Performing Boundary Shape Matching in Spatial Data

Edwin M. Knorr, Raymond T. Ng, David L. Shilvock
Department of Computer Science
University of British Columbia
Vancouver, B.C., V6T 1Z4, Canada
{knorr,rng}@cs.ubc.ca

Abstract

This paper describes a new approach to knowledge discovery among spatial objects—namely that of partial boundary shape matching. Our focus is on mining spatial data, whereby many objects called *features* (represented as polygons) are compared with one or more point sets called *clusters*. The research described has practical application in such domains as Geographic Information Systems, in which a cluster of points (possibly created by an SQL query) is compared to many natural or man-made features to detect partial or total matches of the facing boundaries of the cluster and feature. We begin by using an alpha-shape to characterize the shape of an arbitrary cluster of points, thus producing a set of edges denoting the cluster's boundary. We then provide an approach for detecting a boundary shape match between the facing curves of the cluster and feature, and show how to quantify the value of the match. Optimizations and experimental results are also provided. Finally, we describe several orientation strategies yielding significant performance enhancements.

Keywords: spatial knowledge discovery, pattern matching, GIS

1 Introduction

Knowledge discovery is defined as the nontrivial extraction of implicit, previously unknown, and potentially useful information from data [5]. The purpose of spatial data mining is to discover relationships or patterns in such spatial data as maps, photographs, and satellite images. Although many researchers have investigated relationships that exist among *relational* data [1, 2, 6, 7], a few others have gone beyond the relational model in order to discover relationships in spatial domains [10, 11]. In this paper, we aim to show how to discover spatial relationships that involve boundary shape matching (BSM).

First, we define BSM and explain why it is of interest to the data mining community. Figure 1 shows a cluster of points (x's representing houses) with which we can determine the natural or man-made features (described by simple polygons) that have portions of their boundaries closely matching those of the cluster. In this paper, we refer to the point set as a *cluster* and to the geographic entities in a set of maps as *features*. Figure 1 indicates that part of the cluster's boundary has a reasonably good match with part of the golf course's boundary. Consequently, we would consider the golf course to be a feature of interest for this cluster. The knowledge discovered is the identification of the feature as well as the parts of the feature's boundary that match the cluster.

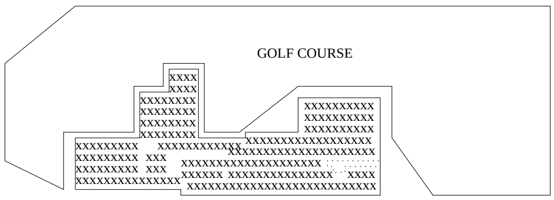

Figure 1: Housing Cluster Bordering a Golf Course. Each house is denoted by an x.

BSM is useful because it can identify some prominent relationships. For example, if part of a cluster borders a shoreline, it makes sense to report this information. A relationship can exist even if the cluster and the feature are relatively far apart. For example, if a cluster of expensive houses is a kilometre away from the shoreline, yet follows the shoreline reasonably well, one would suspect a possible relationship. Thus, if the cluster were on a slope, one might be led to believe that some or many of the houses in the cluster yield a splendid view. Relationships such as those described above may be of interest to such groups as the real estate community, Geographic Information Systems (GIS) analysts, demographers, and planners. Although our testbed is GIS, other spatial domains apply equally well. To show that BSM can be applied concretely, this paper uses real estate examples, and regularly and irregularly shaped geographic features in the city of Vancouver. The main contribution of this paper is the development of an effective BSM technique. We have confirmed the effectiveness of our results with the aid of visualization tools. As a preview of performance, our adaptive BSM algorithm uses approximately 0.02 CPU seconds per cluster-feature combination.

