5-2012

Spatial Queries in Wireless Broadcast Environments [Keynote Speech]

Kyriakos MOURATIDIS
Singapore Management University, kyriakos@smu.edu.sg

DOI: https://doi.org/10.1145/2258056.2258065

Spatial Queries in Wireless Broadcast Environments

Invited Paper *

Kyriakos Mouratidis
School of Information Systems
Singapore Management University
80 Stamford Road, Singapore 178902
kyriakos@smu.edu.sg

ABSTRACT

Wireless data broadcasting is a promising technique for information dissemination that exploits the computational capabilities of mobile devices, in order to enhance the scalability of the system. Under this environment, the data are continuously broadcast by the server, interleaved with some indexing information for query processing. Clients may tune in the broadcast channel and process their queries locally without contacting the server. In this paper we focus on spatial queries in particular. First, we review existing methods on this topic. Next, taking shortest path computation as an example, we showcase technical challenges arising in this processing model and describe techniques to address them.

Categories and Subject Descriptors

H.2.8 [**Database Management**]: Database Applications

General Terms

Algorithms, Design, Performance

Keywords

Spatial Queries, Shortest Path Computation, Air Indices

1. INTRODUCTION

Mobile devices with computational and wireless communication capabilities are becoming increasingly popular. At the same time, the technology behind positioning systems is constantly evolving, enabling the integration of low-cost GPS devices in any portable unit. Consequently, new mobile computing applications emerge, allowing users to issue location-dependent queries in a ubiquitous manner. For instance, users of these applications can request for the nearest business or service to their location, navigate to a target address, locate personal contacts on a map or receive alerts such as warnings of traffic jams.

To answer a location-based query, such as navigation to a specific address, the device typically either (i) pre-loads the map data and runs the query locally, or (ii) it connects to a location server through a GSM/3G/Wi-Fi service provider. The option of storing the map information locally imposes heavy requirements on the already limited storage of the mobile device and it is viable only for few pre-selected maps. In case the user travels to a new city/country, this option would fail. Moreover, storing maps locally may result in routing decisions based on outdated network information.

On the other hand, given the growing number of users and services, the option of querying online location servers is already facing scalability limitations, a situation expected to worsen in the near future. While coping with increasing query loads necessitates continuous infrastructure upgrades at the side of service providers, a greater challenge is network congestion. According to [21] the number of mobile subscribers has risen by 250% in the last 3.5 years, while data traffic volume from handheld devices is growing by more than 10 times per year. The main concern of mobile service providers in the Mobile World Congress 2010 [1] was that the number of data-capable phones is growing faster than network capacity, so network overload is considered an immediate risk, and data traffic management is becoming a major priority for the telecommunications industry.

A promising solution to the above problem is the wireless broadcast model [11]. In this model the location server repeatedly broadcasts the data on the air (using GSM, 3G, Wi-Fi, HD radio, or even a Bluetooth network), while the clients tune in the broadcast channel and process their queries locally. Since the server's hardware requirements are low, multiple servers could be installed at different locations to provide coverage in large areas. The main advantage of the broadcast model is that it can support an arbitrary number of users/queries, since no processing takes place at the server, and the network overhead is irrelevant to the number of clients. A side benefit is that user privacy is guaranteed, as the location server is unaware of user positions and queries; this has been a serious concern recently [3, 17].

Wireless broadcasting has been considered for spatial queries, such as traditional range and nearest neighbor (NN) processing (e.g., [16, 26, 28]). In this paper we provide a general background for the wireless broadcast environment and briefly review spatial processing techniques designed for this setting. Using shortest path computation as a case study, we

*Material based on [14] and [16] appearing in Proceedings of VLDB'10 and IEEE Transactions on Mobile Computing respectively.

MobiDE '12, May 20, 2012 Scottsdale, Arizona, USA

take a closer look into the challenges and solutions described in [14].

2. WIRELESS BROADCASTING

In the wireless broadcast model the server repeatedly transmits identical *broadcast cycles*, each containing the entire database and potentially some indexing information (called *air index*). The broadcast cycle consists of fixed-size *packets*, defining the smallest information unit transmitted.

