



HAL
open science

DZI: An Air Index for Spatial Queries in One-dimensional Channels

Kwangjin Park, Alexis Joly, Patrick Valduriez

► **To cite this version:**

Kwangjin Park, Alexis Joly, Patrick Valduriez. DZI: An Air Index for Spatial Queries in One-dimensional Channels. Data and Knowledge Engineering, 2019, 124, pp.101748. 10.1016/j.datak.2019.101748 . lirmm-02386429

HAL Id: lirmm-02386429

<https://hal-lirmm.ccsd.cnrs.fr/lirmm-02386429>

Submitted on 29 Nov 2019

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

DZI: An Air Index for Spatial Queries in One-dimensional Channels

Kwangjin Park, Alexis Joly, and Patrick Valduriez¹

Wonkwang University, Korea, Inria, Montpellier, France, Inria, Montpellier, France

Abstract

The wireless data broadcast environment characteristics cause the data to be delivered sequentially via one-dimensional channels. A space-filling curve has been proposed for recent wireless data broadcast environments. However, air indexing introduces various problems, including the increase in the size of the index, conversion costs, and an increase in the search space because of an inefficient structure. In this paper, we propose a distribution-based Z-order air index and query processing algorithms suitable for a wireless data broadcast environment. The proposed index organizes the object identification (hereafter called ID) hierarchically only in terms of objects that are present. We compare the proposed technique with the well-known spatial indexing technique DSI by creating equations that represent the access time and tuning time, followed by conducting a simulation-based performance evaluation. The results from experimental show that our proposed index and algorithms support efficient query processing in both range queries and K-nearest neighbor queries.

Keywords: Moving objects, mobile computing, selective tuning, and spatial index.

1. Introduction

Most studies of Location-Based Services (LBS) technologies and services have been conducted with a server-centric approach. However, having all the work handled entirely by a server leads to problems such as high dependence of the client on the server, lack of consideration for moving objects, and privacy invasion [12][25-27]. Furthermore, query processing time can increase due to the overload of the server that processes individual clients' query requests. In particular, real-time information processing tasks for a large number of moving objects can put a heavy burden on the server [2] [9-11] [24]. Service delays due to server workloads in an environment where query targets and requesters move at the same time may result in wrong query results. To solve this problem, the workload of the server can be reduced by disallowing the server to access the client's individual queries.

In wireless data broadcast systems, popular information is periodically disseminated through one-dimensional communication channels (i.e. a channel in which information is sequentially delivered from a server) to mobile clients. The wireless data broadcast method is a one-to-many communication method that sends packets to the entire wireless network and is not affected by the number of clients [1-10][23][34][38]. In recent years, there

17 has been a renewal of interest in spatial data¹ delivery related to wireless data broadcast
18 environments [2] [9-11]. The wireless data broadcast method provides simple periodic
19 delivery of the target object (e.g., hotels, gas-stations, and taxis) to mobile clients. In other
20 words, the server and the client take on the roles and responsibilities of service processing
21 at the same time. For example, a user may issue a query through a mobile device (hereafter
22 called client), such as “give me the names and addresses of the hotels near my location.”
23 Then, the client downloads a spatial index through a wireless data broadcast channel from
24 a server. A spatial index contains information that indicates all locations of objects (e.g.,
25 hotels) near the client. The client can determine its current location via some positioning
26 technology, such as Global Positioning System (GPS). Finally, the client returns the final
27 query result to the user through the spatial query processing after confirmation of the
28 spatial index. In addition, the protection of privacy can also be supported by the wireless
29 data broadcast method [41], because the client’s individual position and the contents of
30 the query cannot be known. However, to answer queries, the clients have to stay in active
31 mode to receive the requested data objects and index information. Recently, a Distributed
32 Spatial Index (DSI) [13, 17] was proposed to improve the problem. The DSI is designed to
33 reduce access time by configuring the index into a distributed structure using a Hilbert-
34 Curve (HC). The DSI has the advantage of reducing probe wait² by adding distributed
35 pointers. However, the amount of data to send increases due to overlapping pointers so
36 that the server’s broadcasting periods become longer. In distributed spatial indexing, the
37 index information is increased by $N \times (\log_2 N + 2)$, where N be the number of data
38 objects. This means that the index size grows rapidly as the number of data objects
39 increases. A distributed spatial index allows a reduction in the tuning time via increasing
40 efficiency of selective tuning by the client. However, the increased index information that
41 is associated with it gives rise to increased latency time, which leads to a longer query
42 processing time. Therefore, index structures are necessary where overlapping pointers
43 can be removed while reducing probe wait for efficient query processing. In a previous
44 work [29], we have exploited a broadcast-based data dissemination scheme, called BBS
45 (Broadcast-Based LDIS Scheme). In BBS, the server periodically broadcasts the IDs and
46 coordinates of the data objects, without an index segment, to the clients. These broadcast
47 data objects are sorted sequentially, according to the locations of the data objects, before
48 being transmitted to the clients.

49 In this paper, we propose a distribution-based Z-order index suitable for spatial queries
50 in one-dimensional channels, as well as a technique in which the capabilities of the client’s
51 independent query processing have been enhanced. The proposed index does not configure
52 the index for all zones as the existing space-filling curve³ does, and indicates only the
53 area in which the required object is present, using a grid cell. Thus, the use of data

¹Spatial data is the data or information that identifies the geographic location of features and boundaries on earth, oceans etc.

²The average duration for getting to the next index segment is called probe wait [6,7]

³A space-filling curve is a continuous path that visits every space in a k -dimensional grid once without crossing itself.

54 caused by an unnecessary identifier can be reduced during index configuration. Moreover,
55 our proposed index does not give location identifier information for a region where the
56 object is not present. Therefore, information regarding moving objects can be efficiently
57 managed, because the location of moving objects can be determined with a small amount
58 of information.

59 The main contributions of this paper are as follows:

- 60 - We propose a lightweight spatial index that assigns a tree-based hierarchy ID (iden-
61 tification) only to objects that are present, and can manage the positions of moving
62 objects. The main difference between our index and the conventional index, such
63 as quadtree-based or grid-based index, is that the proposed index divides space and
64 assigns bit-codes only to those regions where objects exist.
- 65 - We propose breadth-first and depth-first spatial query search algorithms whose appli-
66 cation depends on the location distribution of the objects. The breadth-first provides
67 excellent for tuning time especially when the object is evenly distributed. On the
68 other hand, the depth-first provides excellent for access time by determining the final
69 query result by listening to the partial data information.
- 70 - We propose a data delivery model that is in accordance with the location distribution
71 of objects.
- 72 - We propose an algorithm for the continuous spatial query processing of moving ob-
73 jects that supports and facilitates the determination of the location of future objects,
74 using the client's own calculation.

75 The remainder of this paper is organized as follows. Section 2 discusses related work.
76 Section 3 discusses major performance factors and defines the system model. Section 4
77 describes the distribution-based Z-order index. Section 5 describes the data dissemination
78 and selective tuning algorithms. Section 6 describes an efficient method for the location of
79 moving objects. Section 7 evaluates the performance of distribution-based Z-order index
80 with DSI and BBS. Finally, Section 8 concludes.

81 **2. Related Work**

82 First, we discuss the data broadcasting methods. Then, we discuss the spatial index⁴
83 techniques for the wireless data broadcast environment.