2 Defining Shapes of Clusters by Alpha-Shapes

As we have indicated, BSM will be performed on clusters and features. Each feature is defined by a sequence of points that make up a polygon. Most maps accessible by computer contain features described by an ordered sequence of (x, y) vertices defining the endpoints of a closed curve of line segments (edges). A cluster, on the other hand, is simply a set of unordered points with no specific boundary, perhaps having been produced in response to an SQL query. While clustering algorithms such as CLARANS [4, 11] show how to identify one or more clusters from a set of points, no specific boundaries are associated with those clusters. This leads us to our first question: What is the shape of a cluster? Given a (dense) cluster of points, most humans can easily visualize a polygon whose shape characterizes a cluster. While the shape of such a polygon may be intuitive to humans, it is undefined to machines.

2.1 Alpha-Shapes

We begin by defining the shape of a cluster as a *convex shape*, which is simply a closed boundary of a set of points such that the line segment connecting any two points in the set always lies on or interior to the boundary. Among convex shapes, the *convex hull* is the most natural choice, because it is the unique, minimum bounding convex shape enclosing a set of points [12]. Convex hulls are a powerful construct for many spatial applications, including data mining [9]; however, they are not very good at providing detailed shape information about nonconvex spatial objects. For example, if we were limited to using a convex hull to describe the cluster in Figure 1, most of the interesting aspects of the shape would be lost, and BSM would be difficult, if not impossible.

Because the shape of a cluster may be nonconvex, and because convex hulls are too crude for effective BSM, we turn to a generalization of convex hulls, namely *alpha-shapes* [3, 8]. While a cluster has a unique convex hull, there is a spectrum of alpha-shapes to consider, each alpha-shape being defined by some α-value. A zero α-value gives the convex hull, positive α-values produce cruder convex shapes for a given set of points, and negative α-values provide finer nonconvex resolutions. The boundary of the cluster in Figure 1 is the result of a negative α-value. The time complexity for constructing an alpha-shape is the same as for constructing the convex hull. The complexity is $O(n \log n)$, where n is the number of points in the set.

2.2 Applying Alpha-Shapes to BSM

While certain members of a family of alpha-shapes can describe the intuitive shape of a nonconvex cluster, it is unclear what α-value to use in general, because the choice can vary greatly from cluster to cluster. Furthermore, a fine resolution produced by a large negative α-value may be too fine in the sense that there may be severe, narrow cavities or fissures in the resulting polygon. To illustrate this point, consider Figure 1 again. In the right side of the cluster, we see a fissure (represented by a dotted area) which may be the result of certain negative α-values. Many such fissures may exist. For BSM, we want alpha-shapes that are neither too crude, nor too fine. In general, it is very difficult to determine the best α-value.

We ran many experiments to show how different α-values can affect the shape of a cluster.[1] We observed that an alpha-shape can become disconnected and can change significantly even for small changes in α. For the point sets that we tested, we observed that the best α-values were those for which the number of voids (holes) in the shape was zero, and the number of connected components was one. Even so, a single-component, zero-void alpha-shape can occur for several ranges of α-values. We chose the α-value for a given cluster to be the average of the midpoints of all ranges that yield one component and no voids. Using this value resulted in a reasonable alpha-shape for the cluster. The boundary is described by an ordered sequence of vertices (and hence, line segments), which we use in the BSM algorithm described next.

3 Quantifying Partial Matches

To develop an effective BSM algorithm, many questions have to be addressed. The first question concerns which parts of the boundaries to consider. Regardless of the relative positions of the cluster and the feature on a map, it makes sense (particularly for GIS applications) to consider only those parts of the boundaries that face each other. These facing curves are identified by calculating the outer common tangent lines of the cluster-feature pair. An example of how tangent lines delimit facing curves is shown in Figure 2. Our BSM problem is therefore reduced to determining how well two chains of line segments match.

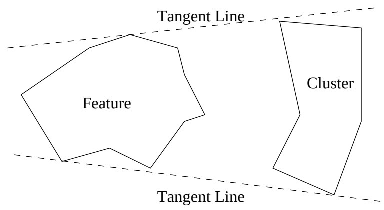

Figure 2: Identifying Facing Curves by Using Tangent Lines

3.1 Identifying the Difference Curve

The next question is how to determine whether two shapes match. One obvious approach is to translate the curves until they touch, and then calculate the area between them. The smaller the area between them, the greater the likelihood of a match. This naive approach has several problems. First of all, it is biased toward shorter curves. For example, a pair of long curves may match well, but receive a poorer score than a pair of short curves that do not fit quite as well. Secondly, a small local change in one curve (for example, a long, sharp protrusion) can seriously affect the score. These problems led us to consider examining the *change* in the distance between the curves as we traverse an appropriate *axis of orientation*. The axis of orientation or "line of scrimmage" separates the curves so that the cluster is on one side of the line, and the feature is on the other. Separating the curves in this manner helps prevent situations in which we have to deal with overlapping curves.

