Efficient Geo-fencing via Hybrid Hashing: A Combination of Bucket Selection and In-bucket Binary Search

SUHUA TANG, The University of Electro-Communications
YI YU, National Institute of Informatics
ROGER ZIMMERMANN, National University of Singapore
SADAO OBANA, The University of Electro-Communications

Geo-fencing, as a spatial join between points (moving objects) and polygons (spatial range), is widely used in emerging location-based services to trigger context-aware events. It faces the challenge of realtime processing a large number of time-variant complex polygons, when points are constantly moving. Following the filter-and-refine policy, in our previous work, we proposed to organize edges per polygon in hash tables to improve the performance of the refining stage. The number of edges, however, is uneven among buckets. As a result, some points that happen to match big buckets with many edges will have much longer response than usual. In this paper, we solve this problem from two aspects: (i) Constructing multiple parallel hash tables and dynamically selecting the bucket with fewest edges, and (ii) Sorting edges in a bucket so as to realize the crossing number algorithm by binary search. We further combine the two to suggest a hybrid hashing scheme, which takes a better tradeoff between realtime pairing points with polygons and system overhead of building hash tables. Extensive analyses and evaluations on two real-world datasets confirm that the proposed scheme can effectively reduce the pairing time, in terms of both the average and distribution.

Categories and Subject Descriptors: H.2.8 [**Database Applications**]: Spatial Databases and GIS

General Terms: Algorithms, Performance

Additional Key Words and Phrases: Geographic information system, Spatial join, Crossing number algorithm, Geo-fencing, Indexing, Hashing, Selection diversity

ACM Reference Format:
Suhua Tang, Yi Yu, Roger Zimmermann, and Sadao Obana, 2014. Efficient Geo-fencing via Hybrid Hashing: A Combination of Bucket Selection and In-bucket Binary Search. *ACM Trans. Spatial Algorithms Syst.* 9, 4, Article 39 (March 2010), 22 pages.
DOI: http://dx.doi.org/10.1145/0000000.0000000

1. INTRODUCTION

Advancements in mobile technologies have led to ubiquitous communications, which enable users to connect to the Internet all the time. Meanwhile, techniques of satellite positioning [Misra and Enge 2010] and indoor localization [Chintalapudi et al. 2010] are integrated into smartphones as well. Location information of latitude and longitude, however, is not directly meaningful to most applications. Instead, finding user's semantic location, or context, by matching their coordinates with electronic maps is a key function. For example, mentioning a user's current location as Tokyo Disneyland

This research has been supported in part by the Singapore National Research Foundation under its International Research Centre @ Singapore Funding Initiative and administered by the IDM Programme Office through the Centre of Social Media Innovations for Communities (COSMIC).
Author's addresses: S. Tang and S. Obana, Graduate School of Informatics and Engineering, The University of Electro-Communications, Japan; Y. Yu, Digital Content and Media Sciences Research Division, National Institute of Informatics, Japan; R. Zimmermann, School of Computing, National University of Singapore.

DOI: http://dx.doi.org/10.1145/0000000.0000000

is more meaningful than $35.6328°N$, $139.8806°E$. Many location-based services (LBS), especially context-aware ones [Baldauf et al. 2007] which can automatically adapt operations to current user context, have recently been introduced to communication systems.

Geo-fencing [Küpper et al. 2011] plays an important role in the context detection of LBS. Here, a virtual geo-fence is defined for a spatial range of interest with a polygon specifying the geographic boundary of the range[1]. An event is automatically triggered and sent to predetermined targets when a user (point) enters or leaves the geo-fences. Geo-fencing has many promising applications [Bareth et al. 2010; Martin et al. 2011; Reclus and Drouard 2009] and is already exploited in different systems (e.g., Foursquare, Placecast, Sensewhere, Zentracker). The users (points) may constantly move and generate continuous spatial data stream. In some scenarios, the geo-fences are fixed. For example, in kid safety tracking, parents pre-set several ranges (e.g., school, nearby parks) as safe areas for their children, and get notified when their children leave these ranges. In other scenarios such as mobile advertising (a store dynamically adjusts its advertising range based on the current number of customers), flooding areas and volcanic eruption cordon, the geo-fences change with time.

Geo-fencing is to detect whether a point (user) is inside or within a distance of a polygon (geo-fence), in other words, pairing a point against a polygon. It is a special form of spatial join [Jacox and Samet 2007] between points and polygons. Spatial join deals with general spatial objects (polygons) and typically uses the filter-and-refine policy: (i) Filtering stage. Spatial join is performed on the approximations, e.g., minimum bounding rectangles (MBR) of objects. (ii) Refining stage. Rough results found in the filtering stage are examined with full objects, by using the plane-sweep technique [Preparata and Shamos 1985]. Most previous works [Nobari et al. 2013], including the query indexing method [Prabhakar et al. 2002], focused on the filtering stage. As for geo-fencing, the filtering stage is similar to that of spatial join. In the refining stage, the INSIDE detection can be realized by either the crossing number (CN) algorithm [Shimrat 1962], or the winding number algorithm [Hormann and Agathos 2001]. As polygons may contain many edges (vertices) to accurately represent a spatial range of interest, the computation cost via either scheme in the refining stage is a big burden.

Due to the increasing importance of geo-fencing, a task was raised by the ACM SIGSPATIAL GIS Cup 2013 [Ravada et al. 2013] as a contest, aiming at finding efficient algorithms. Out of 29 submitted algorithms, three [Zhou et al. 2013; Yu et al. 2013a; Li et al. 2013] were selected as the best in terms of the overall performance of both accuracy and execution speed. The three selected papers share some similar ideas, all using R-tree [Guttman 1984] to organize MBRs of polygons for the filtering stage. In the refining stage, interval index [Zhou et al. 2013] is used for managing edges of a polygon while edge-based hashing is used in [Yu et al. 2013a], and the two methods have similar performance [Ravada et al. 2013].

This paper extends our previous work [Yu et al. 2013a] on pairing points with polygons, and further improves the indexing design for managing edges of polygons in the refining stage. Organizing edges of a polygon in a hash table helps to reduce the computation cost. However, a well-known problem of hashing [Indyk and Motwani 1998; Yu et al. 2013b] is the bias of samples in the buckets. This problem gets serious when many edges are located in few buckets, which not only degrades the efficiency of indexing, but also leads to much longer response for some users. We analyze the distribution of the number of edges per-bucket and the properties of edges inside a bucket. On this basis, we propose a hybrid hashing scheme and the contribution is twofold: (i) Multiple,

[1]It is also possible to use the coverage of wireless cells (e.g., iBeacon) to specify geo-fences.

parallel hash tables are constructed to store edges of polygons, and *selection diversity*[2] [Neasmith and Beaulieu 1998] is applied to dynamically find the most proper bucket (with fewest edges) for each point. (ii) Large buckets are split into atomic sub-buckets, where edges are *sorted* in order. This enables the efficient implementation of the CN algorithm via binary search. Extensive analyses and evaluations on two real-world datasets confirm that the proposed scheme effectively reduces the pairing time, and achieves a better tradeoff between realtime pairing points with polygons and system overhead of updating polygons. This scheme can be applied to different LBS to ensure a fast response even in the presence of a large number of complex geo-fences.

The rest of this paper is organized as follows: Section 2 briefly reviews some related work on moving object database, range query, and indexing techniques for geo-fencing. The proposed algorithm is discussed in detail in Section 3. We introduce the framework, describe the two main techniques of bucket selection and sorting edges. On this basis, we suggest a hybrid hashing scheme and theoretically analyze its performance. Experimental results of execution time and storage are shown in Section 4. Finally, Section 5 concludes the paper.

2. RELATED WORK

With the widespread of positioning techniques and wireless communications, moving objects (people with smartphones, vehicles) can measure their locations and report to servers in a cloud. In the cloud, a database can be used to manage the location information. For example, a scalable location-aware data stream server is implemented on top of the data stream management system [Mokbel et al. 2005]. To facilitate LBS, it is also necessary to provide location-dependent queries [Ilarri et al. 2010] on moving objects. Range query [Xu and Wolfson 2003; Wu et al. 2006] is such an example, finding among a large number of moving objects which are in the spatial ranges of interest.

Moving objects database [Theodoridis 2003; Wolfson and Mena 2004; Güting and Schneider 2005] extends database technology to support the representation and query of moving objects in databases, and tracks the trajectories of objects by timestamp. Spatio-temporal indexing techniques are suggested to facilitate fast spatial access to moving objects. For example, time-parameterized bounding rectangle is used to approximate the trajectory of moving objects, and organized in a time-parameterized R-tree [Saltenis et al. 2000]. An alternative way is STRIPES, which is based on dual transformation [Patel et al. 2004], i.e., representing moving objects (a trajectory with start point and velocity) in two dimensions as a static point in four dimensions (position and velocity). These two schemes are further experimentally evaluated in [Sowell et al. 2013].

Building an index for moving objects and using spatial ranges as queries often suffer from frequent updates of the index due to the continuous moving of objects. When the query ranges are relatively stable, an alternative method is to reverse the role of range queries and data of moving objects, more specifically, building a query index [Prabhakar et al. 2002] (e.g., an R-tree) for spatial ranges. Moving objects associated with each range are found initially, and incremental adjustment is performed to check whether a moving object still matches the spatial range. Velocity constrained indexing is used to reduce the number of updating index of moving objects: using an old index, but expanding MBRs of the query range based on the moving distance of objects. A similar idea is suggested in [Cheema et al. 2011] for continuous monitoring of objects, where a distance-based range query is not re-evaluated if the query remains in a safe zone.

