A Simple But Effective Improvement to the Plumb Line Algorithm[1]

Ralf Hartmut Güting Zhiming Ding

LG Datenbanksysteme für neue Anwendungen
Fernuniversität Hagen, D-58084 Hagen, Germany
{rhg,zhiming.ding}@fernuni-hagen.de

Abstract. *In this paper, we propose an improved plumb-line algorithm, the Partial-Scan Plumb-Line (PSPL) algorithm, to solve the point-in-region problem. In the PSPL algorithm, every edge of the region (or polygon) is represented as two half segments, and all half segments of the region are sorted according to their dominating points. This is a standard representation for regions in spatial databases, providing efficient support for many plane-sweep algorithms. To support PSPL additionally a "coverage number" is associated with each half segment. In this way, the algorithm can employ a binary search to quickly find the half segment whose dominating point is closest to the given point, and then access only the neighbouring half segments to evaluate the query, leading to dramatic performance improvements, especially for large regions.*

Keywords. spatial databases, plumbline algorithm, point-in-region, databases, algorithms

1 Introduction

The plumb-line algorithm is a simple but efficient solution to the "point-in-region" problem [8, 9]. Given a point *p* and a region *r*, the plumb-line algorithm draws a vertical half line from *p* upward, and then counts the number of intersections between the half line and the edges of the region. If the number is an odd number, then *p* is inside *r*, and otherwise it is outside. All edges of the region are considered as half open (left closed and right open, or vice versa) to avoid incorrect results when the half line goes through a vertex of the region. In addition, the algorithm needs to deal with some special cases, for instance, the situation when *p* is on the border of *r*. The idea of the plumb-line algorithm is illustrated in Figure 1.

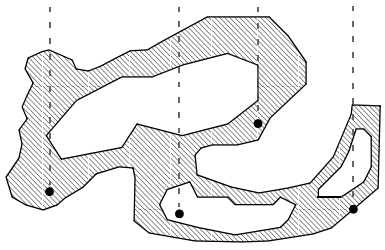


Fig. 1. Plumb-line algorithm

The complexity of the original plumb-line algorithm is O(*N*), where *N* is the number of edges of the region. In most cases, this is a very good performance already. Sometimes, however, this perfor-

1. This research was partially supported by the Deutsche Forschungsgemeinschaft (DFG) research project "Databases for Moving Objects", Gu 293/8-1.

mance is still not good enough, especially when the region is a large and complicated one. For instance, in the experimental data set (see Section 3), the region for the United States has more than 13,300 edges.

One possible solution to this problem is to design a special data structure just for this operator to speed up the computation. However, we consider an environment where the region representation is used in a spatial DBMS to support many different operations (e.g., computing the intersection between a line (curve) and a region, finding the common boundary between two regions, etc.). Then, a special purpose data structure just to support the point-in-region computation does not make sense. We have found, however, that by a very simple and small extension to the standard data structure for regions we can indeed dramatically speed up the point-in-region computation without incurring significant overhead for constructing regions or the storage requirements.

Regarding solutions to the point-in-region (or point-in-polygon) problem in the literature, one has to distinguish between techniques using a given polygon without preprocessing, and methods that provide a specialized data structure. A survey of practical methods for the first case is given in [5]. Besides slight variations of the plumb-line algorithm (called *crossings test* there), one can also compute triangles connecting some vertex with all edges of the polygon and check which triangles contain the query point, or sum the angles from the query point to all vertices of the polygon, for example. Some of these techniques perform a little better than others, as shown in [5]. Reference [6] studies improvements of the plumb-line or ray-crossing algorithm, and of algorithms based on angle summation, generally by cleverly rearranging the pieces of code for each test to speed up execution. However, all such techniques need to process each edge of the polygon in some way and therefore remain with the basic O(n) time complexity.

When specialized data structures can be used, the problem has a theoretically optimal solution, since it can be treated as locating a point in a planar subdivision. This can be done in O(log n) time and O(n) space, e.g. by Kirkpatrick's triangulation refinement method [7]. Other optimal solutions exist.

