest to the *majority* of the houses. This motivates why we want to find the k features which minimize the *aggregate proximity*

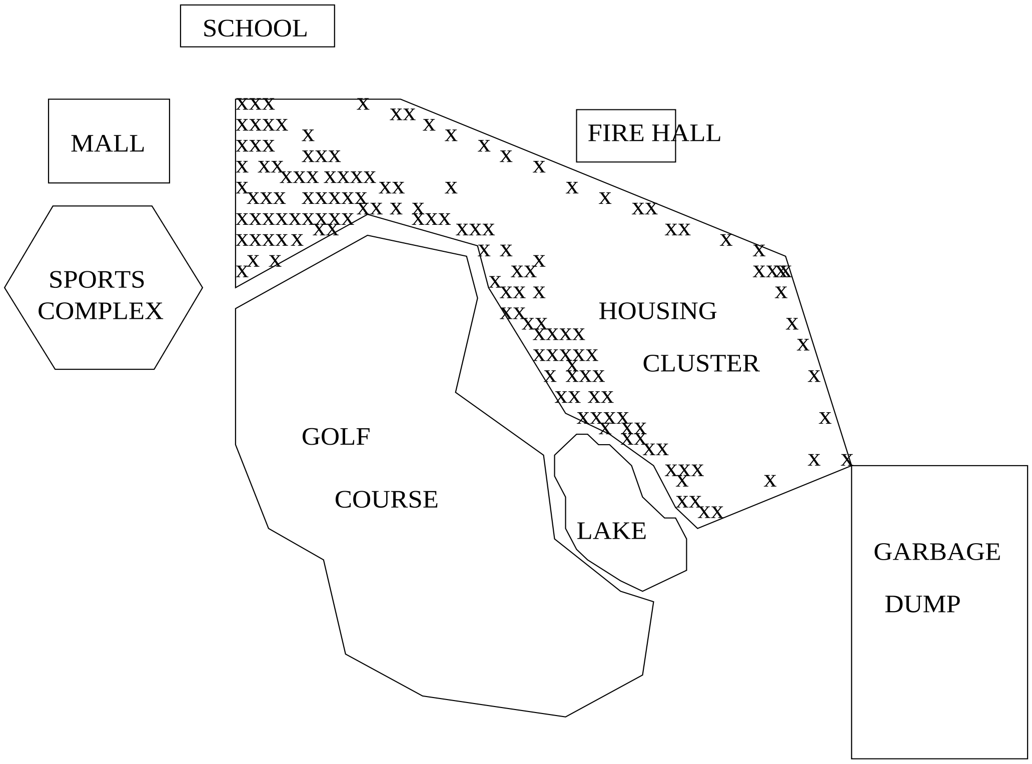

Figure 1: Cluster–feature proximity relationships

value, defined as:

$$ap(Cl, F) \quad = \quad \sum_{pt \in Cl} dist(pt, F) \tag{1}$$

where Cl is a cluster, F is a feature, and $dist(pt, F)$ gives the minimum distance between point $pt \in Cl$ and the boundary of feature F.

2. Finding aggregate proximity commonalities —
   For n given clusters, find common features or classes of features that are nearest to most, if not all, of the clusters. By "nearest", we mean nearest in the aggregate sense, as computed by operation (1) above. By "classes of features", we mean similar types of features that appear in the same taxonomy. For example, in a suitable hierarchy, private grade schools may fall under grade schools, which in turn may fall under educational institutions. Consider a GIS or real estate example in which each of the n given clusters is a cluster of expensive houses. Then this operation of finding commonalities may discover that all clusters are close to some private school (not necessarily the *same* private school), that all clusters are close to some golf course, and so on.

For both operations, the larger the number of features examined, the higher the quality of the results. Thus, scalability is a major design requirement. Since scalability and accu-

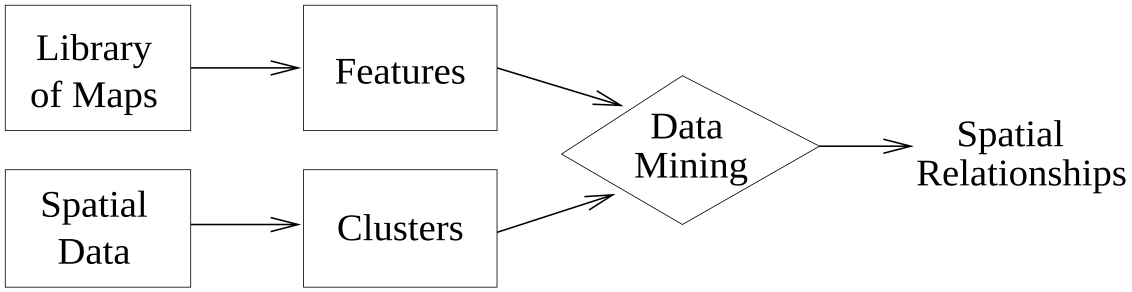

Figure 2: Model of operation

racy often tend to be trade-offs, our goal is to provide a suitable balance between the two. Additionally, we note that most geographic features accessible by computer are stored in free format (i.e., merely represented as a collection of x- and y-coordinates), and do not have accompanying indexes. This needs to be taken into account when we develop algorithms for computing the two operations.

One obvious algorithmic approach is to build an index for the operations. A number of indexing structures come to mind, including k-d trees, quadtrees, Voronoi diagrams, and variants of R-trees [4, 6, 7, 15, 16, 17, 19, 20]. First of all, point indexing structures, such as some forms of k-d trees and quadtrees [19], are too inefficient because computing Equation (1) in a pointwise fashion will be prohibitive. More importantly, constructing and maintaining indexing structures such as region quadtrees, Voronoi diagrams, and R-trees is generally not economical for the data mining task at hand. (See Knorr [10] for a more comprehensive discussion and analysis.) However, one may argue that the amortized cost is not high because an index can be kept around for future operations. But the validity of this argument depends on one factor: the elapsed time between successive operations. If the elapsed time is small, keeping the index around is worthwhile; otherwise, the resources needed to store and maintain the index can be prohibitive.[1] Since most data mining operations are not meant to be performed frequently (e.g., not daily or weekly operations), we do not assume indexes are available unless they are explicitly built. This significantly lowers the attractiveness of index-based techniques. Performance results included in Section 4.2 will confirm this observation with R-trees.

As shown in Fig. 2, our model of operation is that given a cluster, possibly produced by CLARANS, our techniques first load in tens of thousands of features and their attributes, and then carry out the discovery process. Section 4, which presents a case study involving features of Vancouver, will give more information on the kinds of features used and their representations.

The main contributions of this paper are as follows:

[1] For instance, the storage requirement of an order-k Voronoi diagram (k nearest neighbours) is $O(k^2n)$ [3].

- For the first operation of finding aggregate proximity relationships, we develop an algorithm called CRH, which is an acronym for encompassing *C*ircle, isothetic *R*ectangle, and convex *H*ull. Without relying on or building any index structure, CRH uses filters to progressively reduce the number of features in computing the top-k list. Even though CRH is an approximate algorithm, experimental results will show that CRH produces fairly accurate results. By trading off a little bit of accuracy, CRH is able to deliver impressive performance, scalability, and incrementality. For instance, in our case study of features in Vancouver, CRH can examine over 50,000 features and compute the top-25 list in approximately one second of CPU time.

- While the major concern for the first operation of finding aggregate proximity relationships is efficiency, the major concern for the second operation of extracting commonalities is effectiveness. To this end, we develop an algorithm called GenCom that extracts aggregate proximity commonalities from the n top-k lists generated for n input clusters. Whenever necessary, GenCom performs concept generalization to abstract features in the top-k lists. We believe that GenCom is effective since it can derive many meaningful aggregate proximity commonalities.

Section 2 presents CRH and analyzes why circle, rectangle, and hull filters can be put together to deliver performance and accuracy. Section 3 presents GenCom and shows the kinds of commonalities it can extract. Section 4 presents a case study evaluating the efficiency and effectiveness of CRH. Finally, Section 5 concludes with a discussion of ongoing work.