84 *2.1. Data Broadcasting*

85 The server transmits the information that the client wants by identifying that infor-
86 mation in advance through a broadcast channel. For instance, advertisements, weather

⁴A spatial index is a special access method used to retrieve spatial data from within the data-store.

87 forecasts, traffic information, flight and train information, and other information can be
88 regularly sent to unspecified individuals [31].

89 In [14], the authors introduce a technique for delivering data objects to the clients in
90 asymmetric environments. In the environments, the network characteristics in downlink
91 direction is faster than those in uplink direction. In this technique, groups of pages, such
92 as hot and cold groups, with different broadcast frequencies are multiplexed on the same
93 channel. Then, the items stored on the faster disks are broadcast more often than those
94 items on the slower disks. In [19], the authors present the Broadcast on Demand (BoD)
95 model to provide timely broadcast according to requests from users. The goal is to maxi-
96 mize performance with respect to satisfaction of deadline constraints and achieve efficient
97 use of available bandwidth. In [20], the authors investigate how to efficiently generate the
98 broadcast schedule in a wireless environment. They consider the access pattern formed by
99 all the requests where the data dependency and access frequency can be represented by a
100 weighted directed acyclic graph (DAG). In [21], the authors observe that in general, clients
101 are divided into several groups, each one in a different location, with the members of each
102 group having similar demands. Then, they propose a mechanism that exploits locality
103 of demand in order to increase the performance of wireless data broadcast systems. In
104 [22], the authors define two optimization problems, such as MCDR (Minimum Cost Data
105 Retrieval) and LNDR (Largest Number Data Retrieval). MCDR aims at downloading a
106 set of data items with the minimum energy consumption, whereas LNDR aims at down-
107 loading the largest number of requested data items in a given time interval. In this paper,
108 authors consider data retrieval scheduling over multiple channels. In [32], authors address
109 the problem of answering KNN (K -Nearest Neighbor), range, and RNN (Reverse Nearest
110 Neighbor) queries in road networks via broadcast channels. Then, they propose ISW (index
111 for spatial queries in wireless broadcast environments)-index for spatial queries in wireless
112 broadcast environments. The ISW provides a pair of distance bounds, which is effective
113 for pruning the search space. The search space is reduced by using ISW and subsequently
114 the client can download as less as possible data for query processing, which can conserve
115 the energy. In [40], authors discuss the problem of spatial keyword query processing in
116 road networks in wireless broadcast environments. Then they provide algorithms for var-
117 ious queries, such as boolean range, top- k and ranked spatial keyword queries. In [33],
118 authors address the problem of processing CNN (Continuous Nearest Neighbor) queries in
119 road networks under wireless broadcast environments. Then, they propose a method to
120 construct and partition the NVD (Network Voronoi Diagram) structure of the underlying
121 road networks to derive an NVD quadtree and then transfer the tree into a linear sequence
122 of data packets. In BBS[29], the data objects broadcast by the server are sequentially
123 ordered based on their locations. Therefore, it is not necessary for the client to wait for
124 the index segment if the desired data object is able to be identified before the associated
125 index segment has arrived. In BBS, the structure of the broadcast affects the distribution
126 of the data object. BBS provides the fast access time since no index is broadcast along
127 with the data and, thus, the size of the entire broadcast cycle is minimized. The techniques
128 discussed above focus on efficient data transfer by analyzing users' requests. However, most
129 of them do not consider spatial queries, as we do in this paper.

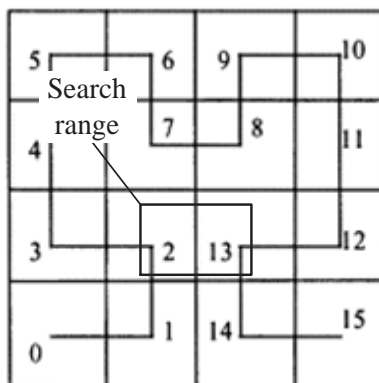


Figure 1: Access time with HC value for range query.

130 *2.2. Spatial Index on Air*

131 DSI has a distributed structure that mixes multiple search paths into a linear index
 132 structure that is distributed into the broadcast cycle. In DSI, a pointer of each data
 133 instance is repeated as many times as the number of entries of an index in a broadcast cycle
 134 to facilitate multiple search paths. Access time represents the period of time elapsed from
 135 the moment a user issues a query to the client to the moment when the required data item is
 136 received by the client. In each frame of the index, a pointer that is increased exponentially
 137 is stored; its role is to support fast access and selective tuning, i.e. mobile devices only
 138 selectively tune into broadcast channels when required. HC-based index techniques set
 139 up the value of the HC order based on the distribution of the object, and assign an
 140 identification number to every vertex of the basic curve, regardless of the presence or
 141 non-presence of the object. In this case, because the HC order does not represent a 1-1
 142 mapping of the order of data delivery, an object mapping process is required for calculating
 143 the object position after confirmation of the HC-value. In the process, the conversion time
 144 of $O(b^2)$ is consumed, with b denotes the number of bits needed for the coordinates of
 145 the object. Also, in case the objects are located close to each other, the value of the HC
 146 of order, which is used to distinguish the position of the object, increases. As a result,
 147 many IDs are required, even though the total number of objects is not large. Furthermore,
 148 because of the nature of the HC, problems may occur, depending on the location and scope
 149 of the query. An increased search space is needed in order to navigate through unnecessary
 150 blocks. For example, as can be seen in Figure 1, the client can obtain the final results only
 151 after waiting until objects 2 to 13 have been transmitted from the server for range query
 152 processing. The AT (access time) and TT (tuning time) will be increased further if the
 153 number of objects is to be increased and the query processing zones include more objects.
 154 In the KNN query processing, more serious problems will occur, depending on the value
 155 of ‘ K ’.

156 The problem with the HC-based distributed index can be summarized as follows:

- 157 - Even if a pointer is used that is exponentially increased by the exponential base e ,
- 158 when the number of the total data equals N , the number of the pointers will increase

159 as much as $\text{Log}_e N$ in each index frame when N increases. As a result, the size of the
160 index will be increased, and the amount of information that needs to be listened to
161 will increase significantly.

- 162 - Even if the existing index structures based on HC have hierarchical structures, in
163 order to obtain the desired results, indexes should be sequentially read from the
164 beginning of the exploration to its end. Tuning time may be reduced by proposing
165 a distributed index structure consisting of overlapping pointers for selective tuning,
166 but the increased index information lengthens the exploration time.
- 167 - In order for the object to confirm the location information, additional mapping, a
168 conversion process for the HC value, and the location calculation of the object are
169 required.

