Semantic Caching for Spatial Mobile Applications [*]

Brendan C. Fruin
University of Maryland
brendan@cs.umd.edu

Hanan Samet
University of Maryland
hjs@cs.umd.edu

ABSTRACT

Mobile location-based applications rely heavily on network connections. When the mobile devices are offline, such applications become less accessible to users. A semantic cache-based method is proposed to improve the offline accessibility for mobile location-based applications. The central idea is that when users are browsing information, the client program stores this information for later retrieval. In order to accomplish this on a continuous space, the map projection must be discretized restricting the space to a finite number of windows. On a map window query call, the continuous window is translated into a series of discretized windows which either need to be retrieved from the server or are already stored in the cache. By doing the discretization and caching on the client, no changes must be made to the server allowing it to be easily implemented into most existing applications. An extension of semantic caching allows for prediction based upon past user behavior. The usability of the technique is demonstrated by prototyping it on top the NewsStand system so that the query window is constantly changing as users pan and zoom around the world using a gesturing interface, among others. Evaluation shows the prototype to be effective while decreasing the response time.

Categories and Subject Descriptors

H.2.8 [**Database Management**]: Database Applications—*Spatial databases and GIS*; H.3.3 [**Database Management**]: Information Storage and Retrieval—*Information Search and Retrieval*

General Terms

Algorithms, Design, Performance, Reliability

Keywords

Semantic caching, window queries, mobile devices

1. INTRODUCTION

Mobile location-based applications, are gaining increasing use in our daily lives. They are rooted in the desire to incorporate the functionality previously available only in geographic information systems (e.g., [18, 19]) for the representation of spatial, spatio-temporal, and distributed data (e.g., [8, 10, 16, 20, 26]). This has led to modern applications such as Google Maps[1], and Yelp[2]. People use them to search for local attractions. The accessibility of these applications becomes an important issue. If the mobile device hosting the applications goes offline, then the applications become inaccessible to users, and thus users' experiences are sacrificed. Existing work [4, 5, 6, 11, 14, 15, 25] has proposed a cache-based method to solve this problem. Such a method works well for those applications whose data is static, but has trouble dealing with applications with rapidly changing data, such as NewsStand [27], a system developed at the University of Maryland for online browsing of news using a map query interface.

In this paper, a semantic-aware cache-based approach is proposed for spatial mobile applications with dynamic data. This allows for offline interaction, reduction in network usage while requiring changes only to the client application and not the server. The focus will be on map window queries where the underlying data displayed changes as the user moves the map whether it be by a pan, pinch or zoom gesture. In a map projection, the set of all possible queries is pseudo continuous since a map query window does not lock to interval values so the probability a past query will be the same as a future query may be negligible in many applications. To deal with this a map discretization method is used to divide the map projection into grid cells based on the zoom level and the device screen size. An extension of our semantic caching technique allows for prediction of future queries which slightly improves our results.

This semantic caching technique is applied to the NewsStand system [27]. NewsStand is a map-based application framework that aggregates and displays news from RSS feeds at their respective locations based on the content of location references in news articles [27]. Only the news stories associated with locations bounded by the current map viewing window are displayed. In the case of a mobile device, the current map viewing window would be the entire screen when the application is being used. Each time a user updates the map viewing window with an action (i.e. a swipe, click, pinch, or tap) a request is sent to the server for the news stories in the updated viewing window. The map-

[*]This work was supported by the NSF under Grants IIS-10-18475, IIS-12-19023, IIS-13-20791, and CNS-1314857.

[1]http://maps.google.com
[2]http://www.yelp.com

.