[2]Selection diversity is a typical technology used in wireless communication to mitigate channel fading.

Our work studies geo-fencing, where the predicates INSIDE and WITHIN have similar meaning as enclosure and nearness defined for spatial join. More specifically, geo-fencing is similar to range query on moving object database. But in geo-fencing, polygons are stored in the database and points are used as queries. In addition, complex polygons with many edges may be used to accurately represent geo-fences. In a general, formal definition of the geo-fencing problem, both a point (user) and a polygon (geo-fence) may be moving and thus have multiple instances. In other words, the geo-fences depend on situations [Pongpaichet et al. 2013]. Each instance (of points and polygons) is identified by an unique ID and has a timestamp. A geo-fence may have holes inside, therefore, a polygon contains one outer ring, and zero or more inner rings to exclude the holes. Geographically, multiple geo-fences may be near to each other, and a point may simultaneously appear in several overlapping polygons.

Most previous works on geo-fencing (and spatial join), including query indexing [Prabhakar et al. 2002], focus on the filtering stage. In the refining stage, the INSIDE detection relies on the CN algorithm [Shimrat 1962] or the winding number algorithm [Hormann and Agathos 2001]. The former counts the number of edges intersected by a ray starting from the point towards any direction[3], and the latter computes the point's winding number with respect to the polygon. Our investigation shows that a polygon in the dataset provided by GIS Cup'13 and in our own dataset can be composed of hundreds of edges. Hence, examining all edges via either algorithm in an exhaustive way is prohibitively expensive, and it is difficult to directly use these techniques to realize a realtime geo-fencing service with a large number of complex polygons.

Compared with previous works, we focus on the refining stage of geo-fencing. Geo-fencing has two main predicates: INSIDE and WITHIN. In the refining stage, a WITHIN predicate is divided into two steps: (1) First deciding whether a point is inside a polygon. (2) If this point is not inside the polygon, the distance of the point to each edge of the polygon needs to be computed. As this is the same as in our previous work [Yu et al. 2013a], we only focus on the INSIDE predicate in this paper.

Different indexing methods [Zhou et al. 2013; Yu et al. 2013a; Li et al. 2013] have been introduced to reduce the computation cost of the CN algorithm, so that only some edges of a polygon are examined for each point. Points and polygons are represented in the Cartesian coordinates[4]. A polygon $Poly$ consists of an ID ($Poly.ID$), a timestamp ($Poly.time$) and multiple rings with edges. The x-coordinate of a polygon spans a range $[x_{min}, x_{max})$. In the edge-based hashing scheme [Yu et al. 2013a], this range is *equally* divided into N sub-ranges $[x_0, x_1), [x_1, x_2), \cdots, [x_{N-1}, x_N)$, where $x_i = x_{min} + \Delta \cdot i$, $\Delta = (x_{max} - x_{min})/N$. The sub-range $[x_i, x_{i+1})$ is associated with the ith bucket B_i. An edge whose x-range overlaps $[x_i, x_{i+1})$ is stored in B_i. In this way, an edge can appear in multiple adjacent buckets. Assume a point P consists of an ID ($P.ID$), a timestamp ($P.time$), and a 2-D coordinate ($P.x$, $P.y$). With $\lfloor (P.x - x_{min})/\Delta \rfloor$ as the hash key, only edges in the associated bucket are examined by the CN algorithm. A similar method, where edges are organized via interval index, is suggested in [Zhou et al. 2013]. Because the bucket width is not constant, it takes more time to locate a bucket via a pre-built index.

The edges of a polygon are not evenly distributed in all buckets. Then, points matching big buckets with many edges will have a long pairing time, which will lead to a longer response time than usual. Our aim is to reduce the response time under the worst case, so that the response time of each user satisfies the realtime requirement.

[3]We consider a moderate area where the space containing geo-fences can be approximated by a 2D plane.

[4]It is easy to convert the latitude/longitude obtained from a GPS receiver to a local Cartesian coordinate. We also confirmed that latitude/longitude can be directly used for the INSIDE predicate, and used to approximately compute the distance for the WITHIN predicate.

Our contribution mainly lies in improving the performance of the refining stage of geo-fencing, and it can be used together with previous works suggested for the filtering-stage, eg., R-tree. Researchers tried to leverage the processing power of graphics processing unit (GPU) for the INSIDE test [Zhang and You 2012]. Our indexing-based implementation of the CN algorithm can also run on the GPU for a better performance.

3. PROPOSED ALGORITHM

Figure 1 shows our framework of geo-fencing, using mobile advertising as an example. The mobile phone of user A (e.g., a customer) periodically reports his ID and coordinate to a server. User B (e.g., a starbuck store) registers to the server. The server holds the tuples of <*semantic location*, *polygon*, *user list*>, where *semantic location* (e.g., a store, user B), *polygon* defining the range, and *user list* (user A, a customer ever visited the store) are provided by user B (the store). The server keeps tracking the coordinate of user A, and performs the INSIDE detection. If user A gets from OUTSIDE to INSIDE the polygon, a context aware processing will take place based on the setting, e.g., a message "User A gets near to the store" will be sent to user B. Then, user B can send a welcome message together with special discount information to attract user A. The range is periodically updated and its setting depends on the number of customers present in the store.

In typical geo-fencing applications, positions of points are changed much more frequently than those of polygons. Taking this into account, we try to build hash tables for polygons in the spare time of the system, and shift some computation cost from the refining stage (which has a realtime requirement) to the spare time. The cost of building hash tables is justified by the fact that hash tables, once built, can be used for many points.

Our framework is composed of two main parts, as follows:

— Polygon management. Polygons of geo-fences are stored in two structures with different details. MBRs of polygons, as rough approximations of polygons, are stored in an R-tree[5] [Guttman 1984], the same as in [Zhou et al. 2013; Li et al. 2013]. Edges of a polygon are stored in hash tables. To facilitate the update of polygons, each polygon has its own hash tables. Multiple hash tables are used in parallel. Inside a hash table, a large bucket is split into sub-buckets and sorted, and these buckets/sub-buckets are organized in a bucket tree.
— Pairing engine. In the filtering stage, R-tree is used to quickly detect whether a point is inside the MBR of any polygon. When a point is inside the MBR of a polygon, buckets in all tables associated with the point are found, and the one with fewest edges is selected. Then, a scheme corresponding to the hash policy is used to perform the CN algorithm (binary search if edges in this bucket are sorted and exhaustive search of all edges in this bucket otherwise), telling whether the point is really inside the polygon.

3.1. Bucket Selection

The basic edge-based hashing policy equally divides the x-coordinate range of a polygon, as shown in Fig. 2(a). The number of edges in each bucket, however, varies with buckets, e.g., B_{N-1} has more edges than B_{N-2}, and it will take more time to perform the CN algorithm for a point in B_{N-1} than another point in B_{N-2}. Accordingly, there might be a long tail in the distribution of the number of edges.

[5]R-tree is used as a simple example for the filtering stage. Other state-of-the-art methods can be used as well.

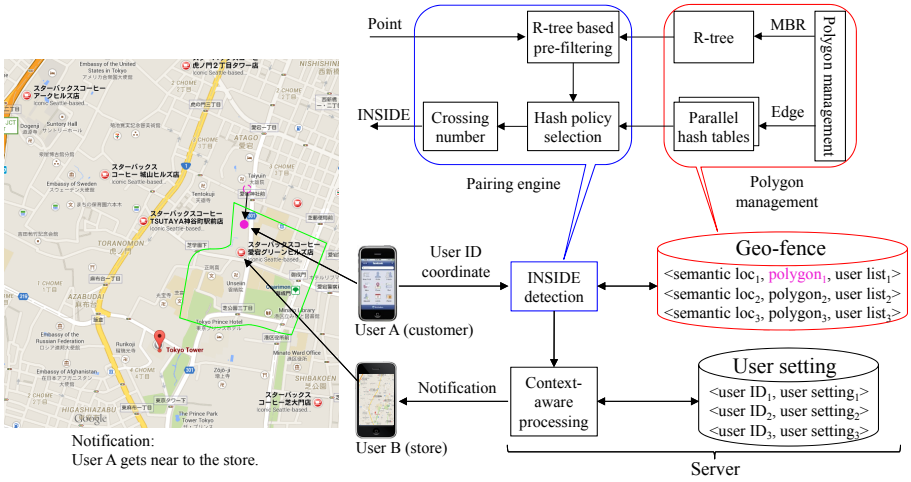
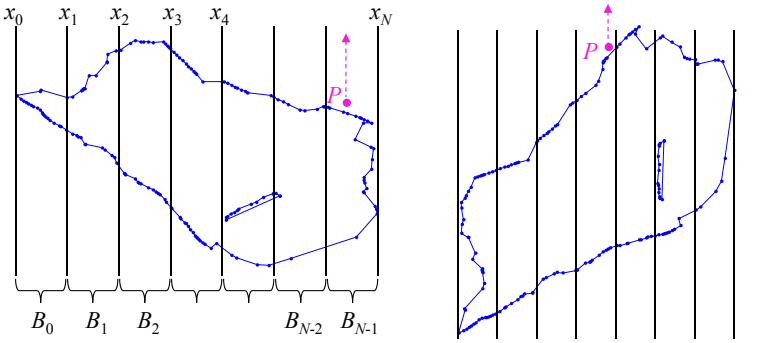


Fig. 1. System framework of geo-fencing.



(a) Organizing edges in buckets (b) Buckets in a different direction

Fig. 2. Effect of the ray direction in the crossing number algorithm.