As we traverse the relevant part of the axis of orientation, we calculate the separation between the curves. The separation gives rise to a *difference curve*, shown in Figure 3, which is created by plotting the distance between the curves as the axis is traversed at discrete intervals. The intervals need not be uniform because our curves consist of line segments. A change in the slope of the difference curve results only when a vertex from either curve is encountered having the same w-coordinate (where w is the axis of orientation). If there are n vertices in

[1] The code we used to compute the alpha-shapes was Shape2D-v1.1 from the National Center for Supercomputing Applications and the Department of Computer Science at the University of Illinois at Urbana-Champaign. This package uses an α-value that differs from the original definition [3] mentioned in Section 2.1. The package's α-value is the reciprocal of the original α-value. The range of allowable α-values for this package results in a family of shapes ranging from the original, unconnected point set, up to and including the convex hull. Unless indicated otherwise, we use the original definition throughout this paper.

the cluster's curve and m vertices in the feature's curve, then at most $n+m$ distance calculations are required for this scheme.

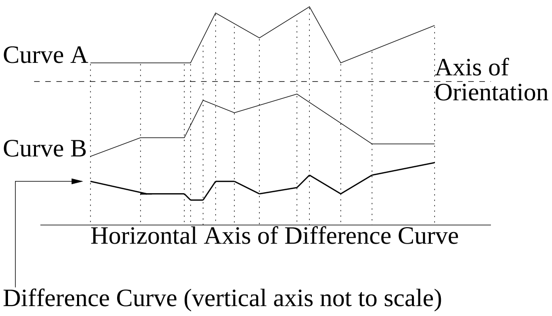

Figure 3: Difference Curve

3.2 Choosing the Axis of Orientation

A key aspect of constructing the difference curve is the choice of axis of orientation. Using Figure 4, we describe how to choose an appropriate axis of orientation. First, we calculate a bounding box for each of the facing curves, and then we construct tangent lines to these bounding boxes. The tangent lines are the dotted lines shown. We define our initial axis of orientation to lie in the middle of these two extremes, and we can rotate the axis through any angle ϕ, if ϕ does not go beyond a tangent line. For any angle ϕ, we calculate the separation according to where each curve intersects a line perpendicular to the corresponding axis.

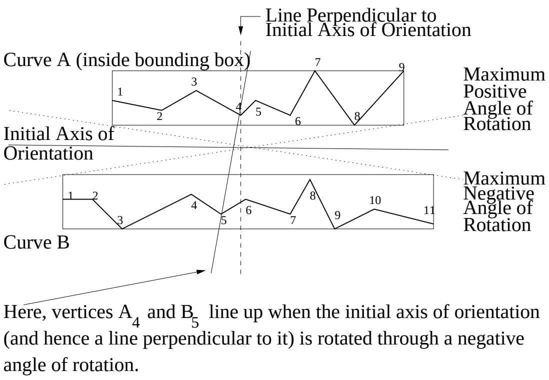

Figure 4: Axis of Orientation

Different axes of orientation result in different matches between peaks and valleys. For example, if ϕ=0, vertices A_6 and B_7 appear to line up or match, but this is not the situation with vertices A_4 and B_5, as shown by the dashed line. On the other hand, if we rotate the angle of orientation through a negative value of ϕ not exceeding the limit posed by the corresponding tangent line, we see that vertices A_4 and B_5 line up. Different values of ϕ can have considerable effect in determining how well the two facing curves match. Note that different angles of rotation may be more suitable for certain pairs of vertices. There is no way of knowing in advance which angle of ϕ is best. A solution to this problem is to perform a lengthy search, varying the angle along which the distance measurements are taken, and simply keeping the best score. (Section 5 provides optimizations of this approach.) Having described the importance of the axis of orientation, we can now calculate the degree of partial match.