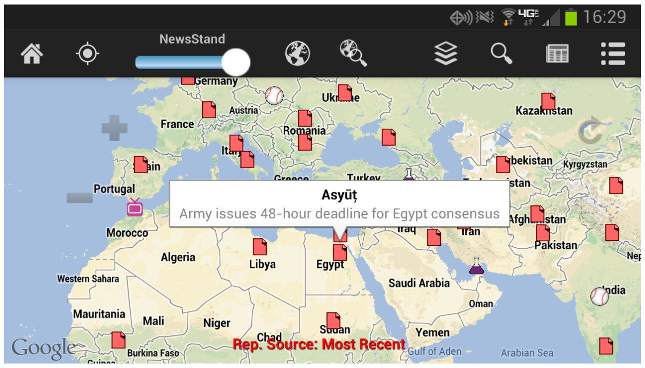


Figure 1: Screenshot of NewsStand for Android App on a Samsung Galaxy S3.

based approach of NewsStand affords an inherent granularity to search based on zoom level providing an approximate search.

NewsStand currently has a web interface and mobile interfaces for the Android (Figure 1), iOS platforms and Windows Phone platforms. In this paper, we apply our approach to the Android NewsStand application.

The rest of this paper is organized as follows. Section 2 describes the semantic caching architecture, along with the SAC extension that provides prediction. Section 3 presents the prototype implementation on top of NewsStand. The evaluation results are discussed in Section 4. Section 5 reviews related work, while Section 6 contains directions for future research.

2. TECHNIQUE

In this section, we define semantic caching and how it can be used as a layer on top of a typical location-based mobile application. We then briefly describe the predictive extension of this work by Liu [13] which is used in the implementation of the semantic caching.

2.1 Semantic Caching

Semantic caching is the process by which the client stores information pertaining to the data currently being cached. This differs from other forms of caching where little additional information may be stored about the underlying data being cached. The use of semantic caching uses the additional semantic information stored by its cached data in order to determine the contents currently held in the cache and the remainder of the data that needs to be retrieved from the server. [7]

Semantic caching can be used in a spatial application once the projection has been divided into finite regions using a process called discretization. For many spatial applications without discretization, there is a small probability that the semantic cache could be utilized even if the underlying data in the cache may be useful due to the additional semantic information not precisely matching the current state. Discretization is the process of taking continuous data and making it discrete. In our case, this means converting pseudo continuous map query windows such as latitude and longitude values into discrete map query windows which can be accomplished by dividing the map projection into a series of grid cells. By making the grid cells at least as large as the device screen, each query can be answered by one, two or four grid cells. A map window query can then be answered by a lookup to the semantic cache for the required cells followed by queries to the server if some of the cells were not currently in the cache.

2.2 Prediction

The caching technique described above only allows for storage of previously seen queries. At times when an application is idle, it may save a user more time in the future by prefetching data that might be used later using prediction. Liu [13] modified the semantic-aware caching technique to calculate predictions based off of previously seen queries and request these from the server while the application was idle. This approach known as Semantic Adaptive Caching (SAC) looked at the query logs to approximate user behavior and predicted up to five queries.

3. IMPLEMENTATION

We applied the SAC algorithm to the Android version of NewStand. This section describes the methods by which SAC was implemented on NewsStand and how its map projection can be divided into a grid. We then describe how caching can be applied to the client side of NewsStand.

3.1 Grid Structure

The Android version of NewsStand uses the Google Maps Android API v2 [2] for its map. The Google Maps Android API uses the spherical Mercator map projection based on the WGS-84 coordinate system used in many commercial mapping APIs including OpenStreetMap, Apple Maps and HERE Maps which preserves the angles of the meridians, but as a result distorts the size and shape of large areas as the object moves away from the Equator. By limiting the maximum latitude to 85.05112878° North and the minimum latitude to 85.05112878° South, the map projection is a square [1]. This square can be mapped to a coordinate grid structure where 179° West and 85.05112878° South corresponds to the point (0,0) and 180° E and 85.05112878 ° N corresponds to the point (360, 360). Latitude and longitude coordinates can be mapped to this grid structure [3] with longitude, *lon*, directly mapping to the first dimension:

$$x = lon + 180$$

and latitude, *lat*, being mapped to the grid with the following formula:

$$y = \frac{180}{\pi} \log\left(\tan\left(\frac{\pi}{4} + \frac{lat}{2} * \frac{\pi}{180}\right)\right) + 180$$