The most common data organization method is the $(1, m)$ *interleaving scheme* [11], shown in Figure 1. The data objects are divided into m distinct segments, and each data segment in the transmission schedule is preceded by a complete version of the index. This way the client may receive a copy of the index (and start processing its query) immediately after the completion of the currently transmitted data segment. [11] also introduces an alternative, distributed index that reduces the degree of replication in order to further improve performance. Specifically, instead of the entire index being replicated prior to each data segment, only the index that corresponds to the subsequent segment is included (i.e., replication occurs at the upper levels of the index tree).

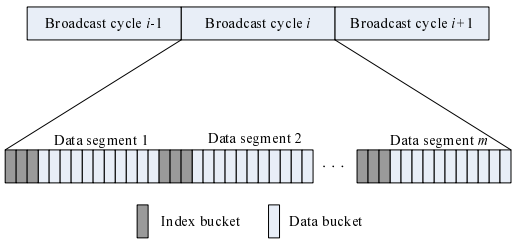


Figure 1: The $(1, m)$ interleaving scheme

The main objective of air indices is to minimize the power consumption at the mobile client. Although in a broadcast environment the uplink transmissions are avoided, receiving all the downlink packets from the server is not energy-efficient. For instance, the (widely used) 802.11 WaveLAN card consumes 1.65W, 1.4W, and 0.045W in transmit, receive, and sleep states respectively [12]. Therefore, it is imperative that the client switches to the sleep mode (i.e., turns off the receiver) whenever the transmitted packets do not contain any useful information. Based on the data organization technique in Figure 1, query processing at the mobile client is performed as follows: (i) the client tunes in the broadcast channel when the query is issued, and goes to sleep until the next index segment arrives, (ii) the client uses the index to determine when the data objects that satisfy its query will be broadcast, and (iii) the client goes to sleep and returns to the receive mode only to retrieve the corresponding data objects.

To measure the efficiency of an indexing method, two performance metrics have been considered in the literature: (i) *tuning time*, i.e., the total time that the client stays in receive mode to process the query, and (ii) *access latency*, i.e., the total time elapsed from the moment the query is issued until the moment that all the result objects are received. In other words, the tuning time is a measure of the power consumption at the mobile client, while access latency reflects the user-perceived system responsiveness.

Most of the existing work on query processing for wireless data broadcast focuses on relational data and indices. Imielinski et al. [10] introduce two methods, namely *hashing* and *flexible* index, for retrieving records based on their key values. The same authors [11] propose the aforementioned $(1, m)$ and distributed index techniques, and study their performance for the B$^+$-tree index. Hu et al. [9] consider multi-attribute queries, and investigate the performance of three indices (index tree, signature, and hybrid index) under this scenario. In another study, Xu et al. [23] propose the exponential index, which offers the ability to adjust the trade-off between access latency and tuning time.

3. AIR INDICES FOR SPATIAL QUERIES

Hambrusch et al. [7] explore the possibility of broadcasting spatial data together with a data partitioning index. They present several techniques for spatial query processing that adjust to memory limitations at the mobile device. The authors evaluate their methods experimentally for range queries (using the R*-tree [2] as the underlying index), and illustrate the feasibility of this architecture.

Xu et al. [24] and Zheng et al. [29] focus on *single* nearest neighbor (NN) search in broadcast environments. Both methods utilize a pre-computed Voronoi diagram that can answer any NN query by identifying the Voronoi cell that encloses it. Specifically, the Voronoi diagram of the data objects is built prior to the construction of the air index. The D-tree [24] then recursively partitions the data space (containing the Voronoi cells) into areas with a similar number of cells. This procedure is repeated until each area contains exactly one cell. On the other hand, the *grid-partition* index [29] divides the space into disjoint grid cells, each intersecting multiple Voronoi cells. Both methods are found superior to broadcast solutions based on R*-trees.