3.3 Quantifying the Degree of Partial Match

Let us re-visit Figure 3. If V_A is the set of vertices belonging to Curve A, and V_B is the set of vertices belonging to Curve B, then let $V = V_A \cup V_B$. Let $dist_i$ be the distance between the two curves (perpendicular to the axis of orientation) at vertex $V_i \in V$, and let L_i be the distance along the axis of orientation between vertices V_i and V_{i+1}. Define θ_i to be the absolute value of the slope of the difference curve (piecewise), that is, $\theta_i = \mid dist_{i+1} - dist_i \mid / L_i$. Finally, let Δ_g define the tolerance or maximum acceptable slope of a segment of the difference curve. The following equation shows how we calculated the scores, where n is the number of segments:

$$score = \sum_{i=1}^{n-1} \begin{cases} L_i(\Delta_g - \theta_i) & \text{if } \theta_i < \Delta_g \\ 0 & \text{otherwise} \end{cases}$$

At first glance, this equation seems to satisfy our basic notion of fairness and of assigning higher scores to flatter difference curve segments. If $\theta_i < \Delta_g$, a larger L_i increases the score, and the closer θ_i is to zero, the larger the score. There are, however, limitations to this relatively simple approach. We address those limitations next.

4 Optimizations

4.1 Optimizing the Scoring Function

To evaluate the effectiveness of our BSM algorithm, we ran it against a set of geographic features of Vancouver. We observed that the resulting score often did not highlight the strength or quality of an excellent match. In other words, there was no strong distinction between excellent matches and middling matches. This was further complicated by the presence of bad segments within a curve. For example, Figure 5 shows two pairs of curves that may receive identical scores using the existing linear equation—depending, of course, on the value of Δ_g.

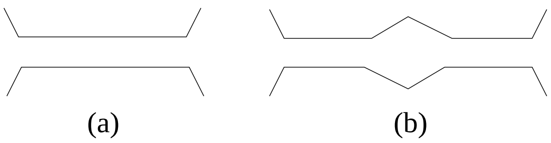

Figure 5: Two Pairs of Curves that May Receive Identical Scores Using the Linear Equation

The linear equation was changed in several ways. First of all, we now count both good and bad segments. Good segments are those for which $\theta_i < \Delta_g$; bad segments are those for which $\theta_i > \Delta_b$. Secondly, we show favouritism toward flatter difference curves. To achieve this goal, we use an exponential function instead of a linear function. Thirdly, because the difference between a good segment and a bad segment can be highly subjective, we allow for a buffer zone, whereby we assign neither a penalty nor a premium to segments falling into the range $\Delta_g \leq \theta_i \leq \Delta_b$. These three improvements produce the final form of our equation:

$$score = \sum_{i=1}^{n-1} \begin{cases} L_i e^{(\Delta_g - \theta_i)c} & \text{if } \theta_i < \Delta_g \\ -L_i e^{\min[1,\ (\theta_i - \Delta_b)/c]} & \text{if } \theta_i > \Delta_b \\ 0 & \text{otherwise} \end{cases}$$

The ceiling $L_i e$ is used to bound the penalty for bad segments; otherwise, if θ_i greatly exceeded Δ_b, the penalty could be too severe (even for a relatively small L_i), and weaken the overall score for what might otherwise be a very good match. Furthermore, a constant $c > 1$ is used to help normalize the positive and negative terms because Δ_g and Δ_b are likely to be small compared to θ_i. Without c (and the ceiling), we would typically get small positive terms and large negative ones, making partial matches difficult to detect. We tried numerous values for Δ_g, Δ_b, and c, and found that $\Delta_g = 0.1$, $\Delta_b = 0.3$, and $c = 10$ gave reasonably good results—roughly comparable to the results a human might obtain from a quick visual observation of two facing curves. Although we cannot claim that these are the ideal values for a particular application, our experimental results suggest that these are good heuristics for our testbed.