2 Algorithm CRH: Efficient Computation of Aggregate Proximity Relationships

In this section, we introduce Algorithm CRH. To motivate CRH, we first evaluate existing geometric techniques for computing aggregate proximity relationships. The key observation is that these techniques are either accurate or efficient, but not both. This explains why the multiple filtering approach adopted by CRH is appropriate. We also show how CRH controls the number of features passing through filters, computes the ranks of features in various ways, and supports incremental processing.

2.1 Geometric Terminology

The following is some basic geometric terminology referred to throughout this paper. All are 2-D definitions only.

Convex Shape This is a closed boundary of a set of points such that the line segment connecting any 2 points in the set always lies on or interior to the boundary.

Convex Hull This is the unique, minimum bounding convex shape enclosing a set of points. It can be computed in $O(n \log n)$ time, although for a simple polygon, the hull can be found in $O(n)$ time [13], where n is the number of points in the set.

Isothetic Rectangle This is a rectangle that is orthogonal to the coordinate axes. It can be computed in $O(n)$ time by capturing the smallest and largest of the n x- and y-coordinates.

Encompassing Circle This is a circle (not necessarily minimum bounding) that encloses a set of n points. An encompassing circle can be obtained in $O(n)$ time by simply using one-half of the length of a diagonal of the isothetic rectangle for its radius, and by using the intersection point of the diagonals for its centre.

2.2 Evaluation of Geometric Techniques for Computing Aggregate Proximity Relationships

In terms of discovering spatial characteristics that may explain the presence of a cluster, one of the most natural ways is to find those features that are nearby. However, a statement such as "one house in the cluster is close to Park P" is not of high quality. This is because there may be five hundred houses in that cluster, and if only one house is close to P, then it is unlikely that P is of high significance. Statements such as "90% of the houses in the cluster are close to P" are more meaningful and suggestive. This explains why aggregate proximity, as defined in Equation (1), is the desired level of expressiveness. Furthermore, it is informative to quantify proximity explicitly. Rather than describing "proximity" in relative terms, it is desirable to use actual distances, like metres or kilometres.

In Equation (1), aggregate proximity $ap(Cl, F)$ is defined as the sum of all individual distances between a cluster point pt and a feature F (i.e., $dist(pt, F)$). However, the term $dist(pt, F)$ can be ambiguous because there is more than one possible definition. One obvious possibility is to define $dist(pt, F)$ as the distance between pt and some representative point pt_F of F, such as the centroid of F. While abstracting a feature to a single point may be valuable in terms of efficiency (i.e., $dist(pt, pt_F)$ can be computed in $O(1)$ time), this definition is very susceptible to the widely varying shapes and sizes of features. For example, if all houses in a cluster were built to surround a large park, the distance between the centre of the park and any house in the cluster may be so large that the park appears to be insignificant from an aggregate proximity point of view. One may also wonder whether it makes sense to define aggregate proximity $ap(Cl, F)$ as the distance between the boundaries of cluster Cl and feature F, instead of as $\sum_{pt \in Cl} dist(pt, F)$. Similar to the situation above, the sizes and shapes of clusters, and the distribution of points within a cluster may vary widely. Thus, the most expressive definition of aggregate proximity among the possibilities considered, is the one based on $\sum_{pt \in Cl} dist(pt, F)$, where $dist(pt, F)$ is the minimum distance between pt and the boundary of feature F.

The problem of computing aggregate proximity $ap(Cl, F)$ in a pointwise fashion is one of efficiency. To compute the minimum distance between pt and the boundary of feature F with k edges, $O(k)$ time is needed in the worst case. So if there are n points in the cluster, $O(nk)$ time is required to compute $ap(Cl, F)$. Furthermore, if the total number of features is large, which is typically the case, then this time is prohibitive. Thus, it is appealing to consider all the points collectively as a structure (e.g., encompassing circle, isothetic rectangle, or convex hull). For instance, by simply comparing the sum of the radii of two encompassing circles (one for the cluster and one for the feature) with the distance between their centres, the intersection of the two circles can be determined in $O(1)$ time. Testing for the intersection of isothetic rectangles is harder, but the time complexity is still $O(1)$. Finally, testing for the intersection of a k-gon hull and an r-gon hull takes $O(k + r)$ time [17].

Although efficiency is gained, accuracy is compromised by considering all points collectively. Any convex approximation of a non-convex shape will introduce some "white space." [2] An encompassing circle is likely to have a fair bit of white space. An isothetic rectangle is typically more accurate than a circle. Finally, a convex hull is the most accurate convex approximation. Thus, we list the techniques considered here, in ascending order of accuracy: (1) encompassing circles, (2) isothetic rectangles, (3) convex hulls, and (4) pointwise computation of aggregate proximity. Of course, as shown in the previous paragraph, the dilemma is that the very same list is in descending order of efficiency.

2.3 Algorithm CRH: A Multiple Filtering Approach

To decide on a suitable balance between accuracy and efficiency, the following observation is valuable. If we are to try tens of thousands of features on the given cluster, experience tells us that a large number of the features will be so far away from the cluster that the introduction of "white space" using a structure does not in any way affect the outcome of the ensuing operation. In other words, "white space" does not change the accuracy of the operation, but rather aids in improving efficiency.

This argues strongly in favour of a multiple filtering approach, whereby filters are set up in an increasing order of accuracy, but in a decreasing order of efficiency. We develop

[2] We deliberately avoid using the term "error" here because it sounds too negative. As will be shown below, approximation has its virtue.

Algorithm CRH based on a multiple filtering approach. For a given collection of features, it first applies the encompassing circle filter. Features that have the highest rank (described below) are passed on to the next filter: the isothetic rectangle filter. More features are eliminated, and the remaining ones are evaluated using the convex hull filter. Finally, for every feature that remains, CRH calculates the aggregate proximity of points in the cluster to the boundary of the feature. Those features are then ranked based on their aggregate proximity scores.

One may wonder why circles, rectangles, and convex hulls were chosen for our filtering algorithm, in place of other structures such as ellipses and a variety of minimum bounding structures. The key reason is efficiency: it is easy to compute encompassing circles and rectangles for features and clusters, and it is easy to (a) test for intersection, or (b) compute the separation of a pair of circles or rectangles. Although circles tend to have the most white space, they can quickly eliminate a large number of distant features. Rectangles can eliminate far away elongated features (such as a long, thin island) that a circle may not have been able to discard. Convex hulls, which are more expensive to compute, provide the greatest accuracy among convex approximations, and complete the filtering process.

2.4 Ranking Features in CRH

An interesting phenomenon of the ordering of filters in CRH is that after the convex hull filter has been applied, typically very few features intersect the given cluster, owing ironically to the high accuracy of the convex hull. For example, if a housing complex were built across the street from a park, the two convex hulls representing the housing complex and the park may not intersect, even though the circle or rectangle filters may have detected an intersection. This shows that the decision to accept or eliminate a feature cannot be based solely on whether the feature has a non-empty intersection with the cluster or not. Thus, for ranking purposes, we assign the highest score to features that intersect a cluster, and we assign lower scores to disjoint features, depending on how far away a given feature is from a cluster.

This way of ranking features leads to the question of how many features are to be reported to the user after filtering. CRH leaves this decision to the user who can specify the minimum number of features desired (e.g., 10 to yield a "top ten" list). Hereinafter, we denote this threshold as th_h (where "h" stands for "hull"). Thus, CRH returns the th_h features with the highest scores. In case there are features that have the same score as the th_h-th highest ranking feature (i.e., features $F_{th_h - m}, \ldots, F_{th_h}, \ldots, F_{th_h + j}$ all have the same score), CRH does not attempt to distinguish them, and simply reports them all to the user.

Note that the ranks of features are not "monotonic" across filters in the sense that Feature F_1 may be ranked higher than F_2 by one filter (e.g., circle), but the converse is true based on another filter (e.g., rectangle). For example, suppose Feature F_1 is a long, narrow island located relatively far away from the cluster of houses, and F_2 is a small shopping centre that is very close to the cluster. Given the elongated shape of the island, the circle approximating it may intersect the cluster, whereas the circle approximating the shopping centre may not. Thus, the island is ranked higher than the shopping centre. However, if rectangles are used to represent both the island and the shopping centre, the latter is correctly assigned a higher rank. To try to avoid this problem, CRH extends the idea of applying a minimum threshold to every filter: th_c for circles and th_r for rectangles—in addition to th_h mentioned before. For the example above, if the threshold th_c for the circle filter is set high enough, the shopping centre will not be discarded prematurely. Depending on the user's perception of the dataset, the user can specify the values of these thresholds. If the user does not know what values to specify, CRH uses some default values that obey the order of $th_c > th_r > th_h$ (e.g., 30, 20, and 10 respectively).