In Fig. 2(a), bucket boundaries and the ray for crossing detection are parallel to the vertical axis. Actually, *the CN algorithm can be implemented in any direction*. Or alternatively, the ray and bucket boundaries are fixed to the vertical direction, but the polygon and point are rotated counter-clockwise by an angle α. With the same point P, organizing the edges of the polygon in buckets shown in Fig. 2(b) leads to fewer edges in the bucket associated with P.

We investigated the complementary cumulative distribution function (CCDF) of the number of edges per bucket on the dataset provided by GIS Cup'13, applying different rotation angles. The results are shown in Fig. 3, where '0 degree', '45 degree', '90 degree' and '135 degree' correspond to the cases of $\alpha = 0$, $\alpha = 45$, $\alpha = 90$, and $\alpha = 135$ degree, respectively. We also investigated the potential worst case (upper bound) and best case (lower bound) by exhaustively iterating rotation angles. According to Fig. 3, it is clear that there might be a long tail in the distribution of the number of edges compared with the lower bound. Though the lower bound of the distribution of edges does exist for each polygon, it is time consuming to rotate each polygon, not to mention the huge time taken to find the optimal rotation angle.

In the design of locality sensitive hashing [Indyk and Motwani 1998], usually multiple parallel hash tables are used. The similar idea can be leveraged here, though a

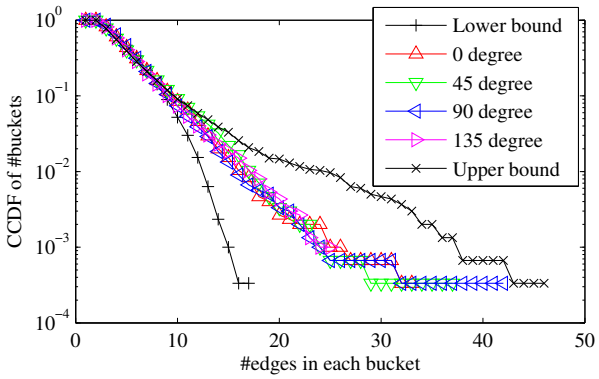


Fig. 3. CCDF of the number of edges per-bucket under different rotation angles of polygons (based on the dataset of GIS Cup'13).

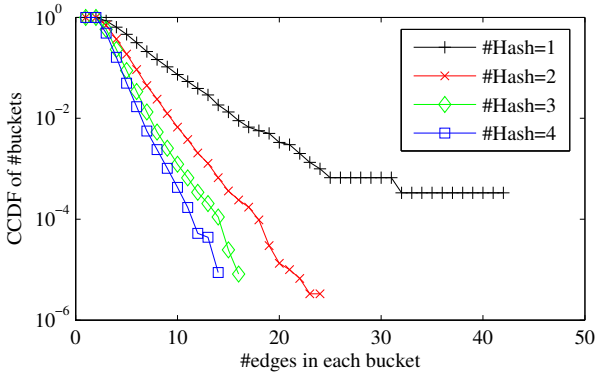


Fig. 4. CCDF of the number of edges examined per point under different combinations of rotation angles: 0 for '#Hash=1', 0 and 90 for '#Hash=2', 0, 60 and 120 for '#Hash=3', and 0, 45, 90, 135 for '#Hash=4' (based on the dataset of GIS Cup'13).

little differently. We construct for a polygon parallel hash tables, each with all edges. In this way, we realize the selection diversity [Neasmith and Beaulieu 1998] as follows: the most suitable bucket is found for each incoming point. Specifically, with a given point, one bucket is found from each table, and the bucket with fewest edges is used in the CN algorithm.

Figure 4 shows the distribution of the number of edges under bucket selection. The rotation angles of polygons are 0 for '#Hash=1', 0 and 90 for '#Hash=2', 0, 60 and 120 for '#Hash=3', and 0, 45, 90, 135 for '#Hash=4'. When there are more than 2 hash tables, the number of edges in the selected bucket depends on actual positions of points. Therefore, we divide the MBR of a polygon into grid points, and use them to obtain the distribution of the number of edges. As shown in Fig. 4, the number of edges examined for each point greatly decreases when there are more hash tables. The biggest gain occurs when the number of hash tables increases from 1 to 2, and the extra gain diminishes as more hash tables are used.

In comparison with the results in Fig. 4, we also investigated the optimal combinations of rotation angles by iterating all possible combinations. The results are shown in Fig. 5. There is a large difference between the two figures when there are only 1 or 2 hash tables. But the difference diminishes at 3 or 4 tables.

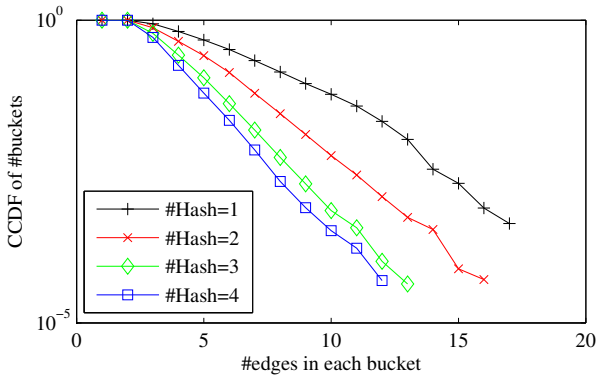


Fig. 5. CCDF of the number of edges examined per point under optimal combination of rotation angles for different numbers of hash tables (based on the dataset of GIS Cup'13).

Rotating a polygon by 0 degree means using the x coordinate of the polygon for hash design, while rotating 90 degree is equivalent to using the y coordinate. Finding the number of crossings in a direction other than horizontal or vertical directions, however, is time-consuming. Fortunately, the simple combination of 2 hash tables in the horizontal and vertical directions in Fig. 4 has satisfactory performance. We will exploit other methods to further enhance its performance.

3.2. Sorting Edges inside a Bucket

Using an interval index to organize edges in [Zhou et al. 2013] is equivalent to *non-equally* dividing the x coordinate range of a polygon. More specifically, bucket boundaries are determined by the vertices of a polygon, as shown in Fig. 6. We find that there are some good properties in such cases: (i) Any line passing a point associated with the bucket and parallel to the vertical axis crosses all edges in the bucket. For example, any vertical line in the range of B_5 crosses 4 edges. (ii) The relative relations between all crossing points, intersected by any vertical line in a bucket, are the same, regardless of the position of the crossing line. E.g., C_1, C_2, C_3, C_4 are the crossing points along the vertical line passing P_1. The relationship $C_1.y < C_2.y < C_3.y < C_4.y$ does not change with the position of P_1, if only P_1 is in the range of B_5. In this way, we can sort all edges in a bucket, using the y coordinate of crossing points along a vertical line. This is called SortEdge hereafter.

Organizing all edges in a bucket in an ordered list facilitates the crossing detection. For example, in Fig. 6, 4 edges split the strip area corresponding to B_5 into 5 regions $R_0, R_1, \cdots, R_4$. By comparing the coordinate $P_1.y$ against $C_1.y, C_2.y, \cdots, C_4.y$, we can find the index of the region where P_1 is located. This *region index* is nothing but the number of crossings by the ray starting from P_1 and along the negative vertical axis. Now, instead of examining all edges in the bucket, a binary search [Cormen et al. 2009] is sufficient to compute the region index of the point. However, it should be noted that the CN algorithm via binary search comes at the cost of building hash tables, as will be discussed later in Sec. 4.

3.3. Hybrid Hashing Policy

Constructing buckets by *equally* dividing x coordinate range of a polygon facilitates to locate buckets. But the effect of bucket selection is limited when there are only two tables, because there may be some blind areas where points match buckets both with many edges. On the other hand, constructing buckets by using vertices of polygons as

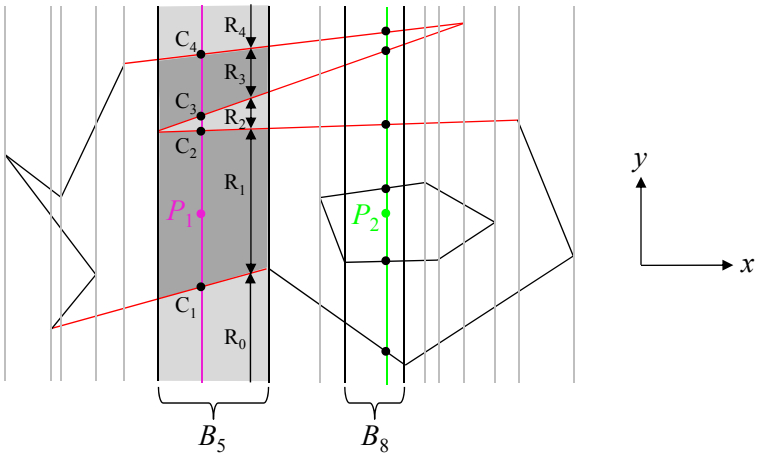


Fig. 6. Sorting edges inside a bucket according to the coordinates of crossing points along a vertical line.

bucket boundaries enables to sort all edges in a bucket, but it needs an extra index to locate a bucket. In addition, it is more time consuming to build the hash tables.

We adopt a hybrid policy to combine the two schemes as shown in Fig. 7. Generally, we equally divide the coordinate range of a polygon to construct buckets. For each bucket, if its number of edges is below a threshold, it is left as usual so as to both facilitate indexing and reduce the time of building hash tables; otherwise, the bucket is divided into sub-buckets where edges are sorted.

— Compared with the bucket selection only scheme, large buckets are split and sorted. This reduces the pairing time for a large bucket, at the cost of increased time of building hash tables.
— Compared with SortEdge, small buckets can still be directly located by their hash values, and the sorting is avoided.