170 In [2], the authors propose an air indexing structure for supporting moving data ob-
171 jects. The goal is to reduce the power consumption and response time at the client side.
172 They design algorithms for snapshot and continuous queries, over static or dynamic data
173 with a regular grid. They apply other space-filling curves such as Peano curve for their
174 index. However, this technique also presents the same structural problems (inefficient
175 selective tuning) as the Hilbert-curve-based index. In [36], authors argued that bitmap-
176 based indexing can be highly effective for running range query workloads on spatial data
177 sets. Then, they introduced a compressed spatial hierarchical bitmap, called cSHB, index
178 structure that takes a spatial hierarchy and uses that to create a hierarchy of compressed
179 bitmaps to support spatial range queries. In [42], authors propose a multi-leveled air in-
180 dexing scheme in non-flat data broadcasting (MLAIN) to process window queries with the
181 popularity of spatial data items. MLAIN partitions the data space by recursively sub-
182 dividing it into four quadrant cells of multiple levels until all cells satisfy the constraint
183 that the number of data items in a cell is not over a specified number. In MLAIN, each
184 cell having one or more hot data items, called a hot cell, is broadcast more frequently
185 than regular cells to help clients quickly access hot data items on the channel. In [35], we
186 proposed the Hierarchical Grid Index (HGI) to remedy the Hilbert-curve-based index. For
187 the HGI, the location of the object is expressed by dividing grids until one grid contains
188 one object. Because grid division and index ID distribution only occurs in the area with
189 the object, the index pointer size can be reduced along with the index deciphering time
190 and TT. The delivery order considers an object's location and delivers it using the grid
191 recognizers' order to be awake, which occurs only when information close to the query
192 point is received, and to process the final query results. However, to confirm the object's
193 location information, coordinate information from the lowest level grid must be confirmed.
194 [37] explores the problem of spatial query processing in road sensor networks by means
195 of wireless data broadcast. The authors present a method to partition the record-keeping
196 information about the underlying road sensor network and its associated objects, by a
197 distributed air index, called integrated exponential index, based on an extended version
198 of the Hilbert-curve. They also propose client-side algorithms to facilitate the processing
199 of spatial queries. In [38], the authors propose a skewed spatial index considering clients'

200 skewed access patterns in the non-uniform wireless broadcast environments. The index
 201 information considering the non-uniform broadcast is interleaved with the spatial objects
 202 on the wireless channel to support efficient access. In [39], we examine the problems expe-
 203 rienced in the existing wireless broadcast environment and propose a novel system model
 204 and index technique. Instead of receiving spatial data information from the server, the
 205 client in the proposed system environment receives spatial data for part of the area from
 206 a sub_server to reduce the query processing time. In this manner, the model selectively
 207 tunes to the data located within the partial area most suitable for the query.

208 In parks or train stations, smartphone users may utilize location-based advertising
 209 messages to find information about nearby restaurants, hotels, and shops. If this makes
 210 their smartphone be flooded with a burst of advertising messages, they are likely to turn
 211 off advertising messaging services. In order to overcome the above problems, an index
 212 structure is required that gives the identification number by considering only the location
 213 of the objects actually present. In addition, fast and efficient algorithm for supporting
 214 spatial queries in wireless data broadcast environments should be developed.

215 3. Preliminaries

216 In this section, we examine the main problems affecting the performance evaluation.
 217 Then, we present the system model.

218 3.1. Problem Statement

219 In a wireless data broadcast environment, the increase of ‘index size’ and ‘the irregular
 220 delivery order of data’ increase AT and TT at the client side. First, let us consider the
 221 impact of the increase of index size on AT and TT. The conventional indexing schemes
 222 based on a space-filling curve, such as DSI allocates the index IDs to all zones even there
 223 is no object, thereby increasing the index size. This gives rise to the following problems:

- 224 - AT increases as the amount of data that is delivered during a single broadcast cycle
 225 increases.
- 226 - TT increases as the amount of data that needs to be read and processed by the client
 227 increases.

228 As shown in Figure 2, the pointer size directly influences the wireless data broadcast
 229 cycle. Figure 2(a) shows a single broadcast cycle in cases where overlapping pointers are
 230 not used and Figure 2(b) shows a single broadcast cycle in cases where overlapping pointers
 231 are used in the distributed index to reduce probe wait time. Let us assume that a logical
 232 time unit of both data object and index pointer be 1. Let N be the total number of
 233 objects. In distributed index, which uses overlapping pointers, the number of pointers
 234 increases by $\log_e N$ as N increases [11]. Let the value of N be 9 and e be 2. Then, each
 235 data object contains 4 pointers in distributed index. As a result, the total logical time of
 236 a single broadcast cycle for non overlapping pointers is 9, whereas distributed index is 36
 237 (9×4). Figure 2 shows an example which demonstrates the impact of adding the forwarding

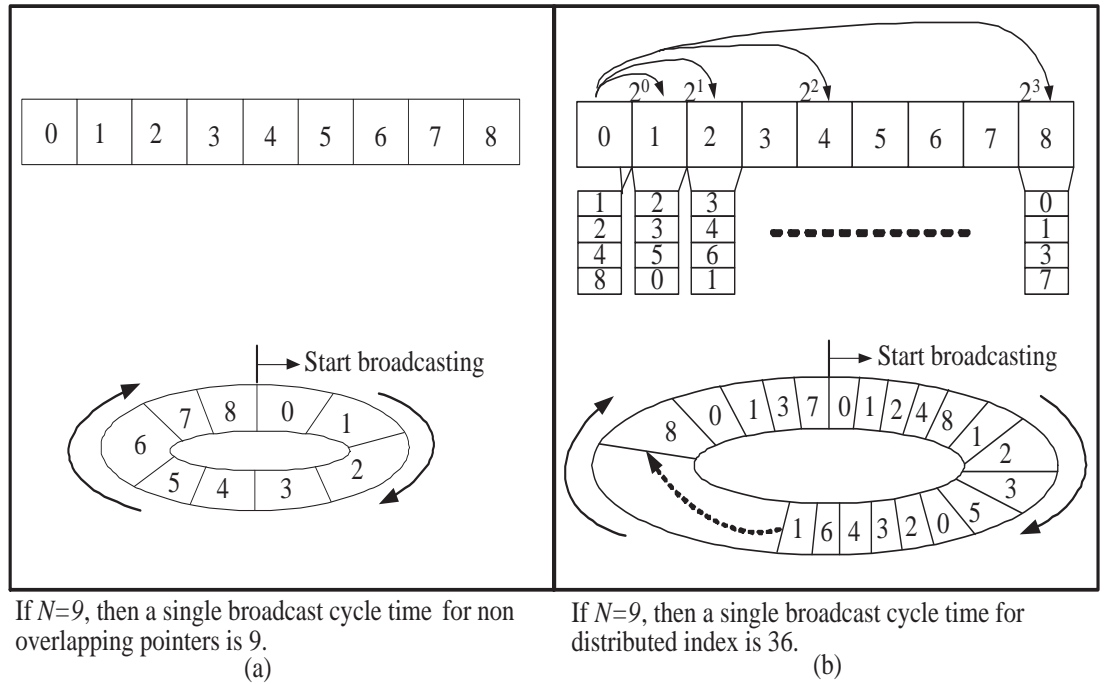


Figure 2: Access time according to the size of index pointers.

238 pointer to the existing spatial index on the broadcast cycle. The total logical time can be
 239 changed depending on the parameter value of e and N and the applying algorithms.

240 Second, let us consider the impact of the irregular delivery order of data on AT and TT.
 241 Let $index_data$ be the size of an index that represents objects. If the delivery order ignores
 242 the location of the object, the query result can only be confirmed after the locations of
 243 all objects have been confirmed at the time of spatial query processing. Therefore, if the
 244 user data is delivered along with the index data, it becomes $AT = N \times index_data$, $TT =$
 245 $N \times index_data$. On the other hand, about half of the ordinary AT and TT are needed
 246 when data are delivered according to a certain predefined rule. Therefore, we get $AT =$
 247 $\frac{N \times index_data}{2}$, $TT = \frac{N \times index_data}{2}$.