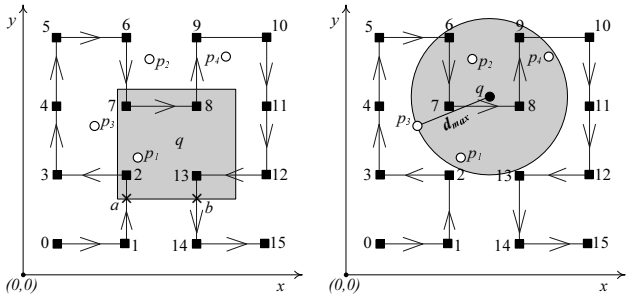
Zheng et al. [27] investigate another class of NN queries, namely *linear* NN (LNN) queries, in the context of wireless broadcast systems. A LNN query retrieves the NNs of every point on a line segment, i.e., the solution comprises of a series of points, each being the NN of a particular segment of the line. The authors adjust the methods introduced by Tao et al. [22] to fit the broadcast environment, and show that their techniques outperform the naïve solution where there is no index available.

Focusing on kNN search on the air, [25] proposes an *approximate* kNN query processing algorithm that is not guaranteed to always return k objects. The idea is to use an estimate r of the radius that is expected to contain at least k points. Using this estimate, the search space can be pruned effectively at the beginning of the search process. The authors also introduce a learning algorithm that adaptively re-configures the estimation algorithm to reflect the distribution of the data. Regarding query processing, two different approaches are proposed: (i) the standard R*-tree index enhanced with the aforementioned pruning criterion, and (ii) a new *sorted list* index that sorts objects on each spatial dimension. The sorted list method is shown to be superior to the R*-tree only for small values of k.

Gedik et al. [5] describe several algorithms to improve kNN query processing in sequential-access R-trees. They investigate the effect of different broadcast organizations on the tuning time, and also propose the use of histograms to enhance the pruning capabilities of the search algorithms. Park et al. [20] focus on reducing the access latency of kNN search by accessing the data segment of the broadcast cycle.

In particular, they propose a method where the data objects are sorted according to one spatial coordinate. This way, the client does not need to wait for the next index segment to arrive, but can start query processing immediately by retrieving the actual data objects.

The Hilbert Curve Index (HCI) [26] is a general framework for processing both range and kNN queries in wireless broadcast systems. HCI is based on the $(1, m)$ interleaving scheme. It exploits the linear access of the broadcast channel by transforming the two-dimensional space into a one-dimensional, using the Hilbert space-filling curve [13]. Once the objects are mapped onto the Hilbert curve, they are indexed with a B$^+$-tree which is then broadcast on the air (as the index segment). Range queries are processed as follows. Consider Figure 2(a) where the Hilbert values range from 0 to 15, and the query region is the shaded rectangle. The client first determines the first (a) and the last (b) points on the Hilbert curve that intersect the query window (illustrated as crosses in the figure). Letting $H(a)$ and $H(b)$ be the Hilbert values of a and b, the client listens to the first broadcast index segment, and retrieves all objects inside the Hilbert range $[H(a), H(b)]$. In our example, $H(a) = 2$ and $H(b) = 13$ (note that the Hilbert value of a point is the integer that corresponds to its closest solid square in the two-dimensional space). Objects p_1, p_2, p_3, p_4 are identified (with $H(p_1) = 2, H(p_2) = 6, H(p_3) = 7, H(p_4) = 9$), but not all of them satisfy the query. Thus, they are mapped back to the two-dimensional space, and their associated data are received (from the corresponding data segment) only if they are inside the query region. In our example, the result includes only p_1.



(a) Range query (b) 2-NN query

Figure 2: HCI examples

In HCI, kNN queries are processed with a two-step approach. In the first step, the query point q is mapped onto the Hilbert curve, and the k objects closest to q (on the curve) are determined. In the second step, the maximum distance d_{max} (from q) across these k objects is calculated, and a range query is processed (in the way described above) to retrieve a set of candidate neighbors. Within this set, the k closest objects to q are identified by comparing their individual distances from q. Figure 2(b) exemplifies this procedure for a 2-NN query q, where $H(q) = 8$. In the first step, the client identifies the $k = 2$ data objects with the closest Hilbert values to $H(q) = 8$; these are p_3 and p_4, with $H(p_3) = 7$ and $H(p_4) = 9$ respectively. In the second step, the client additionally retrieves p_1 and p_2 since they fall inside the circle with center at q and radius $d_{max} = \max(dist(p_3), dist(p_4))$ (shown shaded). Among these candidate objects, the $k = 2$ closest ones (i.e., p_1 and p_2) are selected and their contents are retrieved from the corresponding data segments.