2.5 CRH Modes and Incremental Support for Changing Thresholds

In this section, we describe three different modes of CRH that compute the ranks of features. We begin our discussion by considering the two modes involving shape enlargement.

To rank features during filtering, CRH begins by testing the cluster with each feature to determine if the two structures intersect. If not enough features are found to equal or exceed the threshold associated with a given filter, the following iterative procedure takes place: enlarge the cluster and then repeat the test for intersection. An encompassing circle can be enlarged by increasing the radius of the circle; an isothetic rectangle can be enlarged by increasing the length of the diagonals of the rectangle; and a convex hull can be enlarged by increasing the distance between its centroid and every vertex, while keeping the centroid and the slope of the line between the centroid and the vertex the same.

As for the amount of enlargement in each iteration, CRH can use two straightforward approaches. In the *linear* mode of CRH, the structure is enlarged by a constant amount or by a constant multiplier in each iteration. For example, if a circle is used, the new radius r_{new} is either $r_{old} + \delta$ or $r_{old} * m$, for some δ and m. In the *bisection* mode of CRH, at most a logarithmic number of iterations is needed. Initially, the range of bisection is defined by the cluster boundary and the largest or smallest x- and y-coordinates of the feature dataset. In other words, the maximum range is the maximum distance between the boundary of the cluster and one of the four corners of the underlying map. In each subsequent iteration, one-half of this distance is discarded, depending on whether too many or too few intersections occurred previously. Bisection continues until either the threshold th is met exactly, or no further bisection is possible, given a tolerance ϵ. The comparison between the efficiency of the linear and the bisection modes depends heavily on the "density" of features. If the density of the neighbouring features is high, the linear mode is likely to be more efficient; otherwise, the bisection mode is likely to be better. However, both modes suffer from the shortcoming that multiple passes through the entire set of features may be needed, which is highly inefficient for large sets of features.

The *memoization* mode of CRH aims to avoid the need for multiple passes by computing and storing the approximate distance between each feature and the cluster during the first pass through the set of features. More specifically, in the circle filter, the minimum distance between the circumferences is calculated and stored. If the shapes are rectangles, then again the minimum distance between the boundaries of the two rectangles is calculated and stored. Finally, if the shapes being considered are convex hulls, then the minimum distance between the boundaries of the two polygons is computed and stored. Even though the computation of the exact distance between two rectangles or two convex hulls may not be of minimal cost, the full impact of the additional cost is minimized in an algorithm like CRH, which adopts a multiple filtering approach. It is important to remember that in Algorithm CRH, while the number of features being examined by the circle filter is the original number of features in the dataset, the number examined by the rectangle filter is only around the level of the threshold associated with the first filter. The same idea applies to the hull filter.

A very useful by-product of the memoization mode of CRH is its incremental support for changing thresholds. As argued in Section 2.4, CRH relies on user-defined thresholds to determine how many features to pass through each filter. Given the exploratory nature of the spatial knowledge discovery task, a user may not be able to immediately decide the appropriate thresholds. The user may change the thresholds a few times depending on the results of the current set of thresholds. This can be accommodated very easily in the memoization mode of CRH, so long as a sufficiently large number of circle distances are memoized. By using the memoized distances, CRH can quickly give the results for the new thresholds, without processing the entire set of features from scratch.

So far we have assumed that the distance between every feature and the cluster is computed. One may wonder whether this can be optimized. Shasha and Wang [21] propose using the triangle inequality to approximate distances; however, the proposed algorithm is of $O(n^3)$ and would not be acceptable for tens of thousands of distances. Thus, we compute every distance directly, and given the performance figures to be presented in Section 4, this simple approach seems to work quite well.

2.6 Bucket Implementation of Memoization

Although the memoization mode of CRH computes every distance directly, for memory space considerations it is not necessary for CRH to store every computed distance. For example, it is sufficient to memoize the smallest M distances (e.g., $M = 1000$), or the distances within a certain radius as determined by the size of the area under examination. In other words, regardless of the actual number of features examined, distance memoization can still be memory-resident.

There is still the question of how to organize the M distances efficiently. One obvious way is to sort the distances. The advantage of doing so is that given any threshold th, exactly th features with the minimum distance can be identified immediately and passed on to the next filter.[3] The problem is that for large values of M, sorting is costly. More importantly, sorting is unnecessary. All that is needed is the identification of the sufficient number of features passing through a filter. The exact ranks of the features are immaterial because, as mentioned before, the ranks of features are not "monotonic" across filters.

In CRH, distances are organized into m buckets. The first bucket contains all features that intersect the cluster. The j-th bucket ($2 \leq j \leq m-1$) contains all features that are of a distance between $w*(j-2)$ and $w*(j-1)$ from the cluster, where w is the width (or distance interval) of each bucket. Finally, the last bucket contains all features that are greater than a distance $w*(m-2)$ away from the cluster. The features contained in a bucket are not sorted according to distances. Let B_j denote the number of features contained in the j-th bucket. Then, for any given threshold th, the smallest integer l is found, where the total number of features contained in buckets up to the l-th one equals or exceeds the threshold (i.e., $B_1 + \ldots + B_l \geq th$). All the features contained in buckets up to the $(l-1)$-st one are passed on to the next filter. Furthermore, the appropriate number of features (i.e., $th - (B_1 + \ldots + B_{l-1})$) contained in the l-th bucket with the smallest distances are identified and passed on as well. Note that even for this bucket, it does *not* have to be sorted. Rather, selection (via partitioning) is performed. This means that regardless of the number of entries in the bucket, performance will be far superior to sorting those same entries.

The degree of success of the above scheme is not critically dependent on the choices of the number of buckets m and the width w. Typically, any "reasonable" values for m and w are sufficient. In particular, knowing the size and the density of the area within which all the features lie goes a long way in helping us pick good values for these two parameters. For instance, for our urban case study to be presented in Section 4, picking any number between 10 metres and 250 metres as the value of w works equally well, so long as a large enough m

[3] The exception, of course, is when there are more than th features that intersect the cluster (i.e., distance $= 0$), in which case all of them will be returned.

...
The cluster is now being compared to Alexander MacKenzie School.
Minimum and Maximum house-to-boundary distance: 48.3 metres and 509.4 metres
47.1% of the houses are within 250 metres of Alexander MacKenzie School.
97.1% of the houses are within 500 metres of Alexander MacKenzie School.
100.0% of the houses are within 1000 metres of Alexander MacKenzie School.
...
Top 10 features ranked in descending order of scores:
John Oliver High School (1000), South Memorial Park (903), Mountain View Cemetery (891), Alexander MacKenzie School (780), MacDonald Park (369), Cartier Park (300), John Henderson School (214), Sunset Park (206), Gray's Park (191), and McBride Annex School (151).

Figure 3: Aggregate proximity results found by Algorithm CRH

is chosen to cover, say, 10 to 20 kilometres. This is a tremendous blessing in the sense that a user typically does not have to be concerned with tuning parameters.

2.7 A Sample Execution of CRH

So far in this section, we have presented Algorithm CRH which uses circle, rectangle, and convex hull filters to efficiently identify features for aggregate proximity computations. We have discussed the linear, bisection, and memoization modes of CRH in finding features that pass through filters. We have also presented an efficient bucket implementation scheme for distance memoization.

To complete the discussion on CRH, Fig. 3 shows part of the output of a sample run of CRH. Fig. 3 shows the aggregate proximity statements generated by CRH for one of the top-10 features: Alexander MacKenzie School. The minimum and maximum cluster point to feature boundary distances are computed, and the percentages of cluster points that fall into various distance ranges of the corresponding feature are shown as well. In this way, the user is given more meaningful and concrete information about the spatial relationship between the cluster and the feature. Furthermore, as shown in the bottom part of Fig. 3, CRH ranks the top-10 features based on a simple heuristic formula, which assigns a higher score to a feature that has a higher percentage of cluster points closer to the feature. Alexander MacKenzie School, for instance, is ranked fourth.