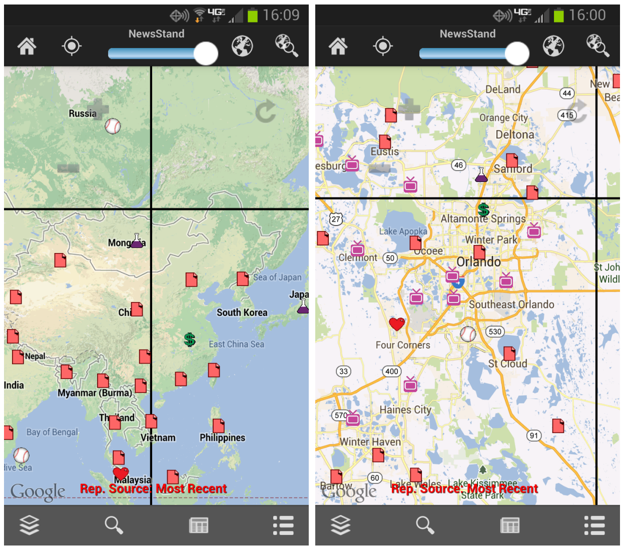
The respective grid coordinates can be mapped back to the projection with longitude:

$$lon = x - 180$$

and latitude:

$$lat = \frac{180}{\pi}\left(2 \arctan\left(e^{\left((y-180)*\frac{\pi}{180}\right)}\right) - \frac{\pi}{2}\right)$$

We then subdivide the grid into equal-sized rectangles based on the current viewing window and the zoom level based on a set of simple rules. The width and the height of the rectangular grid cells must be at least as large as the corresponding width and height of the map viewing window where the width and height correspond to the points of the



(a) (b)

Figure 2: Examples of the grid structure (outlined in black) applied to the NewsStand Android App with (a) showing the grid cells at the maximum zoom out level and (b) showing the grid cells at zoom level 9.

projection's coordinate system visible for a current window given a zoom level. The width and height must also evenly divide 360, the square projection's side length. For example a Samsung Galaxy S3 at the lowest zoom level (furthest zoomed out), contains a map viewing window of approximately 63.28 grid units for its width and 91.23 grid units for its height and the grid at this level is divided into cells that have width 72 units and height 120 units as seen in Figure 2(a). While zooming in to the city level (zoom level 9) has a map viewing window of approximately 0.98 grid units for its width and 1.43 grid units for its height has a grid that is divided into cells that have width 1 unit and height 2 units as seen in Figure 2(b). Given the fact that all grid cells must be at least as large as the current map viewing window, we know that any map viewing window is contained in one, two or four grid cells.

3.2 Client Caching

Due to the nature of news needing to be up-to-date and window queries being sent even with the most minor of updates to the map viewing window, some form of caching must take place in order to reduce duplicate or near-duplicate queries from having to be recalculated. As was seen in Section 3.1, the proposed grid structure could be used so that all window queries could be answered from the sum of at most four grid cells. By keeping track of the results of previously seen grid cells and predicting future grid cells, the grid cell's associated data can be stored in a cache and future requests for these cells may not need to be calculated.

We begin by initializing an empty cache, C, to store the results of previously seen or current grid cells with their associated data and the time that they were downloaded from the server and a τ value which is the maximum time downloaded data can be used before it is considered stale. Each time the map viewing window is updated begin by removing

```
Data: $window$ is the current map viewing window
Data: $C$ is the current cache
Data: $query\_log$ is the query log of the last 200 queries
$grid\_cells$ = calculate grid cells associated with $window$
$cached\_results$ = initialize to empty
for $curr\_cell \in grid\_cells$ do
    $cell\_cached$ = false
    for $cached\_cell \in C$ do
        if $curr\_cell = cached\_cell$ then
            $cell\_cached$ = true
            if $cached\_cell$ is downloaded then
                Append $cell\_cached$ data to $cached\_results$
            end
        end
    end
    if $cell\_cached$ = false then
        request data for $curr\_cell$ from server
    end
end
Wait (for all $grid\_cells$)
Display (results of $grid\_cells$)
$predicted\_cells$ = predict ($grid\_cells$, $query\_log$)
for $predicted\_cell \in predicted\_cells$ do
    $cell\_cached$ = false for $cached\_cell \in C$ do
        if $predicted\_cell = cached\_cell$ then
            $cell\_cached$ = true
        end
    end
    if $cell\_cached$ = false then
        request data for $predicted\_cell$ from server
    end
end
```