The Distributed Spatial Index (DSI) [28] is another general air index, supporting both range and kNN queries. DSI is a distributed index that aims at minimizing the access latency at the cost of an increased tuning time. Similar to HCI, it uses the Hilbert curve to order the data. The broadcast cycle is constructed as follows. The ordered data are partitioned into a number of *frames*. Each frame holds a fixed number of consecutive objects (on the Hilbert curve) and an index table. Each entry of the index table contains a pointer to a subsequent frame, along with the minimum Hilbert value inside that frame. Specifically, the ith entry points to the e^ith subsequent frame, where e is a system parameter. Figure 3 illustrates a situation where each frame contains 2 data objects, $e = 2$, and the subscripts of the objects coincide with their Hilbert order. The index table of every frame contains pointers (and the corresponding minimum Hilbert values) to the 1st, 2nd, 4th, and 8th subsequent frame; the arrows in the figure represent the index entries in Frame 1. These exponentially increasing frame intervals enable fast access to both nearby and distant frames. To identify the object with a specific Hilbert value, the client listens to the current frame, and follows the pointer to the furthest frame that does not exceed the target Hilbert value (i.e., goes to sleep until that future frame is broadcast). The procedure is repeated for this coming frame and terminates when the search converges to the frame that contains the target object. Query processing in DSI is similar to HCI, relying on the locality preservation of the Hilbert curve. The improved access latency of DSI stems mainly from the fact that the client retrieves indexing information directly when it first tunes in the channel (instead of waiting for the next index segment to be broadcast).

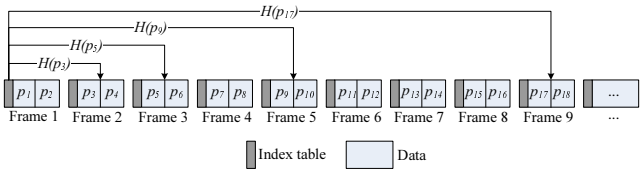


Figure 3: DSI example

In the Broadcast Grid Index method (BGI) [16], the data objects are initially partitioned using a regular grid, i.e., a grid with equi-sized cells. Each grid cell holds the coordinates of objects that lie inside it, plus their total number. This information forms the index of BGI. The full object data are divided into m segments using the $(1, m)$ scheme, and a copy of the index precedes each segment.

In order to compute a kNN query, the algorithm performs two steps. In the first step, the client receives index information for the cells. Using the extent and number of objects in each cell, it calculates an upper bound d_{max} of the radius around its position that contains at least k objects. In the second step, the device receives from the broadcast cycle only the objects within distance equal to or smaller than d_{max}. As the client receives index information about more cells or more object coordinates, it reduces the upper bound d_{max} incrementally, excluding more unnecessary

packets each time. BGI can additionally support continuous kNN queries. These are standing queries that require continuous re-evaluation as the data objects move.

Another body of work considers situations in wireless broadcasting where error rates are high and vary significantly across the system. [19] proposes adaptive error correction techniques to keep access latency and tuning time low. It also develops adaptive query processing methods that cater for the different error rates of individual clients. The same authors demonstrate the feasibility of their techniques, and of wireless broadcasting in general, using a real running prototype [18].

4. CASE STUDY: SHORTEST PATH

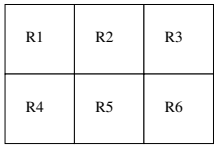
We chose the context of shortest path queries to showcase key considerations in a wireless broadcast system and describe the approach proposed in [14] to address them.