3 Algorithm GenCom: Commonality Extraction by Concept Generalization

In this section, we study how to find aggregate proximity commonalities from the top-k lists generated for n input clusters.[4] We begin by formalizing the notion of commonalities. Then we introduce Algorithm GenCom, and explain how GenCom performs concept generalization to abstract features in the top-k lists. We also show how GenCom can derive many meaningful aggregate proximity commonalities.

3.1 Definition of Common Features

Given n clusters as input, we aim to find characteristics common to most, or all, of the n clusters. In this paper, characteristics are (classes of) features that are in close proximity to the clusters in the aggregate sense. To define the notion of "common features" more formally, there are at least two basic approaches.

The first approach is based on appearance or membership in the n top-k lists generated for the n clusters by Algorithm CRH. The user can specify a threshold m ($m \leq n$), whereby the features appearing in at least m of the lists are designated as "common features". The advantage of this approach is that it captures the spirit of "commonness" in a simple way, and requires no special computation. The disadvantage is that we have no idea of how good the feature is as a common characteristic. We cannot claim that a feature F_1 that appears in 5 of 5 lists necessarily has a "better" relationship to the clusters than a feature F_2 that appears in only 4 of the lists. For example, F_2 may receive a very high rank in each of the 4 lists that it appears in, whereas F_1 may receive a relatively low rank in all 5 lists.

The second basic approach is to simply sum the ranks of the features as they appear in the n top-k lists. Here, we deal with ranks rather than with absolute, distance-dependent scores such as those shown in Fig. 3. Although absolute scores can be informative for a single cluster, they are not very useful when comparing clusters having different sizes. Specifically, for a top-k list, the feature ranked first has "score" k, the feature ranked second has score $k-1$, and so on. In other words, score is based on *inverted rank*. Thus, for any feature appearing in any of the n lists, the best possible score is kn. The advantage of this second basic approach is that it avoids the disadvantage of the first approach, in that features which are ranked higher receive higher scores as common features. However, the disadvantage of the second approach is that no attention is paid to the number of lists that a given feature appears in. For example, a feature F_3 may be ranked first in 2 of the 5 top-10 lists, but may not appear in the other 3 lists. As compared with other common features, F_3's score of 20

[4] The value of k (and its synonym th_h) is chosen by the user.

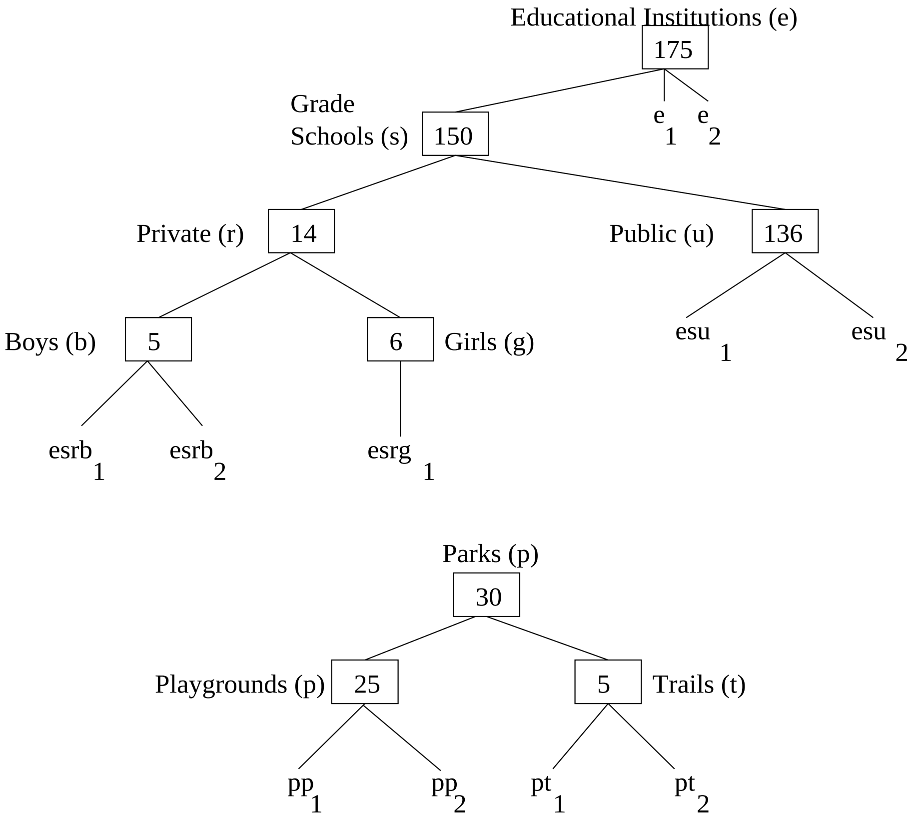

Figure 4: Examples of two partial concept hierarchies

may be quite high. But the fact that F_3 appears in only 2 of the 5 lists begs the question of whether F_3 deserves to be considered a common feature to begin with.

The advantages and disadvantages of these 2 basic approaches suggest a hybrid approach. First, the set of common features is defined to be the set of features $S = \{F_i \mid F_i$ appears in $\geq m$ of the n top-k lists$\}$. This is called the *appearance support* condition. Furthermore, common features are ranked in exactly the same way as described in the preceding paragraph. Thus, in continuing with the examples discussed in the previous paragraphs, F_2 may have a higher rank than F_1, and F_3 is not reported as a common feature if $m > 2$.

While we feel that this is a good way of defining common features from an analytic perspective, it may introduce the following problem in practice. When the value of m is very close to the value of n (which is typical), the set S of common features may be very small, if not outright empty. For example, in a GIS context, when clusters are far apart, it is unlikely that a *specific* school will be close to all the clusters. On the other hand, each cluster may have *some* school that is near it. This motivates us to use concept generalization as a meaningful way to meet the appearance support and to extract more common proximity characteristics.

Inverted Rank	List 1	List 2	List 3
5	$esrb_1$	$esrg_1$	esu_1
4	$esrg_1$	$esrb_2$	pp_2
3	pp_1	esu_2	$esrg_1$
2	pt_2	pt_1	pt_2
1	e_2	e_1	pt_1

Table 1: Initial top-5 lists

3.2 Concepts as Classes of Features

A *concept* represents a class of features. Concepts are organized into concept hierarchies. For simplicity, we assume in this paper that a concept hierarchy is a tree. Fig. 4 shows two such hierarchies: one for educational institutions, and the other for parks. In the educational institutions hierarchy, one subclass of educational institutions (shorthand "e") is grade schools ("s"). Grade schools in turn are classified into private ("r") and public ("u") schools, which can be further sub-classified. Associated with each concept is the number of features belonging to that concept. For instance, there are 175 educational institutions in this example, 150 of which are grade schools. And there are 14 private grade schools, 5 of which are exclusively for boys, 6 exclusively for girls, and 3 exclusively for neither (not shown in Fig. 4). The purpose of these numbers will become apparent in Section 3.3.4.

In the original pool of features, we assume that each feature is assigned a concept that appears in some concept hierarchy. For instance, the feature $esrb_1$ is a boys' private grade school. From now on, to simplify our discussion, we regard a feature also as a concept, which trivially consists of itself. The cardinality of such a concept, which is 1, is not included in the concept hierarchies shown in Fig. 4. Moreover, to simplify the presentation, we only include those features in the concept hierarchies that appear in any of the n top-k lists. For instance, while there are 5 boys' private grade schools in the original pool of features, only $esrb_1$ and $esrb_2$ appear in the lists.

3.3 Algorithm GenCom

Before we present Algorithm GenCom, let us consider an example. Table 1 shows the top-5 lists for $n = 3$ clusters. Suppose for the appearance support requirement, m is set to n. Then, the girls' private school $esrg_1$ is the only feature that is considered a common feature (or concept); therefore, at this point, the answer set is $S = \{esrg_1\}$. Without using concept hierarchies, nothing else can be extracted from the lists.