248 We then define the problem as follows:

249 Input

- 250 - Locations of spatial objects
- 251 - Directions of spatial objects
- 252 - Velocities of spatial objects

253 Output

- 254 - Spatial index

255 Objective

- 256 - Minimize *index_data*
- 257 - Minimize TT and AT
- 258 - Provide client's self-process for spatial query of a mobile object

259 In this paper, we aim to provide a lightweight spatial index that allows fast and efficient
 260 data access and tuning by delivering objects in a certain ordered fashion. In addition, the
 261 proposed index exploits the location, direction, and velocity of mobile objects to enable
 262 self-spatial query processing at the client.

263 3.2. System Model

264 Our system model can be summarized as follows:

265 - **Server:** The server receives reports regarding position, direction, and speed information
 266 from the moving object, and configures the index based on these reports. It then delivers
 267 the index, together with the user data, through the wireless data broadcast channel.

268 - **Client:** It is the user terminal that requests the service to a server. The client, which
 269 receives the user query, takes the following actions: (i) tries to listen to the index through
 270 the wireless data broadcast channel in wake-up mode; (ii) reads the index and is set to sleep
 271 mode after confirming all the information needed, and at what point it will be delivered;
 272 and (iii) wakes up selectively at the time of information delivery, and then tunes, after
 273 which it delivers the final result to the user.

274 - **Objects:** This refers to fixed objects (e.g., department stores, gas stations) or moving
 275 objects (e.g., people, taxis). These objects deliver information regarding their location,
 276 direction, and speed to a server in the jurisdiction area. In this paper, we assume that the
 277 points-of-interest belongs to the same types (e.g., gas stations, restaurants.) of queries.

278 - **Data:** Data can be classified into 'index data' and 'user data'. **Index data** includes
 279 the pointer for the object, its position, direction and speed, delivery time, etc. The index
 280 data includes the arrival time information of the adjacent index. If we assume that the
 281 client wakes up at random times, and if the first data that it listens to is the index data, it
 282 confirms the arrival time information of the adjacent index, and then waits again in sleep
 283 mode until the adjacent index arrives for power conservation. On the other hand, the **user**
 284 **data** represents specific information about the object (e.g., photos of goods, prices and
 285 maps in a department store).

286 Let N be the total number of objects, d be the user data size, and p be the index
 287 pointer size.

288 - **Channels:** These are wireless data broadcast channels through which the server delivers
 289 the information, and the uplink channel through which the object conveys information
 290 regarding its location, direction, speed, etc. In this paper, we assume that spatial indexes
 291 and data be transmitted through a single wireless channel.

292 We assume that a server collects the locations of hotels, gas stations, and certain
 293 moving objects and manages them integrally. The client and object can identify their
 294 locations through the use of location measurement technologies such as GPS. The data

Table 1: Definition of Notations.

Notation	Description
query point	The location where the query is issued
K	The closest K points to a query point
$bcast\ n$	A broadcast cycle n
G_i	A grid node i which contains at least one object
O_i	An object i
QKNN	K -nearest neighbor query
t	The time stamp (time interval)
N	The total number of data objects
e	the exponential base ($e > 1$)
w	The quantity of the objects that is kept in the queue
N'	The tuned total number of data
f	The total index tuple size of the next pointers in each index table
h	The Hilbert-curve order
D	The size of data
T_{e1}	The total size of the exponential pointer during a broadcast cycle
\mathfrak{R}	The number of objects that are contained in the grid nodes
p^n	The child node of the root grid node R_n
z^n	The number of grid nodes that belong to R_n
GID	The grid identifier included in R_n
z'^n	The number of grid-nodes included in the R_n

295 that is delivered from the server is delivered at constant cycle intervals according to the
296 rules given, and the client wakes up at random times.

297 Table 1 summarizes the notations used in this paper.

298 4. Distribution-based Z-order Index

299 The main performance factors we consider are AT and TT. These two factors are
300 affected directly by the distribution and the number of objects, index size, and the order
301 of delivery. We can summarize the conditions to be considered in this regard as follows.

- 302 - The client, without having to listen to the entire index data in order to handle spatial
303 queries, can reduce AT.
- 304 - Having the client quickly determine the portion of the index information that does
305 not need to be listened to can reduce TT. The client wakes up only when the needed
306 data is delivered by identifying the part that is not a query target, and then tuning
307 to the index data and user data.

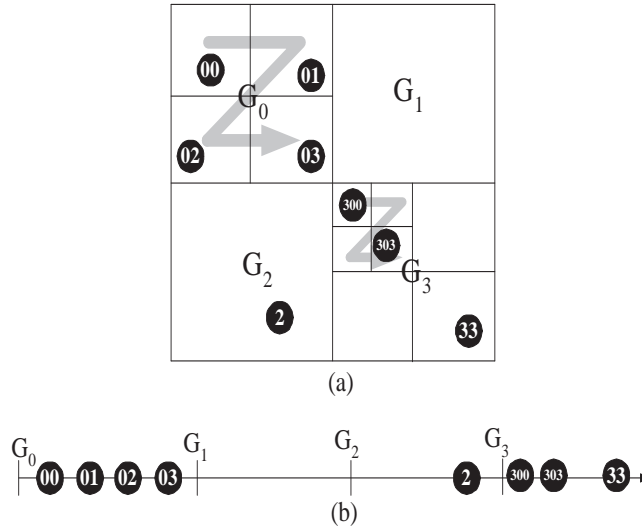


Figure 3: DZI index structure: (a) partitioning a region using a grid, (b) fit in one-dimensional environment.

- The reduction in the size of the index, which is composed of information regarding the object location and delivery time, can reduce AT and TT at the same time.

In order to satisfy these conditions, we can simplify the index data and user data to fit them in a one-dimensional environment.

In the proposed Distribution-based Z-order Index (hereafter called DZI), the index structure is determined on the basis of the location and distribution of the objects, and the server periodically rebuilds the index so that it reflects the mobility of the objects. DZI represents the locations of the objects in a grid structure. The order of delivery of an object is determined by the order of the numbers in the grid. The major different between DZI and Hilbert-curve-based index is that the Hilbert-curve-based index must allocate bit-codes to represent the whole regions regardless of whether there exists an object, whereas DZI divides space and assigns bit-codes only to those regions where objects exist. Thus, DZI supports to reduce the pointer size and search cost.

A grid number is assigned in the order of the character ‘Z’ of the alphabet by dividing the total area into four divisions. Each number in the grid is used as the identification number that represents the location of the object.

DZI is based on the assumption that objects are delivered in order of ‘Z’. However, AT and TT are not influenced even if objects are delivered in other kinds of order-e.g., delivered in order of ‘N’.