Algorithm 1: Map Window Change

all elements from C whose download time is greater than τ which in the case of NewsStand should be fairly short (since it deals with streaming news) such as 120 seconds. Once the stale elements have been removed, pass the current cache C and the current map viewing window to Algorithm 1.

Algorithm 1 begins by calculating the grid cells that intersect with the current window storing the intersected grid cells in *grid_cells* and initializes an empty results list, *cached_results*. Each grid cell in *grid_cells* is then checked for membership in C. If a grid cell in *grid_cells* is not contained in C, an asynchronous server request is dispatched to download the associated data. If a grid cell in *grid_cells* is contained in C and the cached grid cell has finished downloading the data, then the results are appended to the *cached_results*. Note that due to the requests to the server being sent asynchronously, a grid cell may exist in the cache that is still waiting for its data to be returned from the server. On completion of an asynchronous call associated with the current map viewing window, the data is appended to *cached_results* and a check is made to determine whether all of the current grid cells have been downloaded. The *cached_results* are not displayed until all of the data for the current grid cells have been downloaded thereby preventing data from populating in only certain grid cells of the map while other grid cells remain empty.

After each grid cell in *grid_cells* has been checked for membership in C and the server has responded with all windows

k	0	1	2	3	4	5
Dataset 1						
Cache Hit (%)	53.6	55.0	55.0	54.5	54.0	52.4
Coverage (%)	73.1	75.1	74.0	74.0	73.7	73.0
Dataset 2						
Cache Hit (%)	44.0	45.2	45.0	44.8	44.2	43.8
Coverage (%)	58.7	60.9	61.0	60.3	60.6	59.9
Dataset 3						
Cache Hit (%)	68.7	70.0	69.3	68.2	68.5	69.0
Coverage (%)	80.0	81.0	81.0	80.7	80.6	80.4

Table 1: Effectiveness study on mobile devices looking at the cache hit and coverage percentage for three datasets.

not in C, the map is updated with the results from *grid_cells*. Future grid cells are then predicted using *grid_cells* and the query log of the last 200 window queries, *query_log*, using the algorithm proposed by Liu [13] and the results are stored in *predicted_cells*. Each grid cell in *predicted_cells* that is not contained in C sends an asynchronous server request to retrieve the data for the current grid cell. C is set to the union of the *grid_cells* with their associated data and the *predicted_cells* with their associated data.

4. EVALUATION

We evaluate our implementation with respect to both effectiveness and efficiency. We studied the performance of our algorithm on a mobile device by recording window movements on the NewsStand Android application. We created and analyzed three datasets and evaluated the effectiveness of our cache along with the query running times.

Datasets.

We simulated users inputs by recording window movements on the NewsStand Android application on a Samsung Galaxy S3 capturing three different datasets. Each dataset contained 200 window movements. Dataset 1 consisted of primarily (over 60%) panning operations while Dataset 2 consisted of primarily (over 60%) zooming operations. Dataset 3 contained a similar amount of pan and zoom operations. The results for both the effectiveness and running time are the average of each of the datasets of three subsequent runs to control for varying overhead when communicating with the server.

Effectiveness.

To evaluate the effectiveness, we computed the *cache hit* rate and the *coverage*. The *cache hit* rate is the percentage of queries that can be completely answered by the cache. For some queries, even though not all responses are in the cache, a partial result may be available. The *coverage* computes the percentage of the responses for the next query that can be answered based on the contents in the cache.