To allow more common characteristics to be extracted, the idea is to weaken the concepts,

Inverted Rank	List 1	List 2	List 3
5	$esrb$	$esrg$	esu
4	$esrg$	$esrb$	pp
3	pp	esu	$esrg$
2	pt	pt	pt
1	e	e	pt

Table 2: Lists after the first generalization step

and therefore meet the appearance support, by ascending the concept hierarchies, typically one step at a time. That is to say, a concept is replaced by its parent concept in the appropriate concept hierarchy. Table 2 shows the results of the lists shown in Table 1 after generalizing each concept by one step. For simplicity, a concept is represented by concatenating all the shorthand notations of its ancestor concepts. For instance, $esrb$ represents the concept of boys' ("b") private ("r") grade school ("s") educational institutions ("e"). Under this convention, concept generalization corresponds to dropping the last character from the concept. For instance, entries in Table 2 are obtained by dropping all the subscripts from the entries in Table 1.

3.3.1 Superseded Concepts

Notice from Table 2 that the concept $esrg$ satisfies the appearance support. However, this is a direct consequence of the fact that $esrg_1$ (cf: Table 1) satisfies the appearance support in the first place. Thus, so long as $esrg_1$ is considered a common concept, there is no need to regard $esrg$ as another common concept because no new information is captured. The situation would be different if there were other features that contribute to $esrg$ meeting the appearance support. The following definitions distinguish between the two situations.

Each concept C is associated with a *feature list*, L_C, that consists of all the features that are in the concept (sub-)hierarchy rooted at C, and that appear in at least one of the n top-k lists. For instance, based on the concept hierarchies shown in Fig. 4, L_{esrg_1} is the list $\{esrg_1\}$, L_{esrg} is also the list $\{esrg_1\}$, and L_{esr} is the list $\{esrb_1, esrb_2, esrg_1\}$. We say that a concept C is *superseded* in the answer set S if there exist concepts $D_1, \ldots, D_u \in S$ such that:

- for all $1 \leq i \leq u$, D_i is a descendent of C in the appropriate concept hierarchy, and
- the feature list of C is exactly the same as all the feature lists of $D_1, \ldots, D_u$ combined (i.e., $L_C = \cup_{i=1}^{u} L_{D_i}$).

For instance, since $esrg_1$ is a descendent of $esrg$ and L_{esrg_1} is the same as L_{esrg}, the concept $esrg$ is superseded by $esrg_1$. On the other hand, even though $esrg_1$ is a descendent of esr, the concept esr is not superseded by $esrg_1$ because L_{esr} is different than L_{esrg_1}. In our full algorithm for extracting common concepts, superseded concepts are thrown away. But before we can present our full algorithm, we need one more notion.

3.3.2 Deferrable Concepts

Consider the situation shown in Table 2 again. Notice that the concept e is actually an ancestor of the concepts $esrg, esrb$, and esu. There are two reasons why ancestors and descendents can co-exist in a generalized list such as the one shown in Table 2. First, different features may be assigned to concepts at different levels of a hierarchy. For instance, as shown in Fig. 4, feature e_1 is assigned to the concept e (i.e., educational institutions), whereas features such as $esrg_1$ are assigned to lower level concepts like $esrg$. Second, different branches of a concept hierarchy may have different lengths. For instance, in the educational institutions hierarchy, private schools have more sub-divisions than public schools have. Thus, esu_1 becomes es in two steps of generalization, at which time $esrg_1$ is replaced only by esr, a descendent of es.

To deal with cases like these, there are two basic options. First, in the same generalization step, we immediately generalize the lower level concepts to the highest one, like generalizing esr directly to es. But this is not good because if esr has already satisfied the appearance support, generalizing to es unnecessarily weakens the result. Thus, the second option, which is the option we adopt, does the opposite—namely, in the *subsequent* generalization step(s), higher level concepts are not generalized until lower level concepts have caught up. We say that a concept C is *deferrable* if there exists a descendent D of C that has not yet been generalized to C. For instance, in Table 2, the concept e is deferrable because of concepts $esrg, esrb$, and esu. We are now in a position to present the full algorithm for extracting common concepts.

3.3.3 The Complete Algorithm

Fig. 5 presents the full algorithm for extracting common concepts with generalization. The answer set S contains all common concepts that are to be reported to the user. In the first iteration, only *features* can be added to S in Step 3.2.1. The algorithm eventually stops if a sufficient number, denoted by min, of concepts/features have been identified. The parameter min is an input parameter set by the user. If the number of concepts found so far is not enough, generalization takes place in Step 5 for all non-deferrable concepts. If in this step, some concept has been generalized, then the algorithm returns to Step 2 to start

```
1       Initialize answer set S to the empty set
2       Check the appearance support
3       For each concept C meeting the appearance support
3.1         Check if C is superseded in S
3.2         If not superseded and not deferrable
3.2.1           Add C to S
4       If |S| >= min
4.1         Compute and report normalized summation support
4.2         Return
5       Generalize all concepts that are not deferrable at this point
6       If nothing further can be generalized in step 5
6.1         Add to S all root concepts meeting the appearance support
6.2         Compute and report normalized summation support
6.3         Return
7       Goto 2
```

Figure 5: Algorithm GenCom for extracting common concepts with generalization

a new iteration. Otherwise, in Step 6, the algorithm has to eventually stop, even if fewer than min concepts have been found.

Let us apply the above algorithm to the example shown in Table 1 with min set to 5. Since $esrg_1$ meets the appearance support, it is added to S. However, since more concepts need to be found, generalization takes place and Table 2 is obtained. In this iteration, concepts $esrg$ and pt meet the appearance support. However, $esrg$ is superseded in S which now contains $esrg_1$, and is not added to S. On the other hand, concept pt is added to S. Still, the condition $|S| \geq min$ is false. Thus, another round of generalization is needed, but with concept e deferrable.

Table 3 shows the lists after two steps of generalization. This time, concepts esr and p appear in all 3 lists. Although $esrg_1$ has been added to S, esr is not superseded by $esrg_1$; consequently, esr is added to S. Concept p is also added to S, because p is not superseded by pt due to pp_1 and pp_2. This brings the cardinality of S to 4, which means that further generalization is still needed. This time, concepts e and es are deferrable.

Table 4 shows the lists after three steps of generalization. Here, concept es appears in all 3 lists. Since es is not superseded, it is added to S. Since the condition $|S| \geq min$ becomes true, the algorithm no longer needs to go into another iteration. It computes the normalized summation support values (to be described in the next section), and exits.

The processing time required by Algorithm GenCom in Fig. 5 is trivial compared to the processing time required by Algorithm CRH. CRH is invoked n times (for n clusters), and examines a potentially large pool of features. GenCom, on the other hand, deals with n relatively small lists each containing k entries (i.e., the output from CRH), and all of these nk

Inverted Rank	List 1	List 2	List 3
5	esr	esr	es
4	esr	esr	p
3	p	es	esr
2	p	p	p
1	e	e	p

Table 3: Lists after the second generalization step

Inverted Rank	List 1	List 2	List 3
5	es	es	es
4	es	es	–
3	–	es	es
2	–	–	–
1	e	e	–

Table 4: Lists after the third generalization step

entries are processed in main memory. Thus, the efficiency of the entire commonality extraction process is the efficiency of Algorithm CRH. We examine the efficiency and scalability of CRH in Section 4.

3.3.4 Computing the Normalized Summation Support

Two issues arise in computing the "final scores" for the concepts in the answer set $S = \{esrg_1, pt, esr, p, es\}$. The first issue concerns concepts that appear multiple times in the same list, such as pt in the third list in Table 2. Two occurrences of pt exist because pt_1 and pt_2 in Table 1 have both been generalized to the same concept. We need to count all occurrences of pt because the more times pt appears in the top-k lists, the stronger the support is for pt. Thus, we compute the sum of inverted ranks for pt: 2+2+2+1=7. This value is called the *summation support* of pt.