The basic structure of DZI is similar to a quadtree. However, the quadtree is an appropriate structure for a disk-based index, where the server receives and processes the clients’ requests in an on-demand environment. On the other hand, DZI is an appropriate structure for an air-index, where the server transfers the broadcast programming information and the actual broadcast contents in sequence via wireless broadcast channels and the clients use the received index data to selectively tune to the necessary broadcast [35]. That is,

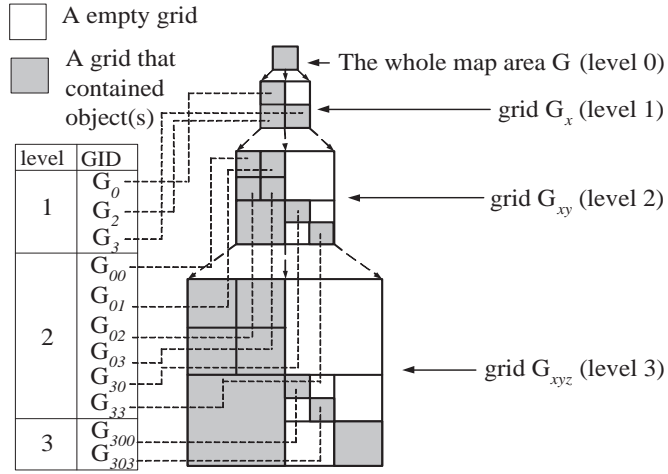


Figure 4: DZI hierarchy and its ID.

333 DZI provides ID only to objects that are present that enables more powerful accesses with
 334 a lightweight spatial index that allows fast and efficient data access and tuning.

335 We note that it is not necessary to wait until the beginning of the next broadcast
 336 cycle even though a client loses some data since grid ID represents its location similar
 337 to the point number of HC-based index [13]. In the DZI, the partitioning process varies
 338 according to the relative locations and distribution among the objects (see Figure 3(a)).
 339 The biggest difference between the existing HC-based index and a grid-based index is
 340 that the identification number of the DZI is granted only to objects that are present as
 341 targets. The search costs for index, mapping, and conversion can be reduced by reducing
 342 the amount of unnecessary ID information. This actually avoids having pointers on empty
 343 regions and reduces the index size. Then, the client can confirm an object's location using
 344 only its grid identification number, which has a hierarchical structure; in this way, the
 345 client is able to perform selective tuning. Another feature of DZI is that the index pruning
 346 ability is enhanced by means of a vertical hierarchy at the time of index browsing. The
 347 division of the grid repeats the division into four parts, depending upon the distribution or
 348 location of an object, and divides the parent nodes (top grid) into child nodes (sub-grid)
 349 until it contains only the last single object. In this scheme, the parent node becomes G_0 ,
 350 G_1 , G_2 , G_3 , after which the four nodes will again have child nodes. For example, let us
 351 assume that the G_0 node has objects in all four regions; they will then have child nodes
 352 of G_{00} , G_{01} , G_{02} , G_{03} . Let us assume that the G_{00} node has all the objects in the four
 353 division regions; they will then again have child nodes with identification numbers of G_{000} ,
 354 G_{001} , G_{002} , and G_{003} . Consider Figure 4 and the object status on the G_0 , G_1 , G_2 , G_3 grid:

- 355 - G_0 : only one object is present in all four grids, including G_{00} , G_{01} , G_{02} , and G_{03} .
- 356 - G_1 : none of objects is present.
- 357 - G_2 : a single object is present.

358 - G_3 : one of each object is present in three different grids: G_{300} , G_{303} , and G_{33} .

359 This object status is expressed in one-dimensional structures shown in Figure 3(b). The
360 structure of the tree is affected by the distribution of the objects. For example, a child
361 node is not blindly divided into four grids. Rather, if the objects are distributed evenly,
362 the tree will have parallel balance, and if the objects are located in a particular area of
363 focus, they will exhibit an imbalanced appearance. Figure 4 shows the DZI hierarchy and
364 ID. As the lower down object goes in the lower levels, the GRID (Grid ID) number increases.
365 The client can read the GRID number and the number of digits and infer the search area
366 and the range.

367 DZI is composed of root, intermediate and leaf entries. We can now define the parent
368 and leaf of the DZI as follows.

369

370 **Definition 1.** (*DZI Parent*) Parent's grid ID represents the location of the grid on a map.
371 The parent node contains a pointer of the child node(s) that represents the delivery time
372 of each child node. The parent node has four child nodes and above two child nodes of the
373 parent node have an object or at least one child node of the parent node has above two ob-
374 jects. That is, the division process only occurs to the grid node which has above two objects.
375

376 **Definition 2.** (*DZI Leaf*) The leaf node contains a pointer of the object (user data) which
377 represents the delivery time of each object.

378

379 The DZI division satisfies the following property:

380

381 **Property 1.** (*DZI division*) In a given set of data regions, a grid G_i repeats the hierarchy
382 division process until it contains only one object. A parent grid can have four or none
383 immediate child grids, and a child grid will have one parent grid.

384

385 The number of objects contained in a grid G_i in the DZI satisfies the following property:

386

387 **Property 2.** (*Number of objects inside G_i*) Let $num_obj_G_i$ be the number of objects in-
388 side G_i , which has child nodes. Let $num_obj_G_i'$ be the number of objects inside G_i' , which
389 has no child node. Then, $num_obj_G_i \geq 1$ and $num_obj_G_i' \geq 0$.

390

391 The main components of an index segment are: the next broadcast start time; the first
392 delivery data arrival time, according to the level (or grid ID) number; user data arrival
393 time, object's x - y coordinates, and the number of objects within the grid. Figure 5 shows
394 the detailed structure of the index data for one complete broadcast cycle. At the beginning
395 of the index, it contains information that indicates the starting time of each block. We
396 assume that the delivery order of object O_i is delivered sequentially according to the grid
397 numbers.

Information indicating the starting time of each block	Next broadcast cycle time	Arrival time for the first data item of each level (or grid id)	Arrival time for user data	Grid IDs	Number of objects inside grid _{<i>i</i>}	Details of object
--	---------------------------	---	----------------------------	----------	---	-------------------

Figure 5: A detailed structure of the index data.

Parameters:
 G_i : A grid node which contains at least one object
 S : A set of objects
Input: Object location, speed, and direction.
Output: Requested object

- 1: **for** each *bcast*
- 2: Disseminate the first data transfer time information for each level
- 3: **for** each level according to the top-down structure of DZI
- 4: **for** each grid G_i sorted by Z-ordering
- 5: Disseminate *ID* of G_i and num_obj_ G_i (step 1)
- 6: **if** G_i doesn't have a child node (step 2)
- 7: Put ' mark
- 8: **for** each object $O_i \in S$ by Z-ordering // bottom level of DZI
- 9: Disseminate location, moving direction, and velocity of O_i (step 3)

Figure 6: Data Dissemination algorithm for BFD.

398 5. Data Dissemination and Selective Tuning on Air

399 The server configures DZI in the form of a snapshot, based on the locations of objects
400 within the service region, which are delivered periodically. The lowest level of the index in
401 the configuration of a hierarchical grid node includes information regarding an object, i.e.,
402 its coordinates, speed of movement, and direction. The information regarding an object is
403 updated and delivered in each cycle. In the DZI at the time of spatial query processing,
404 because the objects are delivered from the server after being divided into a particular order
405 sequentially and are hierarchically based on the location, they can selectively tune to only
406 the objects in the region that they want to listen to. Our method supports selective tuning
407 to the index with reference to each object's position, orientation, and maximum speed, not
408 only in the case of a selective index tuning to static objects but also in the case of continuous
409 spatial query processing of moving objects. To support efficient query processing, we divide
410 our delivery method into BFD (breadth-first-delivery) and DFD (depth-first-delivery).