In other words, there are two kinds of buckets, one is non-sorted buckets with few edges, and the other is sorted buckets with more edges. We use such a hybrid design so as to achieve a better tradeoff between pairing points with polygons and system overhead of building hash tables.

In the hybrid hashing scheme, polygons are organized by Algorithm 1. Basically, MBRs of polygons are stored in an R-tree (Lines 2-3). Then, with each rotation angle, a hash table is built for the polygon (Lines 4-6).

When building the kth hash table, the polygon is rotated counter-clockwise by an angle α_k. Then, parameters ($X_{min,j}, X_{max,j}$ corresponding to leftmost and rightmost points, and Δ_j corresponding to the bucket width) for the hash table are found (Lines 10-11). For each edge with two vertices V_1 and V_2, a range of hash values, $n_1, \cdots, n_2$, is determined by the two ends of this edge (Lines 13-14), and this edge is added to each bucket associated with a hash value between n_1 and n_2 (Line 15). After creating the basic hash table, a bucket B_n^k is assigned a type Regular (Line 19) if its number of edges is no more than a threshold. Otherwise, it is assigned a type Irregular (Line 21), and is further split into multiple sub-buckets B_{nl}^k (Lines 22-23) using vertices of edges as sub-bucket boundaries. Then, each edge in the bucket is added to the sub-buckets (Lines 24-27). Finally, edges in the same sub-bucket are sorted in terms of the y-coordinate of the crossing points along a vertical line.

The actual INSIDE detection is performed in a batch mode on a point set S with almost the same timestamp using Algorithm 2. Each of the M polygons has a separate

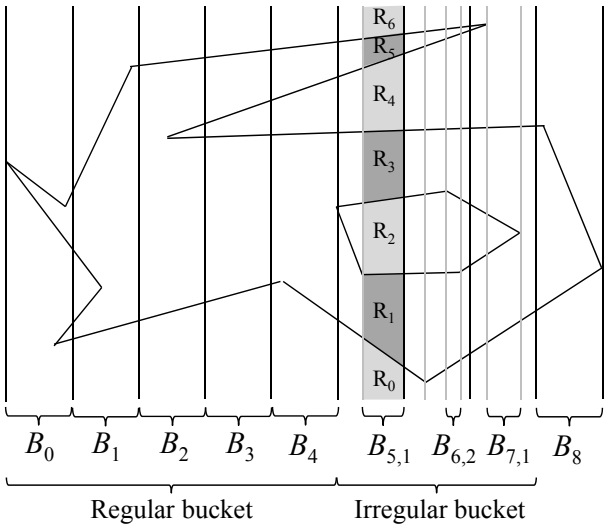


Fig. 7. Hybrid hash policy: most buckets have a regular width and can be directly located by hash values. Some big buckets are split into atomic sub-buckets with irregular width where edges are sorted.

buffer C_j. For each point $P \in \mathbf{S}$, its candidate polygons are found via R-tree and the point is added to the buffers of these polygons (Lines 2-6). Then, for each candidate pair $(P, Poly_j)$, the INSIDE detection is performed (Line 8), and a matched pair is exported (Lines 9-11).

In the hash-based CN algorithm (IsInside), first, a hash value for each hash table is computed based on the coordinates of P (Lines 15-18), and the associated buckets are $B^1_{n_1}, B^2_{n_2}, \cdots$. From these candidate buckets, the bucket $B^k_{n_k}$ with fewest edges is selected (Line 19), and there are two cases. (i) If this bucket is Regular, for all edges inside it, only the ones on top of P are counted (Lines 23-28). P is regarded as inside the polygon if the number of crossing is odd. (ii) If this bucket is Irregular, the sub-bucket actually holding P is found. In that sub-bucket, a binary search is used to find the region index of P. P is regarded as inside this polygon if its region index is odd.

The distributions of the number of edges examined per-point are studied under different numbers of hash tables. Here, we compare the basic bucket selection scheme (#Hash=n(Sel)) with fixed rotation angles, the optimal bucket selection scheme with refined rotation angles (#Hash=n(OptSel)), and the hybrid hashing scheme (#Hash=n(Hybrid)). The CCDF results are shown in Figs. 8, 9, 10, where n is equal to 2, 3, 4 respectively. The basic bucket selection is inferior to the optimal bucket selection, though their difference diminishes as more hash tables are used. The combination of bucket selection and sorting edges in the hybrid scheme always achieves the best performance.

3.4. Analysis of the Hashing Schemes

Here, we compare different hashing schemes: (1) Hash, the basic hashing scheme [Yu et al. 2013a], (2) MultiHash, multiple hash tables with bucket selection, as is discussed in Section 3.1, (3) SortEdge, sorting edges inside each bucket with an irregular bucket width, as is discussed in Section 3.2, and (4) Hybrid, the hybrid hashing scheme suggested in Section 3.3.

As shown in Table I, one table is used in the Hash and SortEdge schemes, and multiple hash tables are used in the MultiHash and Hybrid schemes. In the Hash scheme, the x coordinate range is equally divided to construct buckets and a bucket can be di-

Algorithm 1 Update $Poly_j$, the j^{th} polygon.

procedure UPDATEPOLYGON($Poly_j$)
  Get MBR $(X_{min,j}, X_{max,j}, Y_{min,j}, Y_{max,j})$ of $Poly_j$.
  Add this MBR to R-tree.
  for α_k in the set of rotation angles **do**
    BuildTable($Poly_j$, α_k). ▷ Build kth table.
  end for
end procedure
procedure BUILDTABLE($Poly_j$, α_k)
  Rotate $Poly_j$ counter-clockwise by α_k.
  Get $X_{min,j}, X_{max,j}$ of rotated polygon.
  $\Delta_j = (X_{max,j} - X_{min,j})/N$. ▷ Bucket width.
  for each edge (V_1, V_2) in $Poly_j$ **do**
    $n_1 = \text{GetHashKey}_k(V_1.x)$.
    $n_2 = \text{GetHashKey}_k(V_2.x)$.
    Add (V_1, V_2) to $B_n^k, n = n_1, \cdots, n_2$.
  end for
  for each bucket B_n^k **do**
    if $|B_n^k|$ is no more than a threshold **then**
      $B_n^k.type$ is set to Regular.
    else ▷ Split buckets.
      $B_n^k.type$ is set to Irregular.
      Get x coordinates of all vertices.
      Build sub-buckets B_{nl}^k.
      for each edge (V_1, V_2) in B_n^k **do**.
        Remove edge (V_1, V_2) from B_n^k.
        Add (V_1, V_2) to bucket $B_{nl}^k, l = l_1, \cdots, l_2$.
      end for
      Sort edges inside bucket B_{nl}^k.
    end if
  end for
end procedure

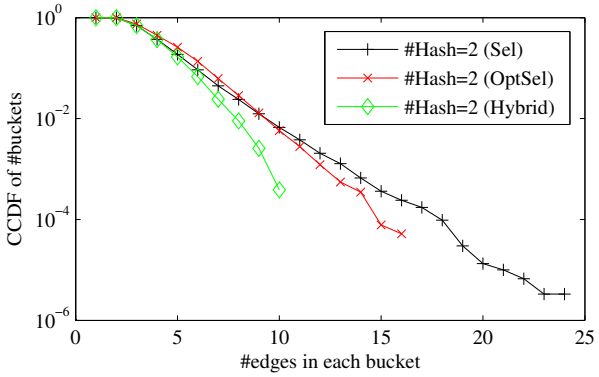


Fig. 8. CCDF of the number of edges examined per point by different hashing schemes (2 hash tables, based on the dataset of GIS Cup'13).

rectly located by its hash key. But within each bucket, all edges have to be examined exhaustively to perform the CN algorithm. In the MultiHash scheme, selecting the

Algorithm 2 INSIDE detection.

```
1: procedure INSIDE(Point set S)
2:     Clear candidate point set C_j, j = 1, ..., M.
3:     for each point P in S do
4:         Perform R-tree detection.
5:         Add P to C_j if P is in the MBR of Poly_j.
6:     end for
7:     for each point P in C_j, j = 1, ..., M do
8:         inPoly = IsInside(P, Poly_j).
9:         if inPoly is true then
10:            Export (P.ID, P.time, Poly_j.ID, Poly_j.time).
11:        end if
12:    end for
13: end procedure
14: procedure ISINSIDE(P,Poly_j)
15:    for α_k in the set of rotation angles do
16:        Rotate point P counter-clockwise by α_k.
17:        n_k = GetHashKey_k(P.x).
18:    end for
19:    Select a bucket k = argmin_k |B^k_{n_k}|.
20:    if B^k_{n_k}.type is Regular then
21:        cross = 0.
22:        for each edge (V_1, V_2) in bucket B^k_{n_k} do
23:            if V1.x ≤ P.x ≤ V2.x then
24:                s = (V_2.y − V_1.y)/(V_2.x − V_1.x).
25:                dy = (P.y − V_1.y) − s · (P.x − V_1.x).
26:                if (dy < 0) then cross = cross + 1.
27:                end if
28:            end if
29:        end for
30:        inPoly=cross is odd.
31:    else
32:        Find the sub-bucket B^k_{n_k l}.
33:        Perform binary search to find region index of P.
34:        inPoly = region index of P is odd.
35:    end if
36: end procedure
```

bucket with fewest edges reduces the cost of the CN algorithm to the minimal number of edges ($min(\#edge)$). In the SortEdge scheme, locating a bucket depends on the interval index, and the computation cost is the log value ($\log_2(\#vertex)$) of the number of vertices in a polygon. Performing the CN algorithm via the binary search on the sorted edges reduces the cost to the log value of the number of edges in a bucket ($\log_2(\#edge)$). The Hybrid scheme combines MultiHash and SortEdge, and in the ideal case, the number of examined edges can be reduced to $\log_2(min(\#edge)) = min(\log_2(\#edge))$, which is actually a lower bound.