For concepts $A, B \in S$, where A is an ancestor of B, the summation support for A will be larger than that for B. Furthermore, if there are more features belonging to concept $C \in S$ than to concept $D \in S$, the summation support for C is *likely* to be larger than that for D. However, consider the following example, which describes the second issue we need to consider before presenting our final results. If there are only 2 cemeteries but 175 educational institutions in the original pool of features, then the appearance of the concept "cemetery" in the answer set S is far more significant than the appearance of educational institutions in

	$esrg_1$	pt	esr	p	es
Summation Support	12	7	21	14	29
Cardinality	1	5	14	30	150
Normalized Summation Support	12	1.4	1.5	0.47	0.19

Table 5: Ranking the concepts in the answer set by normalized summation support

S. This leads us to normalize the summation support by dividing the summation support by the number of features belonging to the concept. Table 5 shows these values for the concepts in S. The normalized summation support for $esrg_1$ is deservingly higher than that for es, even though the summation support values are 12 and 29 respectively. Also, note that the normalized summation support for concept esr, which was added to S after two generalization steps, exceeds that of pt, which was added to S after one generalization step. This shows that concepts found in later iterations are not necessarily ranked lower than concepts found earlier. Based on the results shown in Table 5, GenCom outputs such statements as "all 3 clusters are close to school $esrg_1$," "all 3 clusters are close to some private grade school," "all 3 clusters are close to some trail park," etc. In the spirit of providing explicit distance information, GenCom can provide further statistics based on the aggregate proximity results calculated for the original top-k lists.

The attribute-oriented approach described by Han *et al.* [8] and Lu *et al.* [12] is an excellent example of the use of concept generalization in a data mining context. In that approach, concept generalization is performed to reduce the number of tuples in a generalized relation. In contrast, in Algorithm GenCom, generalization serves the purpose of producing additional concepts that can satisfy the appearance support. Furthermore, GenCom may generalize more than one concept simultaneously in each iteration, whereas in the attribute-oriented approach, only one attribute is chosen for generalization. There is the notion of deferrable concepts in our algorithm that does not have any counterpart in the attribute-oriented approach. Other notions such as superseded concepts and normalized summation support are also unique in our algorithm.

4 Experimental Evaluation of CRH

This section is based on a case study of features in Vancouver. We give experimental results comparing the efficiency of different modes of CRH with the efficiency of the brute-force and R-tree approaches. We also present results showing the scalability and incrementality of Algorithm CRH. Finally, we show the accuracy of the results computed by CRH. All experiments were carried out on a time-sharing, single processor Sun SPARC-10 workstation.

We begin by giving a detailed account of the dataset of our case study and implementation.

4.1 Details of Case Study and Implementation

Our case study involved houses and features in the City of Vancouver. Using maps from the City's Engineering Department, vertices of features were digitized to an accuracy of a few metres. Some of those features are rectangular such as schools and shopping centres; others are highly irregular such as golf courses, parks, and beaches.

Since one of our objectives was to test scalability with tens of thousands of features, we decided to save an enormous amount of effort by duplicating the pool of features (of which there are 326) that were produced in the time-consuming digitization process. However, care had to be taken that the duplication did not change the mixture of the highest ranked features; otherwise, the same feature would be selected numerous times in a top-k list. In particular, we ran our first test against the 326-feature dataset, and then temporarily removed the top 75 ranked features, leaving behind 251 features. These 251 features were then duplicated as many times as needed to provide us with a huge set of features (concatenated with the original 75 features), all of which take part in CRH filtering.

For memoization, we used 100 buckets, each of width 100 metres. We found that it did not matter very much whether 1000, 500, or even 50 buckets were used. Only a poor choice, say 10 buckets at 5000 metre intervals, would noticeably affect the CPU time. As mentioned in Section 2.6, the tremendous leeway in specifying memoization parameters minimizes tuning efforts.

For the R-tree implementation, we used the R-tree code kindly provided and developed by Christos Faloutsos and his group at the University of Maryland, College Park. This package supports both the original R-tree mode and the mode that implements deferred splitting. To simplify the presentation, we only show the results of the better of the two modes, whichever that may be. One may argue that the R-tree code we used may not be as optimized as other implementations such as one for R*-trees, but as shown below, since CRH performs significantly better than our R-tree implementation, it is doubtful whether our conclusions need to be changed even if a more optimized implementation is used.

4.2 Efficiency of CRH Relative to Other Approaches

In this set of experiments, we compared the three modes of CRH with the R-tree approach and two brute force approaches. Whereas CRH uses filtering to yield the k "nearest" features to a cluster (nearest in an approximate sense), the R-tree approach uses a nearest neighbour search to find the k nearest (feature) rectangles to the cluster's rectangle. The "Brute Force (approx.)" approach finds the k *nearest neighbours* using individual cluster vertex to feature

Number of Features	326	2585	25175	50275
CRH Memoization Mode	0.34	0.36	0.65	1.11
CRH Bisection Mode	0.34	0.47	1.94	3.62
CRH Linear Mode	0.67	1.00	4.96	9.42
Brute Force (approx.)	0.88	1.80	13.63	26.85
R-tree	0.39	3.38	59.40	128.61
Brute Force (actual)	6.40	58.19	906.33	>2147

Table 6: CPU times (in seconds) for various approaches

edge computations (and cluster edge to feature vertex computations). Note that this type of brute force approach is still an approximation because the distribution of points within a cluster is not considered. In order to guarantee accuracy, it is necessary to examine *all* of the features rather than just the nearest neighbours. The "Brute Force (actual)" approach, on the other hand, computes the aggregate proximity of every feature directly based on Equation (1). In this series of experiments, the task is the generation of a top-25 list for a 7 hectare cluster of 71 houses in south-central Vancouver. We applied each of the 6 approaches to varying numbers of features—up to 50,275. The thresholds associated with the circle, rectangle, and convex hull filters were set to 75, 50, and 25 respectively.

Each run-time entry in Table 6 includes the total CPU time taken to carry out all operations associated with the approaches, except loading. Specifically, in the case of CRH, the time includes the filtering and the aggregate proximity calculations. In the case of the R-tree approach, the time includes the time to construct the R-trees, but excludes the time to load the raw data and to generate each rectangle's coordinates (this was preprocessed). It is clear from Table 6 that all three modes of CRH (except for the linear case involving 326 features) are more efficient than the other approaches. Also, among the different modes of CRH, memoization mode is the most efficient. It can process over 50,000 features in slightly more than one second of CPU time.

4.3 Scalability of Different Modes of CRH

Fig. 6 shows the scalability of the three modes of CRH. The x-axis of the graph represents the number of features, and the y-axis gives the CPU time in seconds. Fig. 6 indicates that all modes of CRH scale up linearly with the number of features. The key observation is that as far as run-time is concerned, the circle filter in Algorithm CRH is the dominant filter. This can be confirmed by the following table which breaks down the time taken by CRH (with memoization) to process 50,275 features into the CPU time spent on each of the three

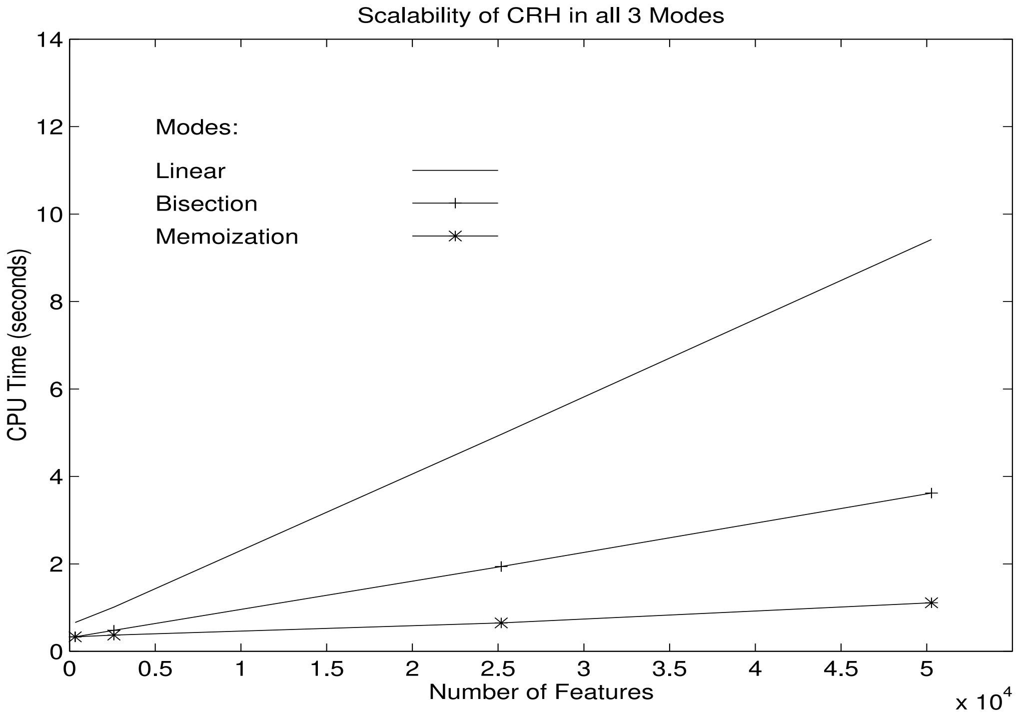

Figure 6: Scalability of Algorithm CRH in compiling a top-25 list

filters.