4.1 Query Definition and Challenges

Unlike spatial techniques described in Section 3, shortest path queries involve no data objects, but a graph – in the context of location-based services, this graph is typically a *road network*. That is a directed weighted graph $G = (V, E)$. V is the set of nodes v_i, each of the form $< id_i, x_i, y_i >$, denoting the identifier and the spatial coordinates of the node. E is the set of edges (v_i, v_j), each modeled as a triplet $< id_i, id_j, w_{ij} >$ that contains the identifiers of the connected nodes v_i and v_j and the weight w_{ij} of the edge; w_{ij} may correspond to the edge's length, travel time, toll fee, etc. The *shortest path* between a source node v_s and a target node v_t is the edge sequence connecting v_s and v_t with the minimum sum of edge weights. The sum of edge weights along the shortest path is called the *graph distance* between v_s and v_t.

A common technique to compute a shortest path is Dijkstra's [4] algorithm. Initially, nodes adjacent to v_s are pushed into a min-heap with their graph weights from v_s as sorting keys. The top node v in the heap is popped in every iteration and expanded, i.e., its adjacent nodes v' are enheaped with key equal to that of v plus the weight of edge (v, v'). The process stops when v_t is popped. The shortest path is returned by tracing backwards the expansions that lead to v_t.

Let's attempt to apply Dijkstra's algorithm in the wireless broadcast setting. The broadcast cycle includes the road network information, i.e., the adjacency lists of all nodes. Assume that the client's device somehow knows when to wake up to receive the adjacency information of the source node v_s. Upon receipt of this information, it pushes in a Dijkstra heap all the adjacent nodes of v_s. Letting v be the node at the head of the heap, Dijkstra needs to listen to v's adjacency list. As this information may have already been broadcast, the device might need to wait for the next broadcast cycle. The access latency of this approach is unacceptable, as the device may have to wait for as many cycles as the number of nodes popped by Dijkstra. Since there is no way to tell in advance what the source and the target of user queries will be, it is not possible to organize the nodes in the broadcast cycle in a certain order so that every needed node appears later than the previously de-heaped one. The situation could be improved if the network was partitioned into regions and the adjacency lists for the same region were broadcast together. Still, however, when Dijkstra search reaches the borders of the current region, the device might have to wait for the next cycle to receive network information about the neighboring regions.



	R1		R2		R3		R4		R5		R6	
	Min	Max	Min	Max	Min	Max	Min	Max	Min	Max	Min	Max
R1			1	5	6	8	1	4	3	7	8	9
R2	1	5			1	3	2	5	1	2	2	4
R3	4	6	1	3			5	8	2	4	1	4
R4	1	3	2	4	5	8			2	3	4	7
R5	3	6	1	3	2	4	1	3			1	2
R6	7	9	2	4	1	2	5	6	1	3		

Figure 4: EB index

Due to this problem, selective tuning is not feasible in Dijkstra's adaptation. An idea would be for the client to listen to the *entire* broadcast cycle, and then process the query locally in the complete network. Access latency this way never exceeds one cycle. However, this approach incurs a prohibitively long tuning time (all packets of the cycle are received). Other established techniques, like A* search [8], or common pre-computation-based approaches (ArcFlag [15], Landmark [6], etc), run into similar problems. This calls for processing methods specifically designed for the broadcast setting. [14] proposes two such methods, described in the following sections.

4.2 Elliptic Boundary (EB) Method

Intuitively, in order to efficiently process shortest path queries we have to partition the road network into regions and use an index structure to guide the search through them. To keep access latency and tuning time low, the index should be particularly concise, much more than existing indices designed for disk-resident networks. The Elliptic Boundary method (EB) follows this approach.

Its crux is to first provide the client with an upper bound of the shortest path distance between v_s and v_t. This bound is used to prune (i.e., to avoid listening to) network information about nodes that lie too far away from v_s and v_t to affect the shortest path search. This is achieved by partitioning the network into regions, and placing in the index of EB information about the minimum and maximum possible distance from any partition to any other. EB owes its name to the fact that the search area is reminiscent of a network-based ellipse with foci the regions of v_s and v_t (although the search space is by no means an ellipse or any other geometric shape, as no relationship between network distance and Euclidean distance is assumed).