Since the theoretical methods are complex and do not necessarily perform well in practice, simple data structures for speeding up the point-in-polygon test have also been developed, such as the *bin test* or the *grid cell test* method (both described in [5]). The bin test method decomposes the bounding box of the polygon into k vertical stripes whose bottom and top edges are adjusted to touch the polygon (these are the bins). Then one stores for each bin the edges overlapping it. Given a query point, one finds the appropriate bin in constant time, and applies the plumb-line algorithm to its edges. The grid cell test divides the bounding box of the polygon into a regular $k \times k$ grid. Each grid cell is classified as either inside, outside, or indeterminate. For the latter category, again a list of overlapping segments is stored. Given a query point p that falls into an indeterminate cell, one applies ray-crossing from p to one corner of the cell whose state (inside or outside) was computed in preprocessing. These methods have been experimentally studied in [5] using parameters $k = 20$ or $k = 100$. They are much faster than the O(n) time methods mentioned above, working on the average in practically constant time. However, this is at the expense of large space requirements, since beyond the space for bins or grid cells an edge (or pointer to it) needs to be stored with each bin or cell it overlaps.

In comparison, our method does use a specialized data structure, but one that is only a very small and simple extension to the standard data structure for regions used in query processing. The

additional space requirements (one number per line segment) and construction time (one scan of an array of segments, see Algorithm 1) are negligible. The latter is important as regions are also constructed as intermediate results in query processing. However, the benefits are great: whereas the worst-case running time is still O(n) for certain "pathological" regions (see Section 3), for regions with short boundary segments the running time is close to O(log n).

Hence, none of the linear methods mentioned above is able to compete with our method, and the other techniques using specialized data structures are obviously inapplicable in our context.

2 Partial-Scan Plumb-Line Algorithm

2.1 Preliminaries

In our research with the spatial algebra in the Secondo system [1, 2], we have adopted the data structures described in the ROSE algebra [4, 3]. For a given region, we map every edge of the region to two half segments, and all these half segments are sorted and then stored in an array. A half segment is just a segment for which either the left or the right end point is considered as significant. An ordered list of half segments is the right input for a plane-sweep algorithm processing the region, and it also supports various algorithms based on *parallel scan* of two regions or other spatial values [3].

Definition 1 (Half Segment). Let $S = \{(p, q) \mid p, q \in X \times Y, p < q\}$ denote the set of line segments in the $X \times Y$ plane, where p and q are the *left* and *right* end point respectively.[1] The set of half segments is defined as follows:

$$H = \{(s, d) \mid s \in S, d \in \{left, right\}\}$$

A half segment $h = (s, d)$ consists of a segment s and a flag d emphasizing one end point of the segment, which is called the dominating point of h. If $d = left$, then the left (smaller) end point of s is the dominating point of h, and h is called "left half segment". Otherwise, the right end point of s is the dominating point and h is called "right half segment". Therefore, every edge of the region, which is a segment in the $X \times Y$ plane, denoted by s, can be mapped to two half segments: (s, *left*) and (s, *right*), which we also denote as s_l and s_r, respectively.

A region is represented essentially as an ordered list (array) of half segments. The order used is the one suitable to support plane-sweep algorithms, basically lexicographic order on dominating points. Ties are resolved by further checks on being left or right end point and on slope [3].

2.2 The Algorithm

To improve the plumbline algorithm, the question is whether one can use the given data structure without looking at all edges. The idea is to implement the plumbline algorithm as follows:

1. The left end point is the smaller one in a lexicographical order of points.

1. Perform a binary search on the array of half segments for the x-position of query point p.
2. Then scan the array from the found position into one direction (say to the left). For every *left* half segment (ignore the *right* half segments) check whether it is above p and increment a counter, if so. Stop the scan when all segments overlapping p's x-coordinate have been encountered. Return true if the counter is odd.

Usually one can expect that edges covering the x-position of p can be found in a rather small neighbourhood of $p.x$. The problem is to discover when to stop the scan, as it is not clear whether there are still edges coming up covering $p.x$. The additional idea needed is illustrated in Figure 2.

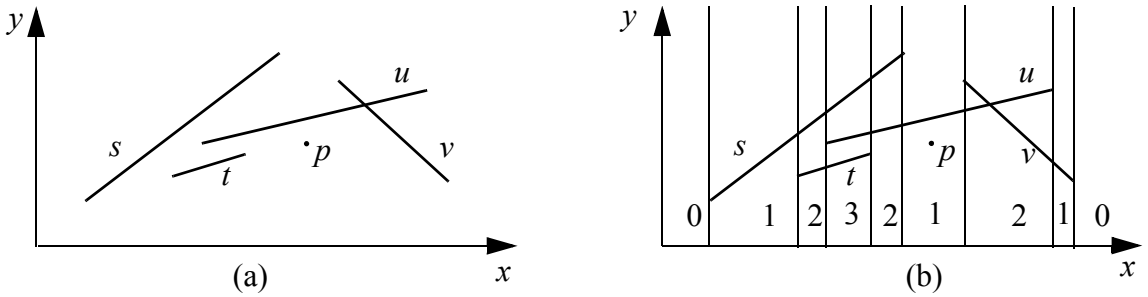


Fig. 2. (a) configuration of segments for plumbline search (b) extended by coverage numbers