411 5.1. Breadth-First-Delivery

412 BFD delivers nodes in order from the top level of the index tree to the lower levels.
413 BFD progresses by delivering the first root node of the DZI that appears and thus going
414 deeper and deeper until a leaf node is delivered. The algorithm of BFD is shown in Figure
415 6. Please refer the following steps. The server in each level unit delivers the ID of G_i
416 and the number of objects inside of G_i (step 1). If G_i no longer contains a child node, it
417 inserts an apostrophe (') and informs that the grid is not divided any further (step 2). At
418 the lowest level, the details of an object, i.e., its *x-y* coordinates, speed, and direction, are
419 delivered (step 3). The information regarding how many child nodes the top node has, and
420 the number of contained objects, can be obtained and considered at the time of *KNN* query

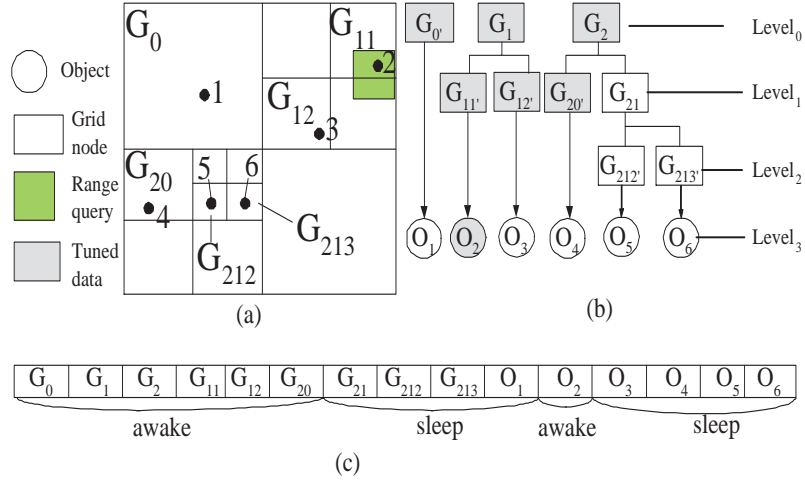


Figure 7: Breadth-First-Delivery: (a) objects and hierarchical grid division. (b) grid tree structure. (c) data listening.

421 processing. Because BFD has to listen to the data at the lowest level for the last query
 422 processing, the efficiency of AT is lower. However, it is possible to listen selectively by
 423 pruning the part that may not need to listen to by checking the top node first. Therefore,
 424 BFD provides the efficiency of TT; in particular, the object is evenly distributed and the
 425 efficiency of the search is excellent. We note that, in BFD, AT might decrease when the
 426 object data and the index data are delivered at the same level. However, in this case,
 427 additional information regarding the arrival time of object and index data deliveries at
 428 each level is required, which can lead to increase both AT and TT. Thus, our indexing
 429 scheme assumes that there is a distinction between the object data delivery and the index
 430 data delivery in BFD.

431 In Figure 7, let us look at the process of spatial query processing that uses BFD. Let
 432 us assume the range query for ease of understanding. In Figure 7(a), G_1 is included in
 433 the query range (a possible object for the answer of range query). The grid and the circle
 434 indicated in light gray denote the data that the client needs to listen to. After listening to
 435 level 0, the client finds that there are two objects in G_1 (see Figures 7(a) and 7(b)). After
 436 verifying up to the G_{20} at level 1, the client can find each of G_{11} and G_{12} in G_1 with an
 437 object within. The query range is set to include G_{11} and G_{13} , and no object is present in
 438 G_{13} . The client is switched to sleep mode after listening as far as G_{20} in level 1, and wakes
 439 up at the time of object delivery of level 3. It then obtains the object for O_2 , and returns
 440 the final query result. In Figure 7(c), when a logical time unit is denoted by 1, TT is 7,
 441 and AT is 11.

442 Table 2 shows the order of delivery according to the level of BFD. The client's *KNN*
 443 algorithm for BFD (see Figure 8) is as follows (the range query algorithm is omitted because
 444 it is similar and simpler to that of *KNN*). Please refer the following steps.

445 The client confirms the number of objects that are included in the ID of nodes G_i and
 446 G_i for each level units (leaf nodes are excluded) (step 1-step 3). If G_i belongs to the *KNN*
 447 query range (a possible object for the answer of *KNN* query) when the query point and

Table 2: The order of delivery of the index data and the user data according to level of BFD.

Level No.	data
0	$G_0(1'), G_1(2), G_2(3)$
1	$G_{11}(1'), G_{12}(1'), G_{20}(1), G_{21}(2)$
2	$G_{212}(1'), G_{213}(1')$
3	$O_1, O_2, O_3, O_4, O_5, O_6$

Parameters:
 KNN' : KNN candidate
Input: Index
Output: Requested object

```

1: for each bcast
2:   for each level except leaf level (step 1)
3:     for each grid  $G_i$  in Z-order (step 2)
4:       read num_obj. $G_i$  (step 3)
5:       check whether  $G_i$  is located inside  $KNN'$  range or not
6:       if  $G_n$  is located inside  $KNN$  range
7:          $KNN' \leftarrow KNN' \cup G_n$  (step 4)
8:       else  $KNN' = KNN' - G_n$  (step 5)
09: for each object  $O_i \in G_n$  from leaf level, where  $G_n \in KNN'$ 
10:   check whether  $O_i$  is located inside  $KNN$  range or not
11:   if  $O_i$  is located inside  $KNN$  range
12:      $KNN' \leftarrow KNN' \cup O_i$ 
13:   else  $KNN' = KNN' - O_i$ 
14: return  $KNN$  objects, where object  $\in KNN'$  (step 6)

```

Figure 8: KNN algorithm for BFD.

```

Parameters:
 $G_i$ : A grid node which contains at least one object
 $S$ : A set of objects
Input: Object location, speed, and direction.
Output: Requested object
1: for each
2:   Disseminate the first data transfer time information for each grid
3:   for each grid  $ID$  according to the structure of DZI
4:     for each level according to the top-down structure of DZI
5:       Disseminate  $ID$  of  $G_i$ , num_obj. $G_i$  (step 1)
6:         if  $G_i$  doesn't have a child node (step 2)
7:           Put ' mark
8:     for each object  $O_i \in S$  by Z-ordering
9:       Disseminate location, moving direction, and velocity of  $O_i$ 
           (step 3)

```

Figure 9: Data Dissemination algorithm for DFD.

448 the distance of G_i are compared, it is included in the KNN candidate set (step 4). If G_i
449 does not belong to the KNN query range, it is excluded from the KNN candidate set and
450 the next G_i is checked. The client reads the object on the inside of G_i that is contained
451 in the KNN candidate set at the level of the last leaf, and returns the final KNN result
452 (step 5). It is important to notice that in the search process, G_i that belongs to KNN can
453 be changed continuously. In other words, the most distant G_i that has previously been an
454 element of KNN is excluded from KNN when a new G_i is included.