The index of EB includes two components. The first is a KD-tree that partitions the road network and provides a mapping of (the Euclidean coordinates of) nodes into the resulting regions. The second component specifies minimum/maximum graph distances between every pair of regions. Figure 4 gives an example of the second component for a network partitioned in the 6 regions shown in the left part of the figure. To keep the access latency low, the index is replicated m times in the broadcast cycle, following the $(1, m)$ scheme.

Posed a query, the client tunes in the broadcast channel, and listens to the current packet. It retrieves a pointer to the next index and sleeps. It wakes up when the index starts being broadcast, and receives it in its entirety. The regions R_s and R_t that include the source and target nodes are identified (using the KD-tree in the first component of the index). The second component of the index specifies the maximum possible graph distance between any node in R_s

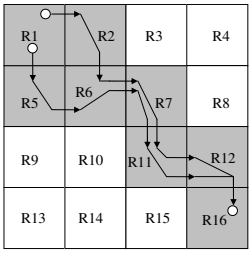


Figure 5: Shortest paths from R_1 to R_{16}

and any node in R_t. This value serves as an upper bound of the shortest path distance. In Figure 4, if the source v_s is in region R_1 and the destination v_t in region R_5, the maximum shortest path distance is 7 (see entry (R_1, R_5) in the array).

The next step is to determine which regions must be received. The client needs to listen to only those regions R for which the sum of minimum distance from R_1 to R plus the minimum distance from R to R_5 is no larger than 7. Hence, the client needs to receive (i) the source and target regions R_1, R_5, and (ii) regions R_2 $(1 + 1 < 7)$ and R_4 $(1 + 2 < 7)$. Regions R_3 and R_6 are not necessary since the sum of their minimum distances is larger than 7.

Upon deciding which regions are necessary, the client sleeps and wakes up when their data (contained nodes and adjacency lists thereof) are broadcast. When all necessary regions are downloaded, the client performs a Dijkstra search in their union (a sub-graph of the network) and derives the shortest path. Observe that access latency never exceeds one broadcast cycle in EB.

4.3 Next Region (NR) Method

While EB allows for selective tuning, its search space (i.e., the set of regions received by the client) may still be large. The problem is exacerbated when the source and destination are far away, as EB's network-based ellipse includes an increasing number of regions. In the extreme case where v_s and v_t are located in the furthest regions, it is possible that EB needs to receive all regions.

In this section we describe the Next Region method (NR), which avoids the above problem. The server again precomputes the shortest paths between all border nodes of different regions, but now the index keeps for each pair of R_i, R_j the identifiers of intermediate regions appearing in any shortest path between any of their border nodes. These regions are guaranteed to contain the shortest path from any node in R_i to any node in R_j.

Consider the example in Figure 5. The source is in region R_1, which has two border nodes. The destination is in R_{16}, which has a single border node. There are two shortest paths between the border nodes of R_1 and R_{16}. One of them traverses regions R_2, R_6, R_7, R_{11} and R_{12}, and the other regions R_5, R_6, R_7, R_{11} and R_{12}. The NR index records that any shortest path from R_1 to R_{16} may only pass through the union of the above region sets (shown gray in the figure).

NR does not place the whole index information in a single segment. Instead it distributes it in the broadcast cycle by forming local, region-specific indices, transmitted immediately before the corresponding regions. This eliminates the need for $(1, m)$ interleaving and keeps access latency low.

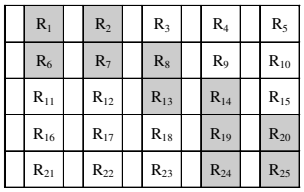


Figure 6: Needed regions

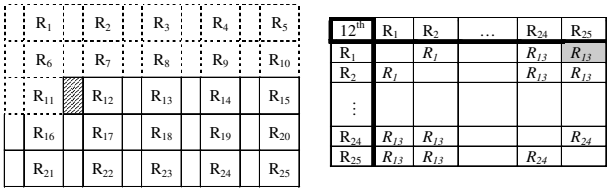


Figure 7: NR algorithm; receiving 12th local index

Figure 6 shows the structure of the cycle, where the unlabeled slot before each region corresponds to its index. The index of region R is an array with n rows and n columns, where n is the total number of regions. Every entry (R_i, R_j) of the array indicates the next region R_{nxt} in the broadcast cycle that may be needed for a shortest path from R_i to R_j. Note that R_{nxt} could be R itself.