Consider first Figure 2(a). Scanning from the position of p to the left, one encounters s_r (the right end, or half segment, of s), then t_r, then u_l. At this point, the scan could stop, as all segments lying either above or below p have been encountered. Unfortunately this is not known. If it were known that there is only one such segment, the scan could safely stop here.

So the idea is to put exactly that information into the data structure as shown in Figure 2(b). For each vertical stripe of the plane between two x-coordinates we record the number of segments crossing this stripe. This number can easily be computed in a single scan through the array of half segments, as shown in Algorithm 1. It associates the coverage number of a stripe between two half segments with the half segment with left end point *left* of the stripe.

Algorithm 1. Coverage number associating algorithm

```
INPUT : SHSA=<h1,h2,...hn> (Sorted Half Segment Array)
OUTPUT: SHSA with Coverage Number Associated

CoverageNo = 0;
FOR i=1 TO n DO
  IF (hi has left dominating point) THEN
    CoverageNo = CoverageNo + 1;
  ELSE
    CoverageNo = CoverageNo - 1;
  ENDIF;
  hi.coverageno = CoverageNo;
ENDFOR;
```

The PSPL algorithm then after the binary search initializes a counter to the coverage number of the stripe containing p. Scanning to the left, the counter is decremented whenever a segment overlapping p's x-position is found. Hence the scan can stop when the counter is zero. The complete

algorithm is presented as Algorithm 2. We need to consider the special case that several adjacent half segments may have the same x coordinate value as that of p (we call these half segments the "sibling" half segments of p, denoted by sibling(p)). Suppose sibling(p) = $<h_j, h_{j+1}, ..., h_k>$, and the result of the binary search is $i \in [j\text{-}1, k\text{+}1]$. In this case, the algorithm needs to check all the sibling half segments first, and then gets the coverage number from $h_{j\text{-}1}$.

Algorithm 2. Partial-scan plumb-line algorithm

```
INPUT : p, r = SHSA = <h1, h2, ..., hn>;
OUTPUT: IsInside;

i = BinarySearch(p, r);
IF (p = dp(hi))∨(p = dp(hi+1)) THEN return true; ENDIF;
intersection = 0;

IF (siblings(p)≠NULL)∧(siblings(p)=<hj, ..., hk>) THEN
    FOR h in siblings(p) DO
      IF (h has left dominating point) THEN
        IF (h contains p) THEN return true; ENDIF; //p is on-border of r
        IF (h intersects with halfline(p)) THEN
          intersection = intersection + 1;
        ENDIF;
      ENDIF;
    ENDFOR;
    i = j - 1;
ENDIF;

candidatenum = hi.coverageno;
candidatefound = 0;
WHILE (i >0) AND (candidatefound < candidatenum) DO
   IF (hi has left dominating point) THEN
     IF (hi contains p) THEN return true; ENDIF;
     IF (hi intersects with halfline(p)) THEN
       intersection = intersection+1;
     ENDIF;
     IF (p.x is between dp(hi).x and sp(hi).x) THEN
       candidatefound = candidatefound + 1;
     ENDIF;
   ENDIF;
   i = i - 1;
ENDWHILE;

IF (intersection is odd) THEN return true;
ELSE return false;
ENDIF;
```

3 Performance Analysis

The complexity of the PSPL algorithm is composed of two parts, the cost of the binary search, and the cost of the further scan. The first part is relatively fixed, which is O(log N), where N is the number of edges of the region. However, the second part can vary a lot according to different data sets. For a certain region, if the length of the edges is evenly distributed, then the overhead of the further scan can be very small. Otherwise, the algorithm may need to go a long way to find all the

candidate half segments. In the worst case, the algorithm may need to scan the whole half segment array.