It should be noticed that reducing the time of the CN algorithm is at the cost of increasing the time of building hash tables. When building hash tables, the Hash scheme takes a short time while the SortEdge takes a long time to order edges inside each bucket. Multiple hash tables are constructed in the MultiHash scheme, and

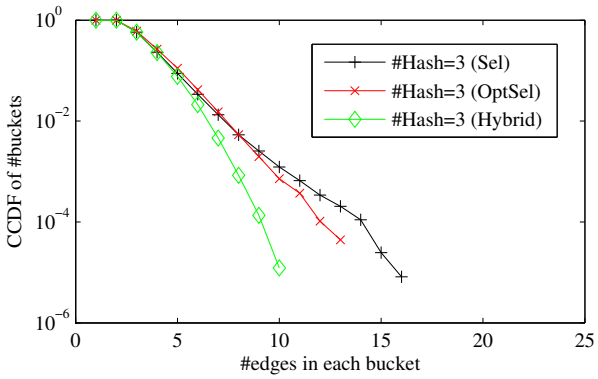


Fig. 9. CCDF of the number of edges examined per point by different hashing schemes (3 hash tables, based on the dataset of GIS Cup'13).

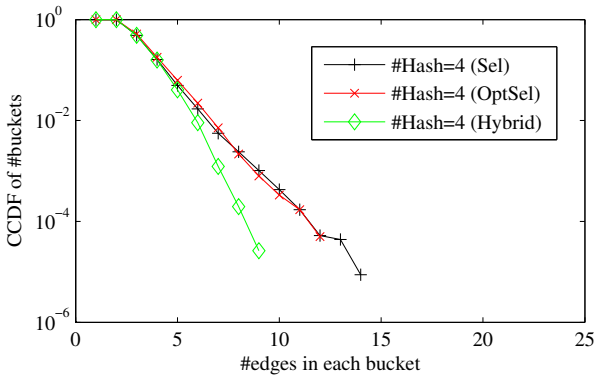


Fig. 10. CCDF of the number of edges examined per point by different hashing schemes (4 hash tables, based on the dataset of GIS Cup'13).

Table I. Properties of different hashing schemes for the INSIDE detection.

	Tables	Building tables	Locating bucket	CN algorithm
Hash	1	Short	1	#edge
MultiHash	#table	Medium	#table	min (#edge)
SortEdge	1	Long	$\log_2$(#vertex)	$\log_2$(#edge)
Hybrid	#table	Medium	#table	min($\log_2$(#edge))

some of the buckets are further split and sorted in the Hybrid scheme. Therefore, the two schemes take medium time to build the hash tables. We will further discuss this in the experiment evaluation.

3.4.1. Effect of bucket selection. Assume the probability density function (PDF) of the number of edges (X) in buckets is $f_X(x)$ for one hash table (the Hash scheme), and its cumulative distribution function is $F_X(x)$. Using n parallel hash tables and bucket selection in the MultiHash scheme, the number of examined edges is equal to $\hat{X} = \min(X_1, \cdots, X_n)$. When $X_i, i = 1, \cdots, n$ are independent and identically distributed random variables, $\hat{X}$ has a PDF

$$\hat{f}_{\hat{X}}(x) = n f_X(x)(1 - F_X(x))^{n-1}, \quad (1)$$

according to the order statistics [David and Nagaraja 2003]. Compared with $f_X(x)$, $\hat{f}_{\hat{X}}(x)$ has a gain

$$g(x) = n \cdot (1 - F_X(x))^{n-1}. \tag{2}$$

At small x, $F_X(x)$ approaches 0 and $g(x)$ approaches n. On the other hand, at large x, $F_X(x)$ approaches 1 and $g(x)$ approaches 0. Therefore, increasing the number of hash tables to n amplifies the probability at small x but decreases the probability at large x. This effectively decreases the tail of the distribution of the number of edges examined per point.

Let $E(.)$ be the expectation operation. With some specific distributions, both $E(X)$ and $E(\hat{X})$ can be computed and we can learn the effect of bucket selection. Here, we consider three typical distributions: exponential distribution (severe unbalance among buckets), linear distribution (moderate unbalance among bucket), and uniform distribution (no-unbalance).

In the case of exponential distribution, $f_{X,e}(x) = \exp(-\lambda x), x \geq 0$ with a parameter λ. Then, we have $E_e = E(X) = 1/\lambda$, $F_{X,e}(x) = 1 - \exp(-\lambda x)$ and $Pr(X > x) = \exp(-\lambda x)$. The distribution of $\hat{X} = \min(X)$ has a simple form, as follows:

$$Pr(\hat{X} > x) = Pr(X_1 > x, \cdots, X_n > x) = \prod_{i=1}^{n} \exp(-\lambda x) = \exp\left(-(n\lambda) \cdot x\right). \tag{3}$$

In this way, $\hat{X}$ is also an exponential distribution, but with an expectation $\frac{1}{n\lambda} = E_e \cdot \frac{1}{n}$.

In the case of linear distribution, $f_{X,l}(x) = \frac{2}{L}\left(1 - \frac{x}{L}\right), 0 \leq x \leq L$. Then, we have $E_l = E(X) = \frac{L}{3}$ and $F_{X,l}(x) = \frac{2x}{L} - \frac{x^2}{L^2}$. Accordingly, $\hat{X} = \min(X)$ has a distribution

$$\hat{f}_{\hat{X},l}(x) = n \cdot \frac{2}{L}\left(1 - \frac{x}{L}\right) \cdot \left(1 - \frac{2x}{L} + \frac{x^2}{L^2}\right)^{n-1} = \frac{2n}{L}\left(1 - \frac{x}{L}\right)^{2n-1}, 0 \leq x \leq L, \tag{4}$$

and its expectation is

$$\int_0^L x \cdot \frac{2n}{L}\left(1 - \frac{x}{L}\right)^{2n-1} dx = 2nL \int_0^1 x(1-x)^{2n-1} dx = \frac{L}{2n+1} = E_l \cdot \frac{3}{2n+1}. \tag{5}$$

In the case of uniform distribution, $f_{X,u}(x) = \frac{1}{L}, 0 \leq x \leq L$, $E_u = E(X) = \frac{L}{2}$, and $F_{X,u}(x) = \frac{x}{L}$. $\hat{X} = \min(X)$ has a distribution

$$\hat{f}_{\hat{X}}(x) = \frac{n}{L}\left(1 - \frac{x}{L}\right)^{n-1}, 0 \leq x \leq L, \tag{6}$$

with an expectation

$$\int_0^L x \cdot \frac{n}{L}\left(1 - \frac{x}{L}\right)^{n-1} dx = nL \int_0^1 x(1-x)^{n-1} dx = \frac{L}{n+1} = E_u \cdot \frac{2}{n+1}. \tag{7}$$

The ratio $E(X)/E(\hat{X})$ is defined as the average gain of bucket selection. This average gain is equal to n for exponential distribution, $(2n+1)/3$ for linear distribution, $(n+1)/2$ for uniform distribution. It increases with n, and gets larger as the distribution gets closer to exponential (the unbalance among buckets gets more severe).

The above analysis is based on the assumption of independence. Actually, the number of edges inside buckets in parallel hash tables may be correlated, because the difference between rotation angles of polygons decreases at large n. This correlation will degrade the average gain of bucket selection. On the other hand, two hash tables can be efficiently implemented by leveraging the x and y coordinates, but a larger n

requires to rotate polygons. Therefore, n is chosen to be 2 as a tradeoff between the average gain of bucket selection and system overhead of building hash tables.

3.4.2. Effect of sorting edges. In the ideal Hybrid scheme, where edges are sorted in all buckets, the number of examined edges is equal to $\overline{X} = \log_2(\hat{X})$, and its distribution is

$$\overline{f}_{\overline{X}}(x) = \hat{f}_{\hat{X}}(2^x) \cdot 2^x \log 2. \quad (8)$$

Compared with $\hat{f}_{\hat{X}}(x)$, the curve of $\overline{f}_{\overline{X}}(x)$ is compressed along the horizontal axis and amplified in the vertical direction. This effectively removes the long tail of the distribution of the number of edges.

4. EXPERIMENTAL RESULTS

We used two datasets in the evaluation. One is provided by the ACM GIS Cup 2013[6], which includes two point files (Point500 with 39,289 instances of 500 points, Point1000 with 69,619 instances of 1000 points), two polygon files (Poly10 with 30 instances of 10 polygons, Poly15 with 40 instances of 15 polygons), and the ground truth of the INSIDE detection under different combinations of inputs. Points constantly change their positions and have different instances. The sequence number for each instance of point or polygon is used as the timestamp. The distances between adjacent instances of a point vary with time and are different for points. The distances have an average 810.0m and a standard deviation 755.2m for Point500, and an average 862.4m and a standard deviation 801.9m for Point1000. The other dataset is built from the data downloaded from OpenStreetMap[7], which contains polygons for all land areas in the world, i.e., continents and islands. We constructed 4 polygon files Poly-OSM1 to Poly-OSM4, each with 200 instances of 20 polygons. On this basis, we created point files (Point-OSM1 to Point-OSM4, each with 80,000 instances) by randomly selecting points from the MBR area of each polygon. We notice that a real system may contain a very large number of geo-fences. It is possible to apply our method to such a system in a distributed way as follows: Geo-fences are divided into groups based on their geographic areas, and each group of geo-fences can be processed by a separate server running our method.