Operations	Circle	Rectangle	Hull	Aggregate Calc.	Total
CPU time (sec.)	0.76	0.00	0.22	0.13	1.11

Our tests indicate that if only the convex hull filter were used to process the same 50,275 features, it would take at least 20 seconds of CPU time. The above table also suggests that if we are to include more/other filters that are more expensive to compute, the run time would still be well under control—so long as the number of features examined by those filters is not large.

As shown in Fig. 6, the memoization mode outperforms the bisection mode, which in turn outperforms the linear mode. One of the key factors in determining the efficiency of a particular mode of CRH is the number of passes through the feature pool that is required. The following table gives the number of passes for each of the three modes in this experiment.

Modes	Linear	Bisection	Memoization
Passes	19	5	1

There are situations in which the linear and bisection modes can perform better than the memoization mode. These are situations where the density of features is very high, and the number of passes required by the two approaches is very small (e.g., 1-3). In general, however, memoization is the route to follow since it guarantees only one pass through the

Number of Features	326	2585	25175	50275
Loading Prior to Invoking CRH	0.47	3.84	37.85	77.99
Finding 25 Nearest Neighbours (CRH)	0.21	0.23	0.52	0.98
Loading Prior to Building R-tree	0.42	3.52	33.27	67.21
Building R-tree	0.23	3.18	59.15	128.48
Finding 25 Nearest Neighbours (R-tree)	0.03	0.07	0.12	0.12

Table 7: Comparison of CPU times to perform loading and filtering

entire set of features and does not depend on the distribution of features. The "one pass" guarantee is particularly important if the feature set is quite large, simply because of the amount of I/O or paging activity that may be needed.

4.4 Incremental Support for Changing Thresholds and Re-runs

In the following set of experiments, we changed the thresholds up and down to evaluate how well memoization can support incrementality. In particular, we changed the set of thresholds 30 (circle), 20 (rectangle), and 10 (hull) for a cluster to the following two sets: (a) 15, 10, and 5; and (b) 75, 50, and 25. Again, 50,275 features were used.

For case (a), negligible time was taken because all results had been previously computed and stored. But even for case (b), when the thresholds were raised, only an additional 0.10 seconds were needed to meet the new thresholds. If memoization were not supported, re-running the entire dataset of 50,275 features from scratch (including loading) would be required. To a large extent, this kind of incremental efficiency is expected because the total time for the rectangle filter, the convex hull filter, and the computation of aggregate proximity values is relatively small.

One may argue that if an index were built, incremental processing can also be supported readily. So in order to provide a fair comparison between CRH and the approach based on spatial indexes (i.e., R-trees in this case), we provide Table 7. The last row of the table indicates that for 50,275 features, it takes 0.12 seconds to perform an R-tree search for 25 nearest neighbours. This is more than the 0.10 seconds reported in the previous paragraph for the memoization mode of CRH. Thus, in terms of supporting changing thresholds, the memoization mode of CRH is still more efficient.

There is, however, another form of re-use supported by the spatial index approach. That is, the spatial index can be kept around for future computations of aggregate proximity. In this case, the 128.48 seconds used to build the R-tree for 50,275 features can be amortized since the build phase only needs to be performed once. The "Loading Prior to Building R-tree" row in Table 7 shows that an additional 67.21 seconds should be added. Thus, loading and tree construction requires 195.69 seconds in the R-tree case. Then, if the tree were kept around, each subsequent nearest neighbour operation could be computed in 0.12 seconds.[5]

In contrast, a complete CRH run requires re-loading the raw data. Thus, each complete re-run of 50,275 features would take about 78.97 seconds. This implies that the R-tree approach will have the edge over CRH *if* the same set of features is expected to be re-used at least 3 times, and *if* the cost of storing the data is justified. But there are two reasons why this argument may not be applicable. First, from one run of CRH to another run, different feature sets may be used depending on the nature of the clusters. Second, as argued earlier, the elapsed time between successive runs may be large. For instance, the kinds of data mining operations described here may not be required every week or every month. In that case, the storage cost and the effort required to maintain the index may not be justified.

4.5 Accuracy of Results Produced by CRH

Recall that by using filtering, CRH is an approximate algorithm in that the top-k features found by CRH may not be the same as those found by the "Brute Force (actual)" approach described in Section 4.2. In this set of experiments, we evaluate the accuracy of the results. In particular, we carried out numerous experiments involving different values of k, different sizes of feature sets, and different clusters having various spatial characteristics (e.g., small or large, dense or sparse, near many or few features, near regularly-shaped or irregularly-shaped features). For similarity retrieval, *precision* and *recall* are two standard measures of accuracy [18]. We use a variation of these measures below.

Table 8 shows the results when using different values of k for the desired threshold (i.e., top-k list) for four clusters. The thresholds used were $th_c = 3k$, $th_r = 2k$, and $th_h = k$. The left hand column of the table indicates the value of k selected, while the next four columns show the value k' that CRH *would have needed* in order to return the top-k features in their *actual* order. We determined these actual or absolute values by the "Brute Force (actual)" approach that computes and ranks the aggregate proximity results of all features.

While Table 8 indicates the value of k' needed before the absolute top-k results were returned, it does not indicate the quality of the results returned up to that point (i.e., at k). Thus, another way to measure the accuracy of CRH is to use the sum of the inverted ranks. Table 9 compares the sum of CRH ranks for a given value of k to the sum of *ideal* ranks: $k(k+1)/2$. Let us first explain the concept of "sum of ideal ranks". Suppose that in a top-5 list, the features ranked 1, 2, 3, 4, and 5 by CRH are actually ranked as 1, 2, 3, 8, and 4 by

[5] For this particular example, another 0.13 seconds would be required to compute the aggregate distance statistics, regardless of whether the R-tree approach or the CRH approach is taken.

Desired Threshold k	k' needed for Cluster A	k' needed for Cluster B	k' needed for Cluster B'	k' needed for Cluster C
5	5	18	5	13
10	10	18	10	19
15	15	31	15	19
25	25	31	25	26
50	50	60	50	60

Table 8: Values of k' that would have been needed by CRH to correctly determine the top-k rankings for four clusters

	k=5	k=10	k=15	k=25	k=50
Ideal Sum and Sum for A and B'	15	55	120	325	1275
Sum for B	42	86	163	398	1307
Sum for C	22	87	127	330	1286

Table 9: Comparison of sum of ideal ranks (brute force) to sum of CRH ranks

the brute force (actual) approach. Then, the sum of CRH ranks would be 18, and the sum of ideal ranks would be 15. The closer the sum of CRH ranks is to the sum of ideal ranks, the better the results returned by CRH.

As shown in both tables, CRH is perfect for clusters A and B' for all values of k. CRH also gives very good results for clusters B and C for larger values of k. However, CRH is not as accurate for B and C for small values of k, like 5 or 10. A closer look at the characteristics of the clusters reveals the following observation. Cluster A is a small cluster, whereas clusters B and C are much larger in size. In fact, the sizes of the latter two clusters are so large that there are features F_i that are located *within* the clusters. Whenever that happens, intersection operations using the circle, rectangle, and hull filters undoubtedly give a high rank for F_i. But in calculating the actual aggregate proximity based on Equation (1), F_i may be quite far away from the majority of the points in the cluster, and thus not deserving such a high rank. Hence, when using CRH with large-sized clusters, there may be features like F_i that get into the top-5 or top-10 lists undeservingly. Actually, for large-sized clusters where there are numerous features contained within them, setting k too small is asking too much from CRH. To verify the above observation, we reduced cluster B to B'. As predicted, CRH is very accurate for B'.