We vary k, the number of predictive queries, from 0 to 5, and evaluate the system using a cache size of 100. Note that when k is zero we are using a cache with no prediction, i.e. we are using just the semantic caching algorithm. The results are shown in Table 1. As can be seen the cache hit varies from 43.8% in Dataset 2 to 70.0% in Dataset 3 while Dataset 1 falls in between. The highest cache hit rate is achieved when k is set to 1 in all three datasets (note

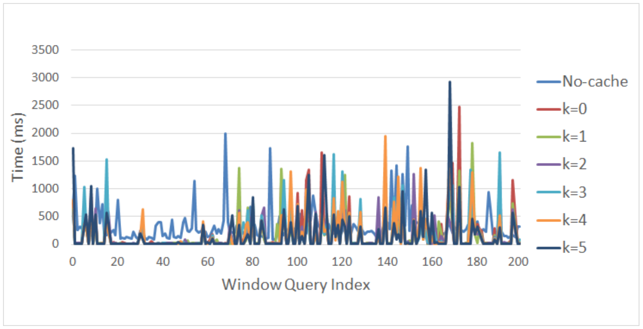


Figure 3: Query response time (ms)

k	0	1	2	3	4	5	No-cache
Dataset 1 (ms)	260	216	209	204	183	201	263
Dataset 2 (ms)	272	266	261	280	294	326	350
Dataset 3 (ms)	188	162	147	170	171	161	325

Table 2: Query response on mobile device (in ms)

k=2 gives same cache hit rate for Dataset 1). The coverage percentage varies from 58.7% in Dataset 2 to 81.0% in Dataset 3 while Dataset 1 again falls in between. Dataset 1 has the highest coverage percentage when k is set to 1 and in Dataset 2 the highest coverage percentage is achieved when k is set to 2. In Dataset 3, the highest coverage percentage is when k is set to 1 or 2. We see that both the cache hit rate and the coverage are quite similar for different k for a given dataset. We observe that overall the results for $k = 1$ and $k = 2$ are slightly better than those for $k > 2$. We believe that this is due to SAC's prediction algorithm only looking at the overall past movements and not with respect to the user's current location or their most recent of requests.

The cache hit rate shown in Table 1 means that over 43.8% of all window queries can be answered without additional communication to the server since the results were previously cached. Additionally, over 58.7% of the discretized windows are in the cache. Important to note, is that the simulated dataset most like a typical user, Dataset 3, can achieve both a 70.0% cache hit rate and 81.0% coverage percentage when k is set to 1. This allows for high usability when SAC is in offline mode due to limited or no connectivity.

Running Time.

We evaluated the query response time using SAC compared with a no-cache solution. For SAC, we varied k from 0 to 5, and the cache size was 100. The average response time for Dataset 3 for varying values of k and the no-cache solution is plotted in Figure 3. From this figure, we observe that the no-cache solution has a more stable response time with periodic longer queries, while SAC's response time is more unstable with many queries taking under 50 milliseconds when SAC uses the cache. For those cache hit queries, SAC's response time is less than the no-cache solution. The reason is that the no-cache solution will always submit a query to the database server, while SAC will respond to a user immediately. For cache miss queries, SAC may result in a larger latency as up to four queries will be requested from the server. Even though these are asynchronous calls, the database server's response time increases when responding to those queries and the client's resources are divided in order accommodate each of the requests.

The average response times in milliseconds for the three datasets is shown in Table 2. Here we see that in all circumstances, including when k=0, on the average the cache outperforms the no-cache solution. We see that by setting k to 2 in Dataset 3 we get over a 50% reduction in average query response time. Interestingly, an increase in k may decrease the average query response time even though there is a decrease in the cache hit rate and coverage as was seen in Table 1. This is most likely due to queries being predicted that are eventually used, but not necessarily when they were expected to be called. In some cases, the SAC query response time decreases compared to the decrease in the no-cache solution may be considered negligible, but more important is the fact that these windows are available offline.