In the evaluation, we compare five methods: Base (the basic scheme exploiting R-Tree but no hash for the INSIDE detection), Hash, MultiHash, SortEdge, and Hybrid. All algorithms are programmed by using Visual Studio Professional C++ 2012. MultiHash and Hybrid are implemented with two tables, directly using x (rotation angle = 0) and y (rotation angle = 90 degree) coordinates.

We conducted evaluations using a desktop PC with Intel Core i7 CPU (3.4GHz) and 64-bit Windows 7. In all experiments, 100% accuracy is achieved. The proposed scheme shifts some computation cost from the refining stage to system spare time, where hash tables are built for polygons. To illustrate the cost in different stages, we separately evaluated (i) the time taken to update polygons (converting coordinates from Geography Markup Language format, updating the R-tree of MBRs, and building hash tables), (ii) the time taken in the filtering stage, and (iii) the time taken in the refining stage by the CN algorithm. As is difficult to accurately measure the per-point pairing time, the total time for pairing a point file with a polygon file is used instead. The execution time is measured by the function QueryPerformanceCounter[8] in the Windows environment. The experiment is repeated 10,000 times. On this basis, we compute the average and standard deviation of execution times and their distributions. The storage

[6]http://dmlab.cs.umn.edu/GISCUP2013/downloads.php
[7]http://openstreetmapdata.com/data/land-polygons
[8]http://msdn.microsoft.com/en-us/library/windows/desktop/ms644904(v=vs.85).aspx

Table II. Total time consumed in pairing Point500 and Poly10 (time unit: ms).

	Updating polygon		Filtering (R-Tree)		Refining (CN algorithm)		Storage
	Average	Std	Average	Std	Average	Std	Average
Base	1.378	0.073	1.002	0.026	8.451	0.508	1
Hash	2.108	0.128	1.001	0.023	1.280	0.057	2.19
MultiHash	2.837	0.096	1.001	0.026	1.186	0.079	4.31
SortEdge	8.031	0.311	1.005	0.029	1.218	0.075	3.35
Hybrid	5.022	0.320	1.009	0.099	1.141	0.080	5.44

Table III. Total time consumed in pairing Point1000 and Poly10 (time unit: ms).

	Updating polygon		Filtering (R-Tree)		Refining (CN algorithm)		Storage
	Average	Std	Average	Std	Average	Std	Average
Base	1.361	0.087	1.641	0.096	12.891	0.653	1
Hash	2.093	0.121	1.623	0.077	2.020	0.073	2.19
MultiHash	2.836	0.119	1.615	0.082	1.849	0.118	4.31
SortEdge	8.040	0.307	1.723	0.092	1.922	0.116	3.35
Hybrid	5.045	0.173	1.620	0.080	1.812	0.098	5.44

of each scheme is also evaluated, by measuring how many times an edge is stored in the hash tables.

4.1. Results on the Dataset of GIS Cup'13

Results under different combinations of points and polygon files are shown in Tables II–V, and the components of computation costs are listed below:

— Updating polygons in system spare time. In the Base scheme, there is no hash table, the time of updating polygons is least. As for each of the other schemes, the average time for updating polygons is almost the same for Tables II,III, and for Tables IV,V. In the SortEdge scheme, sorting edges in all buckets takes much more time than other schemes. In comparison, Hybrid takes medium time by only sorting edges in some of the buckets.
— Filtering stage. The filtering stage is the same for all schemes, so is the average time.
— Refining stage. As for the average time taken for the CN algorithm, the Base algorithm is most time consuming, about 10ms. Using indexing in Hash, MultiHash, SortEdge, and Hybrid reduces this time to about 1 to 2ms. The Hybrid scheme takes the least time for the CN algorithm. Here, the SortEdge scheme is more time consuming than Hybrid, because it takes time to locate a bucket with an interval index, even when there are few edges in the bucket. Although there is no big difference between Hash and other indexing schemes (MultiHash, SortEdge and Hybrid) in the total time of CN, their worst case time is quite different, as reflected in Figs. 4,8.

The numbers of pairs in the filtering and refining stage are listed in Table VI. The time taken for each pair is approximately the total time divided by the number of pairs, for example, the filtering time and refining time per-pair for joining Point500 with Poly15 via Hybrid are equal to 1.134ms/589335 and 1.292ms/19420, respectively. Although only about 3% of pairs remain in the refining stage, the refining stage of Base takes more time than the filtering stage, because each polygon has hundreds of edges and the plain CN algorithm is time-consuming. Using hash schemes reduces the time in the refining stage to a value comparable to that of the filtering stage.

All the hashing schemes reduce the time of the CN algorithm at the cost of increased storage, because each edge may be repeatedly stored in several adjacent buckets. Here, each edge is stored once in the Base scheme, nearly 2.2 in Hash, 4.3 in MultiHash, 3.4 in SortEdge, and 5.5 in Hybrid.

The CCDF results corresponding to Table IV are shown in detail in Fig. 11 (Part of the left plot is amplified in the right side) and Fig. 12 respectively. The difference in

Table IV. Total time consumed in pairing Point500 and Poly15 (time unit: ms).

	Updating polygon		Filtering (R-Tree)		Refining (CN algorithm)		Storage
	Average	Std	Average	Std	Average	Std	Average
Base	1.783	0.035	1.150	0.057	9.002	0.308	1
Hash	2.814	0.085	1.131	0.049	1.430	0.069	2.20
MultiHash	3.601	0.196	1.129	0.055	1.419	0.069	4.33
SortEdge	10.178	0.516	1.178	0.073	1.412	0.052	3.45
Hybrid	6.391	0.311	1.134	0.053	1.292	0.091	5.56

Table V. Total time consumed in pairing Point1000 and Poly15 (time unit: ms).

	Updating polygon		Filtering (R-Tree)		Refining (CN algorithm)		Storage
	Average	Std	Average	Std	Average	Std	Average
Base	1.785	0.053	2.115	0.128	15.867	0.272	1
Hash	2.824	0.087	2.038	0.097	2.532	0.081	2.20
MultiHash	3.628	0.180	2.030	0.090	2.346	0.148	4.33
SortEdge	10.354	0.627	2.241	0.040	2.359	0.148	3.45
Hybrid	6.469	0.346	2.044	0.104	2.280	0.127	5.56

Table VI. Number of pairs in the filtering stage (join between point and MBR) and refining stage (join between point and polygon).

	Point500, Poly10	Point500, Poly15	Point1000, Poly10	Point1000, Poly15
Filtering	392890	589335	696190	1044285
Refining	16858	19420	26398	32754

Table VII. Total time consumed in pairing Point-OSM1 and Poly-OSM1 (time unit: ms).

	Updating polygon		Filtering (R-Tree)		Refining (CN algorithm)		Storage
	Average	Std	Average	Std	Average	Std	Average
Base	11.249	0.239	2.288	0.108	14.297	0.537	1
Hash	15.894	0.343	2.256	0.069	3.201	0.118	1.17
MultiHash	20.050	0.522	2.257	0.071	2.250	0.056	2.29
SortEdge	72.275	3.261	2.279	0.097	1.531	0.121	4.13
Hybrid	40.943	2.212	2.260	0.086	1.802	0.099	4.42

the total time of the CN algorithm between the four hashing schemes is not very large. In contrast, there is a clear distinction in the time of building hash tables in Fig. 12. Only few buckets are sorted in the Hybrid scheme. Therefore, its time of building hash tables is between those of MultiHash and SortEdge.

4.2. Results on the Dataset of OpenStreetMap

We also did experiments on the dataset constructed from the data of OpenStreetMap. Results under different combinations of points and polygon files are shown in Tables VII–X. These results show a similar trend as those in Tables II–V, although the time gets larger due to more polygons and points involved in the computation.

There are also several minor differences here: (i) As for the Hash scheme, the per-edge storage is only around 1.2 in Tables VII–X, compared with 2.2 in Tables II–V. This indicates that the two datasets have different distributions of the number of edges per-bucket, and more edges tend to be located in fewer buckets in the second dataset. As a result, the performance gap between MultiHash and Hash, due to bucket selection, gets larger on the second dataset than the first one. (ii) SortEdge achieves the least time of the CN algorithm. This can be explained as follows: when there are many edges in buckets, the CN algorithm via binary search in a bucket reduces more time than the increased time taken to locate a bucket. Under most cases, Hybrid achieves similar performance as SortEdge in terms of the time for the CN algorithm, but with much less time for building hash tables.

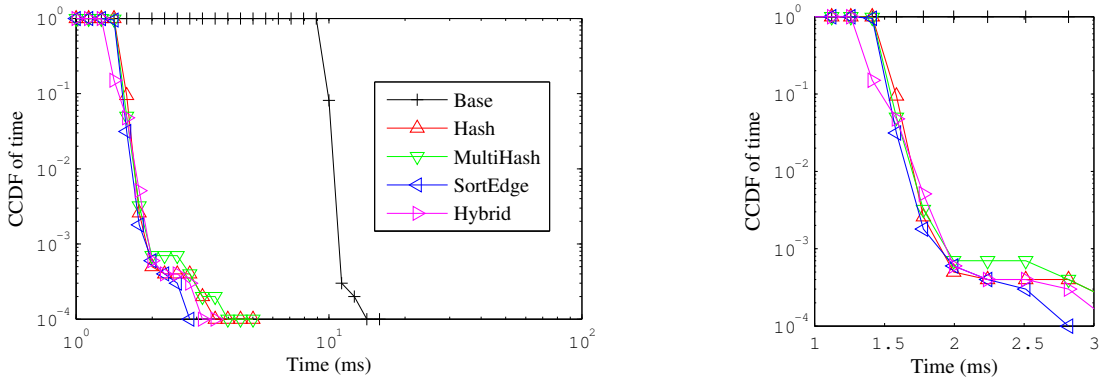


Fig. 11. CCDF of the time taken for the CN algorithm in different hashing schemes (based on the dataset of GIS Cup'13).