The equation readily supports the identification of *partial* matches within a pair of facing curves, which is particularly important for this type of knowledge discovery application, because it may be quite unlikely that two facing curves yield a perfect match. We also noted that some curves match quite well except for the final segment at either end of a curve's chain (that is, those segments touching the tangent lines). It seems unfair to penalize an otherwise good curve that just happens to "tail off" at its extremities. For example, although the rightmost segment of Curve B in Figure 6 falls within the tangent lines, it obviously does not contribute to curve matching. We decided not to count these bad areas in the score. Of course, bad areas lying between good areas are counted.

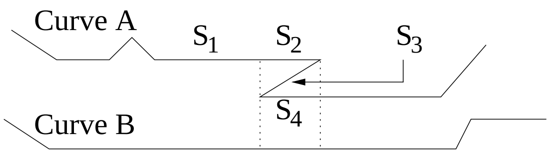

Figure 6: Curve A Has a Hidden Segment

4.2 Trivial Matches and Hidden Segments

Our testbed included a lot of rectangular features, such as, schools, playgrounds, parks, and shopping centres. We observed that many of the best scores resulted from trivial matches involving one-line segments. To understand why this is a problem, consider a rectangular shopping centre located directly north of a cluster. If one segment in the cluster (especially a long segment) is approximately parallel to a line segment in the feature, then the resulting score can be quite high. We cannot claim, however, that this constitutes a relationship of interest. On the other hand, if many line segments match, such as when an irregularly shaped cluster matches an irregularly shaped feature (for example, a cluster following a jagged shoreline), it makes sense to report this match as a relationship of interest. We therefore apply the following rule. After determining the tangent lines, if the number of vertices in either the cluster's curve or the feature's curve is ≤ 3, we simply ignore the feature, and move on to the next one. We chose the value 3 because many features in an urban area are rectangular (or nearly rectangular). If an entire curve consists of a corner (that is, two adjacent edges), we simply skip the calculation, thereby avoiding many trivial matches. Our experiments confirm that this is a useful heuristic.

Another issue that we had to address concerned overcounting. We encountered this situation when dealing with hidden curves, that is, curves that backtrack. When calculating the score for the pair of curves shown in Figure 6, we ignore the hidden parts of Curve A, which are the line segments in the diagram labelled S_2 and S_3, but we retain S_1 and S_4. If we were to include S_2, an undeservingly high score would result from counting part of Curve A twice.

5 Varying the Orientation More Intelligently

Section 3.2 described a brute-force approach for choosing the best axis of orientation for the facing curves. To summarize, we tried the range of allowable ϕ values using a small linear increment, and then chose the axis that produced the highest score. Although this approach worked, it did not scale very well. Efficiency is important because we want to try many features.

During our analysis of the brute-force results, we plotted each score against its corresponding value of ϕ. We noticed that many graphs contained prominent patterns. For example, a graph often contained several spikes of high scores that stood out from the rest of the graph. This suggested that we might be able to exploit these patterns, and avoid having to test a wide range of ϕ values. Several alternatives were tested:

1. Search for the first local peak.[2] Begin by calculating the score when ϕ=0, and then either increase or decrease ϕ, depending on which direction produces a score greater than or equal to the previous one. After the direction is determined, continue increasing (or decreasing) ϕ until the new score is less than the previous score, or the limit of the range is reached. At that point, report the peak value as the score for that pair of curves.

2. Divide the range of allowable ϕ values into three equal parts, search for a local peak in each part, and report the highest of the three scores.

3. Use an adaptive technique. Begin by traversing the range of allowable ϕ values using a coarse increase (for example, 1/30 of the total range). While progressing through these coarse increases, retain the best score and its corresponding ϕ. After calculating the score for a subsequent ϕ_k, if the slope of the graph changes from increasing to decreasing, note that a peak occurs somewhere between ϕ_{k-2} and ϕ_k; therefore, traverse the range defined by ϕ_{k-2} and ϕ_k using a finer increase, to see if a better score exists. (Following this, continue traversing the original range with the coarse increment.)