5. RELATED WORK

Client-side caching for mobile devices has been widely researched [4, 5, 6, 11, 14, 15, 25]. Barbará and Imieliński [5] studied the issue of a large number of mobile devices querying remote databases assuming there would be periods of both awake and sleep times for the devices similar to current mobile applications switching between other applications or the devices being on standby. In this model, they believed that cached data would have to be explicitly invalidated with an invalidation report sent from the server while our invalidation is determined on the client side requiring no additional communication with the server. Lee et al. [11] looked into semantic query caching for mobile devices and had a timer associated with each caching unit where on expiration a refresh would be sent to the server to update the expired cache data. The cache accommodation used by Lee et al. is similar to our approach in that it utilizes semantic locality, adapting to the patterns of query results. However, their proposed caching algorithm only accounts for previously seen query results while our algorithm uses both predictive analysis of future queries and the results of previously seen queries resulting in an increase in network flow in order to potentially reduce wait times.

Semantic caching for mobile devices has been researched with regards to spatial applications [4, 15, 25]. Similar to our work, Ren and Dunham [15] looked into applying semantic caching techniques to store location queries for areas that may have wide areas of overlap in order to answer future queries locally as opposed to always making server calls. They looked at predicting future queries based on the user's current location, direction and velocity derived from past locations, while we look at a comparable problem of caching and predicting window queries. Sun and Zhou [25] created a system for semantic caching of location queries by iteratively decomposing the map into cells which they term Peano cells which are able to answer queries at different zoom levels based on the combination of lower level Peano cells. These different levels of caching allow for zooming to be contained in the cache, but not panning without predictive caching or prefetching. Based on the level of the decomposition, the proposed grid structure may require a large amount of space to be allocated to the cache for each of the Peano cells. Amini et al. [4] developed Caché which groups a set of location queries in order to issue one large request to the server. By issuing one large request, the user is granted stronger location privacy in that their exact location is not continuously sent to the server and offline accessibility is improved by downloading and storing and a larger area on the client. This work primarily considers point queries while we focus on window queries.

6. DIRECTIONS FOR FUTURE WORK

Future work includes improving SAC's prediction algorithm so that as the number of predictions increases the cache hit rate and the coverage percentages increase as well. Possible improvements for the prediction algorithm include taking into account the user's current location and what the user's past actions were at this location. For example, in a general case a user may be more likely to zoom in over a land mass or to pan over an ocean. Greater improvements to the prediction algorithm may come from more specific prediction based on a current grid cell or collection of grid cells. For example, user A may live in Orlando, FL so that s/he often zooms into grid cells that contain Orlando while as user B from Miami, USA may zoom into some grid cells that contain Orlando since it is relatively close to Miami from a world perspective, but at a close enough zoom level may pan to the south in order to reach Miami. In addition to historical prediction, prediction can be modified to give weighting to recent queries, i.e. given the user has panned to the left 5 times in a row the chance that s/he will continue in this direction is increased.

Additional future work involves caching the results of other operations which can be the subject of subsequent zooming and panning operations. This is especially relevant for nearest neighbor (e.g., [17, 21, 24]), shortest path (e.g., [22, 23]), and spatial join (e.g., [9]) queries.

7. ACKNOWLEDGEMENTS

This paper is a shorter version of joint work done with Liu Chang in developing the SAC system [13] that appeared in SIGSPATIAL 2013. For detailed information on the prediction algorithm, refer to this paper.

The NewsStand web version can be found at `http://newsstand.umiacs.umd.edu`. The code that was used to generate the test data sets can be found at `http://tiny.cc/winrecorder`.

8. REFERENCES

[1] Google map types. `https://developers.google.com/maps/documentation/javascript/maptypes`.

[2] Google maps android api v2. `https://developers.google.com/maps/documentation/android/`.

[3] OpenStreetMap Mercator. `wiki.openstreetmap.org/wiki/Mercator`.

[4] S. Amini, J. Lindqvist, J. I. Hong, J. Lin, E. Toch, and N. M. Sadeh. Caché: caching location-enhanced content to improve user privacy. In *MobiSys*, pages 197–210, 2011.