Posed a query, the device tunes in the channel, receives the current packet, and waits until the subsequent index is broadcast. The client receives this index, and finds out what the next required region R_{nxt} is (depending on the source and target regions). It wakes up when R_{nxt} is broadcast and keeps listening until the immediately next local index is also received. From that index it determines the next needed region, and so on. When the latest index received indicates that R_{nxt} is a region that the client already possesses, listening stops and a (locally run) Dijkstra search computes the shortest path over all downloaded regions.

To exemplify, consider the broadcast cycle in Figure 6. The user wants to find the shortest path from a source in R_1 to a destination in R_{25}. The needed regions for this shortest path computation are shown in gray color, but the client does not know this in advance. Assume that the query is posed while R_{11} data are broadcast, which points the client to the 12th local index. The position of this index in the broadcast cycle is shown in the left part of Figure 7 and its contents in the right. The index suggests that R_{13} is the next needed region, so the device sleeps and wakes up to receive R_{13} and also the adjacent index (i.e., the 14th). The 14th index indicates that R_{14} is also required, so the client continues to receive data from the channel, until then 15th index points to R_{19}, as shown in Figure 8. The device sleeps until R_{19} is broadcast, and so on. The process continues this way until R_8 is received and the 9th index points to the already available R_{13}. Listening stops and the shortest path is computed locally.

Similarly to EB, the access latency in NR does not exceed one broadcast cycle. Regarding tuning time and memory requirements, NR is superior to EB, as the client listens only to a subset of the regions necessary in EB. The same holds for CPU time at the client (for the local execution of Dijkstra).

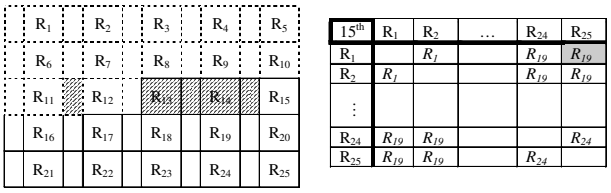


Figure 8: NR algorithm; receiving 15th local index

Pre-computation cost is identical to EB (assuming the same partitioning), as the same number of shortest paths among regions are computed. For an extensive empirical evaluation on real road networks the reader is referred to [14].

5. CONCLUSIONS

The continuing diffusion of mobile devices enables a flourishing market of location-based services, but also poses serious scalability concerns. This motivates the wireless broadcast model, where a server repeatedly transmits the entire database, while clients selectively tune in the communication channel and process their queries locally. In this paper we review spatial query processing in wireless broadcast environments and use shortest path computation as an example of specific challenges and solutions considered in that setting. Given the limitations faced in traditional query processing architectures and the ability of wireless broadcasting to alleviate many of them, it bears strong potential to serve as an alternative or a complement to common computation paradigms. It is therefore a promising research direction to extend the air indexing literature with solutions for other query types, be them of a spatial nature or otherwise.

6. REFERENCES

[1] Mobile World Congress: Network Breaking Point, February 17 2010.

[2] N. Beckmann, H.-P. Kriegel, R. Schneider, and B. Seeger. The R*-tree: An efficient and robust access method for points and rectangles. In *SIGMOD*, pages 322–331, 1990.

[3] C.-Y. Chow, M. F. Mokbel, and W. G. Aref. Casper*: Query processing for location services without compromising privacy. *ACM TODS*, 34(4), 2009.

[4] E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269–271, 1959.

[5] B. Gedik, A. Singh, and L. Liu. Energy efficient exact kNN search in wireless broadcast environments. In *GIS*, pages 137–146, 2004.