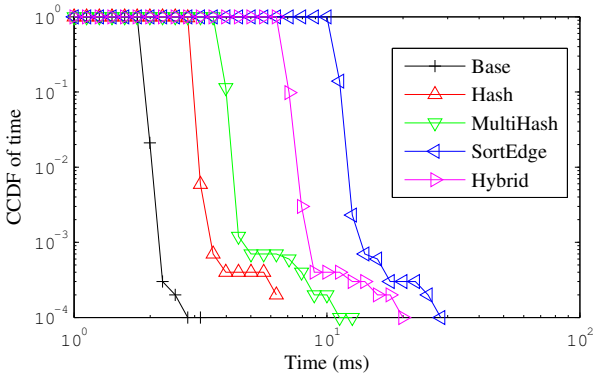


Fig. 12. CCDF of the time taken for building hash tables in different hashing schemes (based on the dataset of GIS Cup'13).

Table VIII. Total time consumed in pairing Point-OSM2 and Poly-OSM2 (time unit: ms).

	Updating polygon		Filtering (R-Tree)		Refining (CN algorithm)		Storage
	Average	Std	Average	Std	Average	Std	Average
Base	11.460	0.537	2.413	0.134	14.896	0.868	1
Hash	15.972	0.481	2.264	0.081	3.694	0.206	1.18
MultiHash	20.163	0.697	2.264	0.081	2.341	0.139	2.31
SortEdge	80.355	2.844	2.356	0.136	1.614	0.090	4.89
Hybrid	45.587	2.152	2.266	0.086	1.817	0.091	4.93

Table IX. Total time consumed in pairing Point-OSM3 and Poly-OSM3 (time unit: ms).

	Updating polygon		Filtering (R-Tree)		Refining (CN algorithm)		Storage
	Average	Std	Average	Std	Average	Std	Average
Base	11.236	0.228	2.307	0.123	14.267	0.515	1
Hash	15.888	0.276	2.261	0.075	2.982	0.174	1.28
MultiHash	20.126	0.651	2.260	0.076	2.079	0.125	2.45
SortEdge	79.683	1.702	2.295	0.111	1.636	0.124	4.84
Hybrid	39.839	1.223	2.255	0.071	2.028	0.090	4.62

The CCDF results corresponding to Table VIII are shown in detail in Fig. 13 and Fig. 14 respectively. The difference in the total time of the CN algorithm between the four hashing schemes gets clear in this case, and the performance of Hybrid ap-

Table X. Total time consumed in pairing Point-OSM4 and Poly-OSM4 (time unit: ms).

	Updating polygon		Filtering (R-Tree)		Refining (CN algorithm)		Storage
	Average	Std	Average	Std	Average	Std	Average
Base	11.528	0.614	2.273	0.100	15.871	0.323	1
Hash	16.070	0.658	2.248	0.057	3.169	0.062	1.23
MultiHash	20.370	0.940	2.249	0.055	2.065	0.116	2.40
SortEdge	80.336	2.817	2.267	0.084	1.622	0.103	4.39
Hybrid	42.392	2.641	2.247	0.049	1.920	0.131	4.53

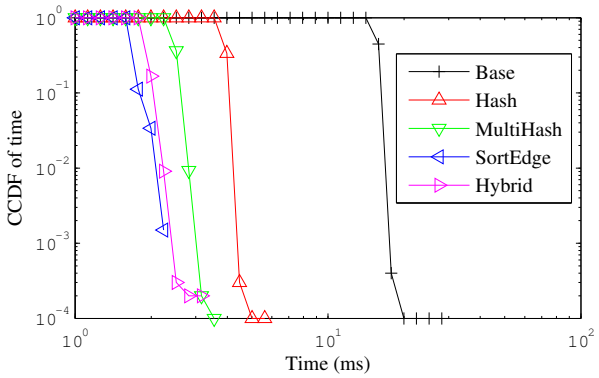


Fig. 13. CCDF of the time taken for the CN algorithm in different hashing schemes (based on the dataset of OpenStreetMap).

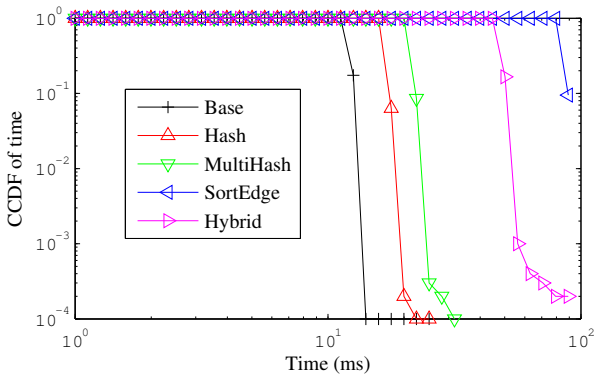


Fig. 14. CCDF of the time taken for building hash tables in different hashing schemes (based on the dataset of OpenStreetMap).

proaches that of SortEdge, but Hybrid takes much less time for updating the hash tables. To better understand the performance difference on the two datasets, we also show the distribution of the number of edges examined per point on the second dataset in Fig. 15. There are more edges in each buckets compared with Fig. 4, and this explains why sorting edges (SortEdge) is more effective than bucket selection (MultiHash) on the second dataset.

Based on the two datasets, we learn that *the refining stage (the CN algorithm) can be accelerated at the cost of increased system overhead (e.g., building hash tables for polygons) in the spare time*, and the cost of more storage. SortEdge and Hybrid are good choices when polygons in the system are seldom updated. On the other hand, when polygons are frequently updated, MultiHash and Hybrid will be better. In general, *Hybrid achieves a better tradeoff between realtime pairing of points and polygons (by*

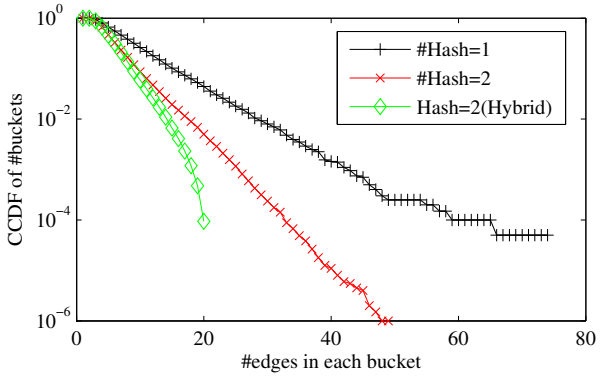


Fig. 15. CCDF of the number of edges examined per point in different hashing schemes (based on the dataset of OpenStreetMap).

accelerating the CN algorithm) and system overhead (building hash tables), compared with the other hashing schemes. Although the performance of bucket selection and sorting edges varies with the dataset, combining the two in Hybrid makes it possible to achieve good performance on both datasets, and *effectively reduce the worst case response time*. These schemes are proposed for geo-fencing applications with complex polygons, where positions of points are changed much more frequently than those of polygons. This justifies the system overhead of building hash tables for polygons. When polygons are simple (e.g., the number of edges is below a threshold) or polygons change at a frequency comparable to that of points, it might be better to directly perform the plain CN algorithm. When there are both simple and complex polygons in the database, a flag can be assigned to each polygon, indicating whether indexing is used for the polygon.

5. CONCLUSION

Based on pairing points with polygons, a system supporting geo-fencing triggers an action when users enter or leave predefined geo-fences. The efficient implementation of geo-fencing decides the response performance of large-scale realtime applications. In this paper, we studied the distribution of the number of edges examined per point and the properties of edges inside a bucket. On this basis, we suggested a hybrid hashing policy to organize edges of polygons in buckets, and an efficient implementation of the crossing number algorithm which combines bucket selection and in-bucket binary search. Extensive analyses and evaluations on real-world datasets confirm that the proposed algorithm helps effectively (1) reduce the pairing cost in terms of both average time and the distribution, and (2) achieve a better tradeoff between realtime pairing points with polygons and system overhead. In the future, we will construct larger databases to perform large-scale evaluations.

REFERENCES

Matthias Baldauf, Schahram Dustdar, and Florian Rosenberg. 2007. A Survey on Context-Aware Systems. *Int. J. Ad Hoc Ubiquitous Comput.* 2, 4 (June 2007), 263–277. DOI:http://dx.doi.org/10.1504/IJAHUC.2007.014070

Ulrich Bareth, Axel Küpper, and Peter Ruppel. 2010. geoXmart - A Marketplace for Geofence-Based Mobile Services. In *Computer Software and Applications Conference (COMPSAC), 2010 IEEE 34th Annual*. 101–106. DOI:http://dx.doi.org/10.1109/COMPSAC.2010.16

Muhammad Aamir Cheema, Ljiljana Brankovic, Xuemin Lin, Wenjie Zhang, and Wei Wang. 2011. Continuous Monitoring of Distance-Based Range Queries. *Knowledge and Data Engineering, IEEE Transactions on* 23, 8 (Aug 2011), 1182–1199. DOI:http://dx.doi.org/10.1109/TKDE.2010.246

Krishna Chintalapudi, Anand Padmanabha Iyer, and Venkata N. Padmanabhan. 2010. Indoor Localization Without the Pain. In *Proceedings of the Sixteenth Annual International Conference on Mobile Computing and Networking (MobiCom '10)*. ACM, New York, NY, USA, 173–184. DOI:http://dx.doi.org/10.1145/1859995.1860016

Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. 2009. *Introduction to Algorithms (3rd Edition)*. The MIT Press.