455 5.2. Depth-First-Delivery

456 The following DFD technique delivers information that belongs to the same grid, first.
457 The components of the index and the delivery order consist of the number of each grid
458 and delivery time, number of objects inside a grid, and object information (i.e., location,
459 speed, direction, etc.). The algorithm of DFD (see Figure 9) is as follows. For each grid ID
460 unit, the server delivers the number of objects that are included in the ID and G_i for node
461 G_i , which contains the objects (step 1). If G_i no longer contains child nodes, it informs the
462 client that the grid is not being divided any further by inserting an apostrophe (') (step
463 2). DFD delivers details about the object that belongs to the end G_i , which is not divided
464 further according to Z-ordering, the x - y coordinates of the object, speed and direction, etc.
465 (step 3). Because the division node and object information are immediately delivered by
466 the grid at the time of spatial query processing, query processing can be performed without
467 having to wait unconditionally until the lowest level, as in BFD. Therefore, compared to
468 BFD, AT gets much better. In particular, the performance gets better when the distri-
469 bution of objects is clustered in a particular area. Note that we assume that the index
470 information is delivered by a certain t time interval. Therefore, it is possible for the client
471 to predict when the desired index information arrives, because of the nature of the index,
472 which is delivered sequentially according to the order of the location at an interval of t
473 hours.

474 Let us consider the range query processing that uses DFD, keeping Figures 10(a) and
475 10(b) in mind. In the figure, the query range is shown to be located within G_1 . The grid

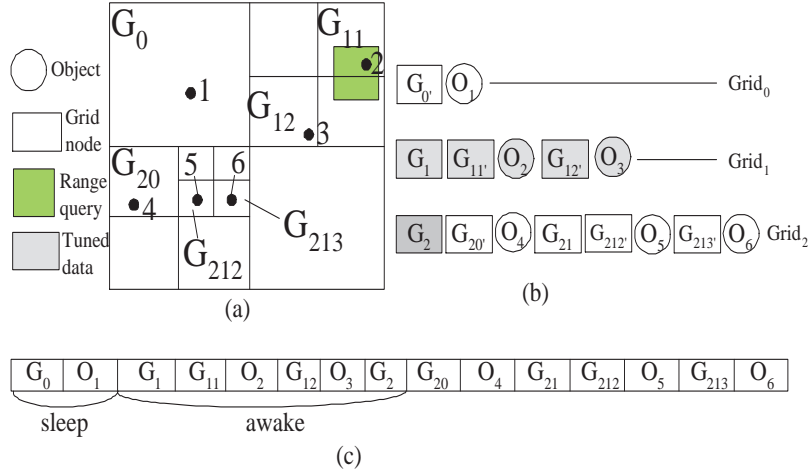


Figure 10: Depth-First-Delivery: (a) objects and hierarchical grid division. (b) grid tree structure. (c) data listening.

Table 3: The order of delivery of the index data and the user data according to GID.

Level No.	data
0	$G_0(1'), O_1$
1	$G_1(2), G_{11}(1'), O_2, G_{12}(1'), O_3$
2	$G_2(3), G_{20}(1'), O_4, G_{21}(2), G_{212}(1'), O_5, G_{213}(1'), O_6$
3	null

Parameters:
 KNN' : KNN candidate
Input: Index
Output: Requested object
1: **for** each $bcast$
2: **for** each grid id (step 1)
3: **for** each grid G_i in Z-order (step 2)
4: read $num_obj_G_i$ (step 3)
5: check whether G_i is located inside KNN range or not
6: **if** G_n is located inside KNN range
7: $KNN' \leftarrow KNN' \cup G_n$ (step 4)
8: **else** $KNN' = KNN' - G_n$ (step 5)
09: **for** each object $O_i \in G_n$, where $G_n \in KNN'$
10: check whether O_i is located inside KNN range or not (step 6)
11: **if** O_i is located inside KNN range
12: $KNN' \leftarrow KNN' \cup O_i$
13: **else** $KNN' = KNN' - O_i$
14: return KNN objects, where $object \in KNN'$ (step 7)

Figure 11: KNN algorithm for client DFD.

476 and circle in light gray indicate the data that the client needs to listen to. The client
477 checks the first data delivery time information of each grid. Next, the client wakes up at
478 the G_1 -level and listens to the data up to $G_2(3)$ in level G_2 , then finally checks to confirm
479 that objects O_2 and O_3 are in grid1, and that the object that belongs to the range query is
480 O_2 . In 10(c), when one unit of logical time is denoted as 1, TT is 6, which is the wake-up
481 time, and AT is 8, which is the *sleep time + wake-up time*. Selective tuning is possible
482 using the grid ID in the case of DFD. In this case, the grid that is the query result target
483 must be tuned to entirely until it finds the desired object. Thus, the amount of information
484 that the client must tune to in its waking state is increased according to the distribution
485 and the number of objects. In this case, more tuning time can be consumed than BFD.
486 Table 3 shows the order of delivery of the index data and the user data according to GRID.

487 The client's *KNN* algorithm for DFD (see Figure 11) is as follows (the range query
488 algorithm is omitted because it is similar to that of *KNN*). The client checks the ID of
489 node G_i and the number of objects that are included in G_i by each grid ID unit (step 1
490 - step 3). If G_i belongs to the *KNN* query range, which is determined by comparing the
491 distance between the query point and G_i , it is included in the *KNN* candidate set (step 4).
492 If G_i is outside the scope of the *KNN* query range, it is excluded from the *KNN* candidate
493 set, and then the next G_i is checked (step 5). The client reads the objects within G_i that
494 are included in the *KNN* candidate set, and configures the *KNN* candidate set by checking
495 if it is included in the *KNN* query range (step 6). It then returns the final results of *KNN*
496 (step 7).

497 In summary, BFD, which searches through the indexes at all levels, has a longer access
498 time than does DFD, which locates desired data objects via partial index information.
499 However, BFD yields better tuning time because it exploits high-level index information
500 to prune unnecessary search ranges beforehand. By contrast, DFD examines every data
501 object in each partitioned zone via a depth-first search, making its tuning time longer
502 than that of BFD. The server delivers the object location information periodically, using
503 the information regarding the object's location, direction, and speed. The client performs
504 spatial query processing by taking advantage of the object information that was delivered
505 from the server. The client wipes off the objects that are not taken into account by using the
506 information on location, direction, speed, and stores, etc., and manages only those objects
507 that will affect future query results. Therefore, this process can reduce the computation
508 time for updating and can reduce the dependence on the server. If the speed or direction
509 of an object exceeds the predicted scope within the range and time where it is desired to
510 be looked for, the server data of the next cycle must be tuned to.

511 When search ranges for *KNN* queries are updated by the client, the BFD and DFD
512 algorithms utilize the following two principles. First, when the K in *KNN* queries is
513 decreased, the existing query processing results are used without the need to listen to the
514 broadcast server. Second, when the K in *KNN* queries is increased, query processing is
515 repeated by once again listening to the servers broadcast channel.

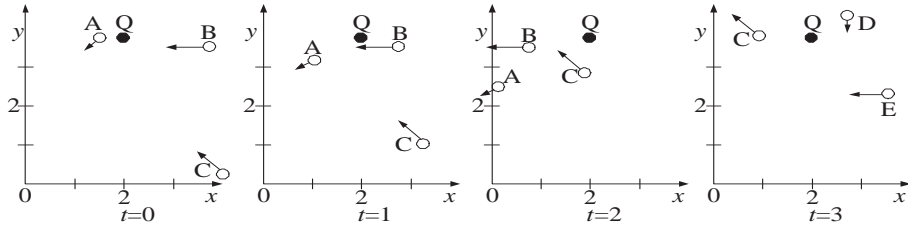


Figure 12: 3NN objects for a period of time T_0 - T_3 .