In order to compare the performance of the PSPL algorithm with that of the original plumb-line algorithm, we have conducted a series of experiments based on the world map data set [10, 11]. Table 1 presents some statistics on the data set.

number of points (cities)	1232
number of regions (countries)	239
average number of edges (for regions)	317.23
maximum number of edges (for regions)	13,318
minimum edge numbers (for regions)	4

Table 1. Statistics of the experimental data set

In the experiments, we compare each city with each country and invoke the PSPL algorithm after a preceding MBR test, that is, whenever the point is within the MBR of the region. For each such call, we record the number of accessed half segments. This number is then compared with the number of edges of the region, which is supposed to be the number of accesses for the original plumb-line algorithm. Figure 3 shows the statistics of the experimental results, explained in more detail in Table 2.

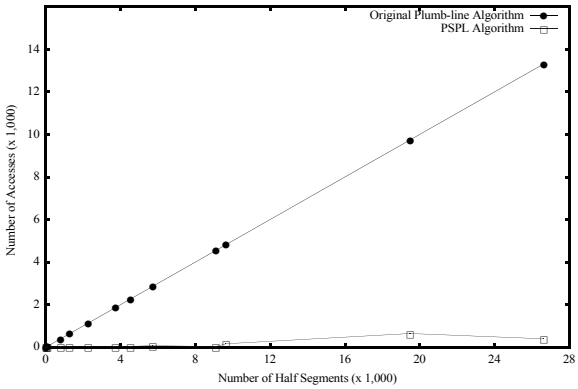


Fig. 3. Comparison of Number of Accesses

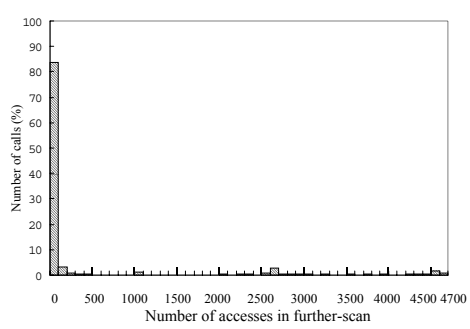
As stated earlier, the cost of the PSPL algorithm consists of the cost of the binary search and the cost of the further scan. In Table 2, we list the detailed information concerning these two parts. For this purpose, we classify countries into groups by their numbers of edges. The three largest countries are listed separately.

From the above experimental results we can see that the PSPL algorithm can greatly reduce the cost involved in solving the point-in-region problem. On average, the cost is only 5% or less of the original cost. Since the binary search can be finished in O(log N) time, the PSPL algorithm is relatively more efficient when the number of edges of the region increases.

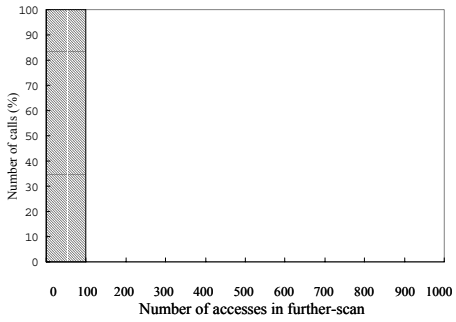
By a closer look at the experimental results, however, we find that for some regions such as the USA and Canada, the overhead of the PSPL algorithm is higher than we have expected, while for

Number of Edges (= Number of Half Segments / 2)	Average Number of Edges	Average Accesses of PSPL	Accesses in Further Scan			Average Accesses in Binary Search
			Min	Max	Average	
0-500	217.56	21.48	1	136	13.06	8.42
500-2000	625.56	18.41	1	319	8.28	10.13
2000-4800	2968.5	40.4	1	208	28.1	12.3
4820 (Russia)	4820	16.73	1	89	3.73	13
9744 (Canada)	9744	647.19	4	3076	633.19	14
13318 (USA)	13318	398.87	1	4632	383.87	15

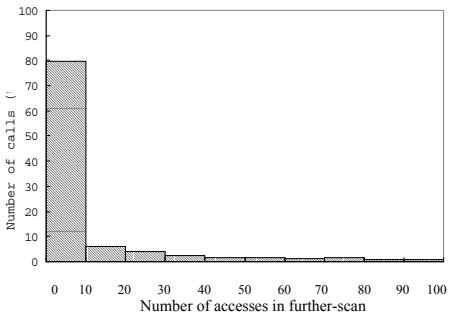
Table 2. Detailed cost of the PSPL Algorithm