[5] D. Barbará and T. Imieliński. Sleepers and workaholics: caching strategies in mobile environments. In *VLDB*, pages 567–602, 1995.

[6] B.-G. Chun, C. Curino, R. Sears, A. Shraer, S. Madden, and R. Ramakrishnan. Mobius: unified messaging and data serving for mobile apps. In *MobiSys*, pages 141–154, 2012.

[7] S. Dar, M. J. Franklin, B. Jónsson, D. Srivastava, and M. Tan. Semantic Data Caching and Replacement In *VLDB*, pages 330–341, 1996.

[8] G. R. Hjaltason and H. Samet. Speeding up construction of PMR quadtree-based spatial indexes. *VLDBJ*, 11(2):109–137, 2002.

[9] E. G. Hoel and H. Samet. Benchmarking spatial join operations with spatial output. In *VLDB*, pages 606–618, 1995.

[10] G. S. Iwerks, H. Samet, and K. Smith. Maintenance of spatial semijoin queries on moving points. In *VLDB*, pages 828–839, 2004.

[11] K. C. K. Lee, H. V. Leong, and A. Si. Semantic query caching in a mobile environment. In *SIGMOBILE*, pages 28–36, 1999.

[12] M. D. Lieberman and H. Samet. Supporting rapid processing and interactive map-based exploration of streaming news. In *GIS*, pages 179–188, 2012.

[13] C. Liu, B. Fruin, and H. Samet. SAC: semantic adaptive caching for spatial mobile applications In*SIGSPATIAL*, pages 174–183, 2013.

[14] F. Qian, K. S. Quah, J. Huang, J. Erman, A. Gerber, Z. M. Mao, S. Sen, and O. Spatscheck. Web caching on smartphones: ideal vs. reality. In *MobiSys*, pages 127–140, 2012.

[15] Q. Ren and M. H. Dunham. Using semantic caching to manage location dependent data in mobile computing. In *MOBICOM*, pages 210–221, 2000.

[16] H. Samet. A quadtree medial axis transform. *CACM*, 26(9):680–693, 1983.

[17] H. Samet. K-nearest neighbor finding using MaxNearestDist. *TPAMI*, 30(2):243–252, 2008.

[18] H. Samet, H. Alborzi, F. Brabec, C. Esperança, G. R. Hjaltason, F. Morgan, and E. Tanin. Use of the SAND spatial browser for digital government applications. *CACM*, 46(1):63–66, 2003.

[19] H. Samet, A. Rosenfeld, C. A. Shaffer, and R. E. Webber. A geographic information system using quadtrees. *Pattern Recognition*, 17(6):647–656, 1984.

[20] H. Samet and M. Tamminen. Bintrees, CSG trees, and time. In *SIGGRAPH*, pages 121–130, 1985.

[21] J. Sankaranarayanan, H. Alborzi, and H. Samet. Efficient query processing on spatial networks. In *GIS*, pages 200–209, 2005.

[22] J. Sankaranarayanan and H. Samet. Distance oracles for spatial networks. In *ICDE*, pages 652–663, 2009.

[23] J. Sankaranarayanan, H. Samet, and H. Alborzi. Path oracles for spatial networks. *PVLDB*, 2(1):1210–1221, 2009.

[24] J. Sankaranarayanan, H. Samet, and A. Varshney. A fast all nearest neighbor algorithm for applications involving large point-clouds. *Computers & Graphics*, 31(2):157–174, 2007.

[25] S. Sun and X. Zhou. Semantic caching for web-based spatial applications. In *APWeb*, pages 783–794, 2005.

[26] E. Tanin, A. Harwood, and H. Samet. A distributed quadtree index for peer-to-peer settings. In *ICDE*, pages 254–255, 2005.

[27] B. Teitler, M. D. Lieberman, D. Panozzo, J. Sankaranarayanan, H. Samet, and J. Sperling. Newsstand: A new view on news. In *GIS*, pages 144–153, 2008