Herbert A. David and H. N. Nagaraja. 2003. *Order Statistics (3rd Edition)*. Wiley.

Ralf Hartmut Güting and Markus Schneider. 2005. *Moving Objects Databases*. Morgan Kaufmann.

Antonin Guttman. 1984. R-trees: A Dynamic Index Structure for Spatial Searching. In *Proceedings of the 1984 ACM SIGMOD International Conference on Management of Data (SIGMOD '84)*. ACM, New York, NY, USA, 47–57. DOI:http://dx.doi.org/10.1145/602259.602266

Kai Hormann and Alexander Agathos. 2001. The Point in Polygon Problem for Arbitrary Polygons. *Comput. Geom. Theory Appl.* 20, 3 (Nov. 2001), 131–144. DOI:http://dx.doi.org/10.1016/S0925-7721(01)00012-8

Sergio Ilarri, Eduardo Mena, and Arantza Illarramendi. 2010. Location-dependent Query Processing: Where We Are and Where We Are Heading. *ACM Comput. Surv.* 42, 3, Article 12 (March 2010), 73 pages. DOI:http://dx.doi.org/10.1145/1670679.1670682

Piotr Indyk and Rajeev Motwani. 1998. Approximate Nearest Neighbors: Towards Removing the Curse of Dimensionality. In *Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing (STOC '98)*. ACM, New York, NY, USA, 604–613. DOI:http://dx.doi.org/10.1145/276698.276876

Edwin H. Jacox and Hanan Samet. 2007. Spatial Join Techniques. *ACM Trans. Database Syst.* 32, 1, Article 7 (March 2007). DOI:http://dx.doi.org/10.1145/1206049.1206056

Axel Küpper, Ulrich Bareth, and Behrend Freese. 2011. Geofencing and Background Tracking - The Next Features in LBS. In *INFORMATIK'11*.

Suikai Li, Weiwei Sun, Renchu Song, Zhangqing Shan, Zheyong Chen, and Xinyu Zhang. 2013. Quick Geo-fencing Using Trajectory Partitioning and Boundary Simplification. In *Proceedings of the 21st ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems (SIGSPATIAL'13)*. ACM, New York, NY, USA, 580–583. DOI:http://dx.doi.org/10.1145/2525314.2527265

David Martin, Aurkene Alzua, and Carlos Lamsfus. 2011. A Contextual Geofencing Mobile Tourism Service. In *Information and Communication Technologies in Tourism 2011*, Rob Law, Matthias Fuchs, and Francesco Ricci (Eds.). Springer Vienna, 191–202. DOI:http://dx.doi.org/10.1007/978-3-7091-0503-0_16

Pratap Misra and Per Enge. 2010. *Global Positioning System: Signals, Measurements, and Performance (Revised 2nd Edition)*. Ganga-Jamuna Press.

Mohamed F. Mokbel, Xiaopeng Xiong, Moustafa A. Hammad, and Walid G. Aref. 2005. Continuous Query Processing of Spatio-Temporal Data Streams in PLACE. *GeoInformatica* 9, 4 (2005), 343–365. DOI:http://dx.doi.org/10.1007/s10707-005-4576-7

Elisabeth A. Neasmith and Norman C. Beaulieu. 1998. New results on selection diversity. *Communications, IEEE Transactions on* 46, 5 (May 1998), 695–704.

Sadegh Nobari, Farhan Tauheed, Thomas Heinis, Panagiotis Karras, Stephane Bressan, and Anastasia Ailamaki. 2013. TOUCH: In-memory Spatial Join by Hierarchical Data-oriented Partitioning. In *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data (SIGMOD '13)*. ACM, New York, NY, USA, 701–712. DOI:http://dx.doi.org/10.1145/2463676.2463700

Jignesh M. Patel, Yun Chen, and V. Prasad Chakka. 2004. STRIPES: An Efficient Index for Predicted Trajectories. In *Proceedings of the 2004 ACM SIGMOD International Conference on Management of Data (SIGMOD '04)*. ACM, New York, NY, USA, 635–646. DOI:http://dx.doi.org/10.1145/1007568.1007639

Siripen Pongpaichet, Vivek K. Singh, Ramesh Jain, and Alex (Sandy) Pentland. 2013. Situation Fencing: Making Geo-fencing Personal and Dynamic. In *Proceedings of the 1st ACM International Workshop on Personal Data Meets Distributed Multimedia (PDM '13)*. ACM, New York, NY, USA, 3–10. DOI:http://dx.doi.org/10.1145/2509352.2509401

Sunil Prabhakar, Yuni Xia, Dmitri V. Kalashnikov, Walid G. Aref, and Susanne E. Hambrusch. 2002. Query indexing and velocity constrained indexing: scalable techniques for continuous queries on moving objects. *Computers, IEEE Transactions on* 51, 10 (Oct 2002), 1124–1140. DOI:http://dx.doi.org/10.1109/TC.2002.1039840

Franco P. Preparata and Michael I. Shamos. 1985. *Computational Geometry: An Introduction*. Springer-Verlag New York, Inc., New York, NY, USA.

Siva Ravada, Mohamed Ali, Jie Bao, and Mohamed Sarwat. 2013. ACM SIGSPATIAL GIS Cup 2013: Geo-fencing. In *Proceedings of the 21st ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems (SIGSPATIAL'13)*. ACM, New York, NY, USA, 584–587. DOI:http://dx.doi.org/10.1145/2525314.2527266

Fabrice Reclus and Kristen Drouard. 2009. Geofencing for fleet & freight management. In *Intelligent Transport Systems Telecommunications,(ITST),2009 9th International Conference on*. 353–356. DOI:http://dx.doi.org/10.1109/ITST.2009.5399328

Simonas Saltenis, Christian S. Jensen, Scott T. Leutenegger, and Mario A. Lopez. 2000. Indexing the Positions of Continuously Moving Objects. In *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data (SIGMOD '00)*. ACM, New York, NY, USA, 331–342. DOI:http://dx.doi.org/10.1145/342009.335427

M. Shimrat. 1962. Algorithm 112: Position of Point Relative to Polygon. *Commun. ACM* 5, 8 (Aug. 1962), 434–. DOI:http://dx.doi.org/10.1145/368637.368653

Benjamin Sowell, Marcos Vaz Salles, Tuan Cao, Alan Demers, and Johannes Gehrke. 2013. An Experimental Analysis of Iterated Spatial Joins in Main Memory. *Proc. VLDB Endow.* 6, 14 (Sept. 2013), 1882–1893. DOI:http://dx.doi.org/10.14778/2556549.2556570

Yannis Theodoridis. 2003. Ten Benchmark Database Queries for Location-based Services. *Comput. J.* 46, 6 (2003), 713–725.

Ouri Wolfson and Eduardo Mena. 2004. Applications of moving objects databases. In *Spatial Databases: Technologies, Techniques and Trends*, Michael Gr. Vassilakopoulos Yannis Manolopoulos, Apostolos N. Papadopoulos (Ed.). IDEA Group Publishing, Hershey, PA, 186203.

Kun-Lung Wu, Shyh-Kwei Chen, and Philip S. Yu. 2006. Incremental Processing of Continual Range Queries over Moving Objects. *Knowledge and Data Engineering, IEEE Transactions on* 18, 11 (Nov 2006), 1560–1575. DOI:http://dx.doi.org/10.1109/TKDE.2006.176

Bo Xu and Ouri Wolfson. 2003. Time-series Prediction with Applications to Traffic and Moving Objects Databases. In *Proceedings of the 3rd ACM International Workshop on Data Engineering for Wireless and Mobile Access (MobiDe '03)*. ACM, New York, NY, USA, 56–60. DOI:http://dx.doi.org/10.1145/940923.940934

Yi Yu, Suhua Tang, and Roger Zimmermann. 2013a. Edge-based Locality Sensitive Hashing for Efficient Geo-fencing Application. In *Proceedings of the 21st ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems (SIGSPATIAL'13)*. ACM, New York, NY, USA, 576–579. DOI:http://dx.doi.org/10.1145/2525314.2527264

Yi Yu, R. Zimmermann, Ye Wang, and Vincent Oria. 2013b. Scalable Content-Based Music Retrieval Using Chord Progression Histogram and Tree-Structure LSH. *Multimedia, IEEE Transactions on* 15, 8 (Dec 2013), 1969–1981. DOI:http://dx.doi.org/10.1109/TMM.2013.2269313

Jianting Zhang and Simin You. 2012. Speeding Up Large-scale Point-in-polygon Test Based Spatial Join on GPUs. In *Proceedings of the 1st ACM SIGSPATIAL International Workshop on Analytics for Big Geospatial Data (BigSpatial '12)*. ACM, New York, NY, USA, 23–32. DOI:http://dx.doi.org/10.1145/2447481.2447485

Tianyu Zhou, Hong Wei, Heng Zhang, Yin Wang, Yanmin Zhu, Haibing Guan, and Haibo Chen. 2013. Point-polygon Topological Relationship Query Using Hierarchical Indices. In *Proceedings of the 21st ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems (SIGSPATIAL'13)*. ACM, New York, NY, USA, 572–575. DOI:http://dx.doi.org/10.1145/2525314.2527263

Received February 2007; revised March 2009; accepted June 2009