5 Summary and Ongoing Work

In this paper, we have studied the problem of finding aggregate proximity relationships efficiently. Using a brute force approach, the computation of aggregate proximity relationships involving a large number of features may be too costly. To this end, we have developed Algorithm CRH, which uses circle, rectangle, and convex hull filters to reduce the number of candidates (i.e., features). We have developed three different modes of operation for CRH. Among them, the memoization approach gives a "one pass" guarantee and provides efficient incremental support for changing thresholds. Our experimental results clearly show that all three modes of CRH are very efficient in comparison with other approaches, including one that uses R-trees. The memoization mode, in particular, can examine spatial relationships between a cluster and 50,275 features, and can compute the aggregate proximity statistics for the most promising features, in approximately one second of CPU time. Though approximate in nature, CRH is capable of delivering very accurate aggregate proximity results for small clusters, and reasonable results for large clusters, particularly for higher numbers of features to be returned. Whatever accuracy is sacrificed by CRH, CRH is able to gain far more back in efficiency and scalability.

Now that we have identified an efficient way of determining proximity relationships involving a large number of features, we can apply CRH to multiple clusters. This brings us to the second problem solved in this paper: commonality extraction by identifying common features among multiple clusters. We have presented Algorithm GenCom that uses concept generalization to extract more information from the top-k lists of clusters. GenCom deals with superseded and deferrable concepts, and computes the final rankings of concepts, appropriately adjusted by the number of features belonging to the concepts. We believe that GenCom is effective because it is able to derive many meaningful commonalities that cannot be found otherwise. GenCom is a very efficient algorithm because all of its calculations are performed on small lists stored in main memory. These lists were obtained from CRH, which is where most of the processing takes place.

While we believe that the research reported in this paper has built a foundation for efficient and effective discovery of certain kinds of spatial relationships and knowledge, in ongoing work we attempt to build on this foundation by developing techniques that can discover other kinds of spatial relationships and knowledge. For instance, techniques that can handle abstract features, represented in a way similar to contour lines and isobars, may lead to correlations between a cluster and many kinds of demographic data. Examples of this kind of discovered knowledge may be that all expensive housing clusters in Vancouver are located in areas "that have an annual homicide rate of no more than 2," or "that have the lowest unemployment rate in the city." Moreover, incorporating a filter into CRH using *alpha-shapes* [5] may help to discover features whose boundary shapes (or portions thereof) closely approximate that of a cluster. An example of this kind of discovered knowledge may be that the shape of a certain expensive housing cluster approximates the shape of the coastline it faces. Finally, in a forthcoming paper [11] and in ongoing work, we specifically study the issue of how to find discriminating properties of clusters—properties that serve to distinguish one group of clusters from another.

References

[1] R. Agrawal, S. Ghosh, T. Imielinski, B. Iyer, and A. Swami. "An Interval Classifier for Database Mining Applications", *Proc. 18th VLDB*, pp. 560–573, 1992.

[2] R. Agrawal, T. Imielinski, and A. Swami. "Mining Association Rules between Sets of Items in Large Databases", *Proc. SIGMOD*, pp. 207–216, 1993.

[3] F. Aurenhammer. "Voronoi Diagrams—A Survey of a Fundamental Geometric Data Structure", *ACM Computing Surveys*, Vol. 23, No. 3, pp. 345–405, September, 1991.

[4] N. Beckmann, H.-P. Kriegel, R. Schneider, and B. Seeger. "The R*-Tree: an Efficient and Robust Access Method for Points and Rectangles", *Proc. SIGMOD*, pp. 322–331, 1990.

[5] H. Edelsbrunner, D.G. Kirkpatrick, and R. Seidel. "On the Shape of a Set of Points in the Plane", *IEEE Transactions on Information Theory*, 29, 4, pp. 551–559, July, 1983.

[6] C. Faloutsos, T. Sellis, and N. Rossopoulos. "Analysis of Object-Oriented Spatial Access Methods", *Proc. SIGMOD*, pp. 426–439, 1987.

[7] R. Guttmann. "A Dynamic Index Structure for Spatial Searching", *Proc. SIGMOD*, pp. 47–57, 1984.

[8] J. Han, Y. Cai, and N. Cercone. "Knowledge Discovery in Databases: an Attribute-Oriented Approach", *Proc. 18th VLDB*, pp. 547–559, 1992.

[9] D. Keim, H.-P. Kriegel, and T. Seidl. "Supporting Data Mining of Large Databases by Visual Feedback Queries", *Proc. 10th Data Engineering*, pp. 302–313, 1994.

[10] E.M. Knorr. *Efficiently Determining Aggregate Proximity Relationships in Spatial Data Mining*, MSc Thesis, Dept. of Computer Science, University of British Columbia, 1995.

[11] E.M. Knorr and R.T. Ng. "Extraction of Spatial Proximity Patterns by Concept Generalization", *Proc. 2nd KDD*, pp. 347-350, August, 1996.

[12] W. Lu, J. Han, and B. C. Ooi. "Discovery of General Knowledge in Large Spatial Databases", *Proc. Far East Workshop on Geographic Information Systems*, Singapore, pp. 275–289, 1993.

[13] A. Melkman. "On-Line Construction of the Convex Hull of a Simple Polyline", *Information Processing Letters*, 25, pp. 11–12, 1987.

[14] R. Ng and J. Han. "Efficient and Effective Clustering Methods for Spatial Data Mining", *Proc. 20th VLDB*, pp. 144–155, 1994.

[15] A. Okabe, B. Boots, and K. Sugihara. "Nearest Neighbourhood Operations with Generalized Voronoi Diagrams: A Review", University of Tokyo, Dept. of Urban Engineering, Discussion Paper Series, No. 51, September, 1992.

[16] J. O'Rourke. *Computational Geometry in C*, Cambridge University Press, New York, 1994.

[17] F.P. Preparata and M.I. Shamos. *Computational Geometry*, Springer-Verlag, New York, 1985.

[18] G. Salton and M. McGill. *Introduction to Modern Information Retrieval*, McGraw-Hill, 1983.

[19] H. Samet. *The Design and Analysis of Spatial Data Structures*, Addison-Wesley, 1990.

[20] T. Sellis, N. Rossopoulos, and C. Faloutsos. "The R^+-Tree: A Dynamic Index for Multi-dimensional Objects", *Proc. VLDB*, pp. 507–518, 1987.

[21] D. Shasha and T.-L. Wang. "New Techniques for Best-match Retrieval", *ACM Transactions on Information Systems*, 8, 2, pp. 140–158, 1990.

Affiliation of Author

E. Knorr and R. Ng are with the Department of Computer Science, University of British Columbia, Vancouver, B.C., Canada V6T 1Z4. E-mail: {knorr,rng}@cs.ubc.ca.

Acknowledgments

This research was partially sponsored by NSERC Grants OGP0138055 and STR0134419, IRIS-2 Grants HMI-5 and IC-5, and a CITR Grant on "Distributed Continuous-Media File Systems."

Biographies

Ed Knorr received the BMath (Co-op) degree in Mathematics from the University of Waterloo in 1983, and the MSc degree in Computer Science from the University of British Columbia in 1995. He is currently pursuing his PhD in Computer Science at the University of British Columbia. Mr. Knorr has over 10 years' experience in large corporate mainframe environments, largely in database systems. His research interests include data mining, smart cards, data security, and privacy.

Raymond Ng received the BSc (Hons) degree in Computer Science from the University of British Columbia in 1984, the MMath degree in Computer Science from the University of Waterloo in 1986, and the PhD degree in Computer Science from the University of Maryland, College Park, in 1992. Since then, he has been an assistant professor at the University of British Columbia. His areas of research include data mining, image databases, and multimedia systems.

List of Index Terms

Index Terms—spatial knowledge discovery, concept generalization, proximity relationships, geometric filtering, GIS