[2] Here, we are not referring to a local peak in one of the facing curves, but rather a local peak in the graph of score versus ϕ.

	Relative Effectiveness	Relative Efficiency
Brute-Force	100	100
1 Local Peak	92	58
3 Local Peaks	96	83
Adaptive	100	52

Table 1: Relative Effectiveness and Efficiency of Various Orientation Strategies

Table 1 shows a representative comparison of the four approaches, with the brute-force method being the baseline. The adaptive method has an effectiveness of 100% which means that the score returned by this method is equal to that returned by the brute-force method. The CPU time used is 52% of that used by the brute-force method. Our tests indicate that the adaptive method is as good as the brute-force approach, while taking approximately one-half the time. The adaptive approach is also the fastest of these four approaches. Our testbed had 326 real-life features in it, with a broad mix of convex, non-convex, large, small, regularly shaped, and irregularly shaped features. Using a Sun**[3] SPARCstation** LX computer with 24 MB of main memory, we could examine a cluster-feature pair in approximately 0.02 seconds, on average.

6 Conclusions and Ongoing Work

We have described a problem in spatial knowledge discovery involving boundary shape matching. We began by introducing alpha-shapes as a tool for finding a boundary to describe the shape of a cluster of unordered points. We then described how to perform BSM, and how to quantify the results. Several optimization and orientation strategies were presented. Our experiments have shown that our adaptive approach produces efficient and effective BSM, including partial and total matches of facing curves.

Much of the work described in this paper deals with calculating the score for a specific cluster-feature pair. Because approximately 0.02 seconds are required to calculate the score for a pair of curves, this implies that 1000 seconds would be required to examine 50,000 features, plus the time required to rank the scores to yield the top-k list of interesting features. We are exploring ways of producing the top-k list for a large set of input features more efficiently. For example, our current research reveals that filtering techniques can lead to significant performance gains.

[3] Sun and SPARCstation are trademarks of Sun Microsystems, Incorporated.

Acknowledgements

Our research is partially sponsored by NSERC Grant OGP0138055, IRIS-2 Grant HMI-5 and Grant IC-5, and CITR Grant on "Distributed Continuous-Media File Systems."

References

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

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

[3] H. Edelsbrunner, D. 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, 1983.

[4] M. Ester, H. Kriegel and X. Xu. "Knowledge Discovery in Large Spatial Databases: Focusing Techniques for Efficient Class Identification", *Proceedings of the 4th International Symposium on Large Spatial Databases (SSD'95)*, pp. 67-82, 1995.

[5] W. J. Frawley, G. Piatetsky-Shapiro, and C. J. Matheus. "Knowledge Discovery

in Databases: An Overview", *Knowledge Discovery in Databases*, Piatetsky-Shapiro and Frawley (eds.), AAAI/MIT Press, pp. 1-27, 1991.

[6] J. Han, Y. Cai, and N. Cercone. "Knowledge Discovery in Databases: an Attribute-Oriented Approach", *Proceedings of the 18th VLDB Conference*, pp. 547-559, 1992.

[7] D. Keim, H. Kriegel, and T. Seidl. "Supporting Data Mining of Large Databases by Visual Feedback Queries", *Proceedings of the 10th International Conference on Data Engineering*, pp. 302-313, 1994.

[8] D. Kirkpatrick and J. Radke. "A Framework for Computational Morphology", *Computational Geometry*, G. Toussaint (ed.), The Netherlands: Elsevier Science Publishers B.V., 1985, pp. 217-248, 1985.

[9] E. M. Knorr and R. T. Ng. "Finding Aggregate Proximity Relationships and Commonalities in Spatial Data Mining", *IEEE Transactions on Knowledge and Data Engineering* (Special Issue on Data Mining), 1996, Forthcoming.

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

[11] R. Ng and J. Han. "Efficient and Effective Clustering Methods for Spatial Data Mining", *Proceedings of the 20th VLDB Conference*, pp. 144-155, 1994.

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