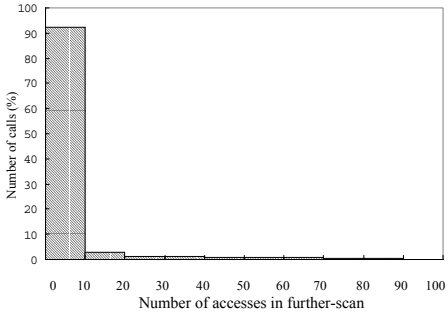
(a) distribution of the cost for the region of USA



(b) distribution of the cost for the region of Russia



(c) further analysis of Figure 4(a) (0-100)



(d) further analysis of Figure 4(b) (0-100)

Fig. 4. Histogram of the cost involved in the further scan

the region of Russia, which is also very big and complicated, the cost is quite small. Figure 4 shows the histograms of the cost of the further scan concerning the regions of USA and Russia.

The reason for this result is that in the regions of USA and Canada, the length of the edges is not evenly distributed. In the data set, there is a long edge between the regions of USA and Canada (see Figure 5), and as a result, the algorithm needs to go for a long distance to reach the half segment corresponding to the long edge, as illustrated in Figure 5.

One solution to this problem is to employ a trivial "break-up" technique. That is, whenever an edge is longer than a predefined threshold ξ, it will be "broken up" into two or more shorter edges so that in the system, no edges are longer than ξ.

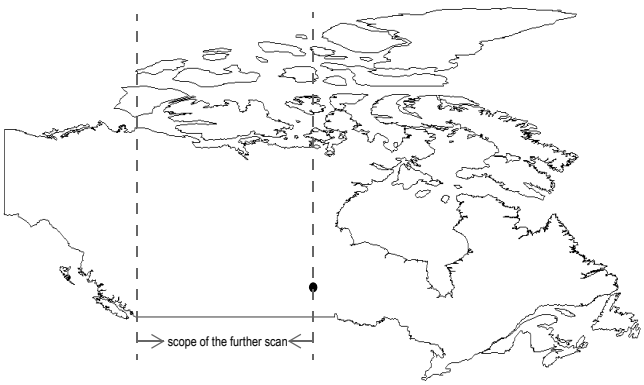


Fig. 5. An unevenly distributed region (Canada)

4 Conclusion

The plumb-line algorithm is a basic algorithm to solve the point-in-region problem. However, the original algorithm needs to access all the edges of the region, which can be a considerable overhead in some circumstances, especially when the region is a big and complicated one. In this paper, we have proposed an improved plumb-line algorithm, the Partial-Scan Plumb-Line (PSPL) algorithm. For regions with short edges it reduces the running time from O(n) to about O(log n) on the average, and so greatly speeds up query processing.

References

[1] S. Dieker and R. H. Güting, Plug and Play with Query Algebras: SECONDO. A Generic DBMS Development Environment. *Proc. of the Int. Database Engineering and Applications Symp* (*IDEAS*), Japan, September 2000.

[2] R.H. Güting, T. Behr, V. Almeida, Z. Ding, F. Hoffmann, and M. Spiekermann, SECONDO: An Extensible DBMS Architecture and Prototype. Fernuniversität Hagen, Informatik-Report 313, 2004.

[3] R.H. Güting, Th. de Ridder, and M. Schneider, Implementation of the ROSE Algebra: Efficient Algorithms for Realm-Based Spatial Data Types. *Proc. of the 4th Intl. Symposium on Large Spatial Databases*, Portland, August 1995.

[4] R.H. Güting and M. Schneider, Realm-Based Spatial Data Types: The ROSE Algebra. *VLDB Journal,* (4) 1995.

[5] E. Haines, Point in Polygon Strategies. *Graphics Gems IV,* Paul Heckbert (ed.), Academic Press, San Diego, 1994.

[6] K. Hormann and A. Agathos, The Point in Polygon Problem for Arbitrary Polygons. *Computational Geometry*, 20(3), 2001.

[7] D.G. Kirkpatrick, Optimal Search in Planar Subdivisions. *SIAM Journal of Computing*, 12(1), 28-35, 1983.

[8] P. Rigaux, M. Scholl, and A. Voisard, *Spatial Databases: With Application to GIS.* Morgan Kaufmann Publishers. 2002.

[9] M. F. Worboys, *GIS - A Computing Perspective*. Taylor & Francis, 1995.

[10] World countries shape data (world_countries_shp.zip), available at http://www.bluemarblegeo.com/products/worldmapdata.php?op=download.

[11] World cities coordinate data, available at http://www.koordinaten.de/online/koord.shtml