[6] A. V. Goldberg and C. Harrelson. Computing the shortest path: A* search meets graph theory. In *SODA*, pages 156–165, 2005.

[7] S. E. Hambrusch, C.-M. Liu, W. G. Aref, and S. Prabhakar. Query processing in broadcasted spatial index trees. In *SSTD*, pages 502–521, 2001.

[8] P. Hart, N. Nilsson, and B. Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE TSSC*, 4(2):100–107, 1968.

[9] Q. Hu, W.-C. Lee, and D. L. Lee. Power conservative multi-attribute queries on data broadcast. In *ICDE*, pages 157–166, 2000.

[10] T. Imielinski, S. Viswanathan, and B. R. Badrinath. Power efficient filtering of data an air. In *EDBT*, pages 245–258, 1994.

[11] T. Imielinski, S. Viswanathan, and B. R. Badrinath. Data on air: Organization and access. *IEEE TKDE*, 9(3):353–372, 1997.

[12] E.-S. Jung and N. H. Vaidya. An energy efficient MAC protocol for wireless LANs. In *INFOCOM*, 2002.

[13] I. Kamel and C. Faloutsos. On packing R-trees. In *CIKM*, pages 490–499, 1993.

[14] G. Kellaris and K. Mouratidis. Shortest path computation on air indexes. *PVLDB*, 3(1):747–757, 2010.

[15] E. Köhler, R. H. Möhring, and H. Schilling. Fast point-to-point shortest path computations with arc-flags. In *9th DIMACS Implementation Challenge - Shortest Paths*, 2007.

[16] K. Mouratidis, S. Bakiras, and D. Papadias. Continuous monitoring of spatial queries in wireless broadcast environments. *IEEE Trans. Mob. Comput.*, 8(10):1297–1311, 2009.

[17] K. Mouratidis and M. L. Yiu. Shortest path computation with no information leakage. *PVLDB*, 5(8):692–703, 2012.

[18] E. Müller, P. Kranen, M. Nett, F. Reidl, and T. Seidl. A general framework for data dissemination simulation for real world scenarios (demo). In *MOBICOM*, 2008.

[19] E. Müller, P. Kranen, M. Nett, F. Reidl, and T. Seidl. Air-indexing on error prone communication channels. In *DASFAA (1)*, pages 505–519, 2010.

[20] K. Park, M. Song, and C.-S. Hwang. An efficient data dissemination schemes for location dependent information services. In *ICDCIT*, pages 96–105, 2004.

[21] J. Pigg. Mobile backhaul: Will the levees hold?, 2009.

[22] Y. Tao, D. Papadias, and Q. Shen. Continuous nearest neighbor search. In *VLDB*, pages 287–298, 2002.

[23] J. Xu, W.-C. Lee, and X. Tang. Exponential index: A parameterized distributed indexing scheme for data on air. In *MobiSys*, 2004.

[24] J. Xu, B. Zheng, W.-C. Lee, and D. L. Lee. Energy efficient index for querying location-dependent data in mobile broadcast environments. In *ICDE*, pages 239–250, 2003.

[25] B. Zheng, W.-C. Lee, and D. L. Lee. Search k nearest neighbors on air. In *MDM*, pages 181–195, 2003.

[26] B. Zheng, W.-C. Lee, and D. L. Lee. Spatial queries in wireless broadcast systems. *Wireless Networks*, 10(6):723–736, 2004.

[27] B. Zheng, W.-C. Lee, and D. L. Lee. On searching continuous k nearest neighbors in wireless data broadcast systems. *IEEE Trans. Mob. Comput.*, 6(7):748–761, 2007.

[28] B. Zheng, W.-C. Lee, K. C. K. Lee, D. L. Lee, and M. Shao. A distributed spatial index for error-prone wireless data broadcast. *VLDB J.*, 18(4):959–986, 2009.

[29] B. Zheng, J. Xu, W.-C. Lee, and D. L. Lee. Grid-partition index: a hybrid method for nearest-neighbor queries in wireless location-based services. *VLDB J.*, 15(1):21–39, 2006.