516 6. Client's Own Calculation for K NN Queries on Moving Objects

517 In this section, we examine a client's method to self-process a spatial query of a mobile
 518 object. For ease of presentation, we describe the query processing method focusing on
 519 snapshot K NN queries over static query point and dynamic objects. The client's query
 520 processing can be summarized as follows:

- 521 - The client performs its own future spatial query processing without having to rely on
 522 a server by receiving the information from the server regarding the object's location,
 523 speed, and direction of movement.
- 524 - The quantity of objects that is kept in the client's queue (w), and the objects that are
 525 at a greater distance than the information stored in the queue among the information
 526 transmitted from the server, are excluded from the K NN target.
- 527 - The point in time at which the time stamp t is changed is the point in time at which
 528 the target of K NN or the priority can be changed.

529 Figure 12 shows an example with $K = 3$ (3-NN) in the client's K NN query processing.
 530 In the figure, let us assume that object A and object D move at 0.5 increments of the
 531 maximum travel distance (with maximum speed) for the unit time t , and that object B,
 532 object C, and object E move at increments of 1 (with maximum speed). As can be seen
 533 in the figure, 3-NN is processed up to T_0 - T_3 using the object's information received from
 534 the server in the first cycle. In T_3 , new objects D and E have been added, with the object
 535 information received from the server in the second cycle, and objects B and A are excluded
 536 from the K NN results target. Table 4 shows the 1st NN, 2nd NN, and 3rd NN resulting
 537 objects according to T_n . The table shows that the client performs its own spatial query
 538 processing without having to rely on a server for a period of time, by using an index that
 539 has been delivered through a wireless data broadcast channel. The client is then passed
 540 through a wireless data broadcast channel using an index for a period of time, in order
 541 to perform its own query processing of spatial information. Let us assume that objects
 542 travel over a period of time in a straight line, like cars on the highway. The client's query
 543 processing can then be explained using the example in Figure 13. Observe the range queries
 544 and 3-NN processing by dividing the query processing time into logical time units of up to
 545 T_0 - T_3 . Assume that the server's broadcasting of object information of up to 1-4 to *bcast1*

Table 4: 3NN resulting objects according to T_n

Time stamp	1st NN	2nd NN	3rd NN
T_0	A	B	C
T_1	B	A	C
T_2	C	B	A
T_3	D	C	E

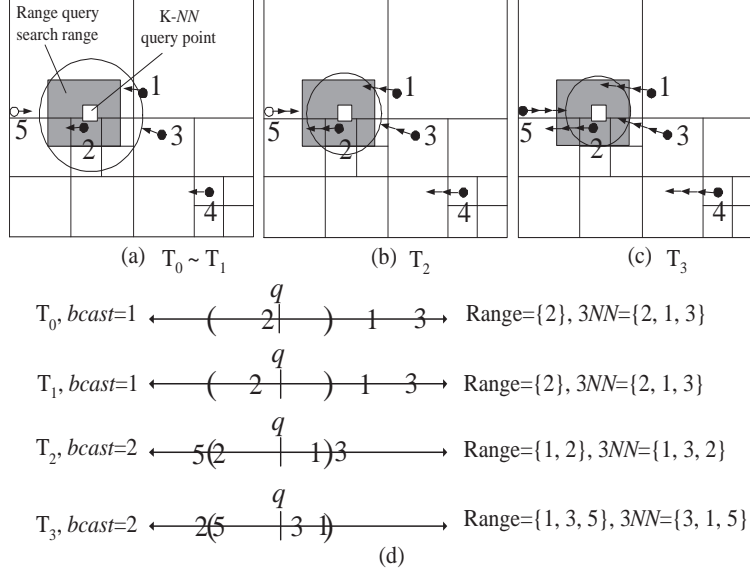


Figure 13: 3NN query processing for moving objects.

546 is the first broadcast cycle, and that it broadcasts the object information of 1-5, including
 547 new object 5, to $bcast2$. In Figure 13, the empty box represents the query point in the KNN
 548 query; the gray box, the range query; the black circles and arrows (e.g., O_1, O_2, O_3, O_4),
 549 the objects that have been confirmed for the location and direction from the server; and
 550 the empty circles with arrows (e.g., O_5), objects that have not yet been confirmed by the
 551 server. The objects such as cars, trains, bikes, and boats have the maximal travel speed.
 552 For example, a moving car may not exceed a speed of 300 km/h. An arrow represents
 553 the travel direction of an object, and the length of a line represents the maximum travel
 554 speed of an object, which increases by one scale every time T_n increases. The client can
 555 predict the object's location in advance at the time of $T_n + n$, with reference to the object's
 556 current location, movement speed, and direction of travel, since the logical time T_n . In
 557 Figure 13(d), the parentheses in a range of segments indicate the range query region, and
 558 the numbers in parentheses in the query results for 3NN are listed in order of the closest
 559 object. We assume that the client's queue can store a maximum of three objects, using w
 560 $= 3$.

561 Let us consider the snapshot of KNN query processing at the time T_0 - T_3 , considering
 562 the movement of the object. The KNN algorithm is as follows (the range query algorithm

Parameters:
t_Order: A period of time that the result set and its *KNN* order does not change.
t_Entity: A period of time that the result set does not change.
T_n: A logical time unit of client.
Input: index data which includes location, max_velocity, moving direction.
Output: Requested object
The value of *n* of *T_n* is initially set to 0
Initially *QKNN_result_set* = { \emptyset }

```

1: for each bcast
2:   read index (step 1)
3:   for each object  $O_i \in S$ 
4:     if  $O_i$  is located in a range of QKNN
5:       QKNN_result_set ← QKNN_result_set  $\cup$   $O_i$  (step 2)
6:     Find t_order, t_Entity (step 3)
7:     for each Tn
8:       if  $T_n \leq t\_order$  (step 4)
9:         QKNN_result_set remains the same
10:      else if  $t\_order < T_n \leq t\_Entity$  (step 5)
11:        for each object  $O_i \in S$ 
12:          if  $O_i$  is located in a range of QKNN
13:            QKNN_result_set ← QKNN_result_set  $\cup$   $O_i$ 
14:          else tune to the next bcast (step 6)

```

Figure 14: *KNN* algorithm for snapshot query.

563 is omitted because it is similar to that of *KNN*).

564 Let us now explain the *KNN* algorithm by applying it to the example in Figure 14. The
565 client listens to the index received from the server (step 1). If the location of the object
566 is included in 3NN, it is included in *Q3NN_result_set* (step 2). The client then looks for
567 *t_Order* and *t_Entity* with reference to the object’s location, direction of movement, and
568 speed, which have been listened to (step 3). The index that has been delivered by the
569 server in *bcast1* is then read, and the *Q3NN_result_set* = { O_2, O_1, O_3 }, which represents
570 the query results from T_0 - T_1 , ensures that it does not change (step 4). Thus, up to T_0 - T_1
571 can obtain the same 3NN query result without listening to the server’s index information
572 or confirming the distance between the query point and the object. In addition, the order
573 of the query results in T_2 is changed to *Q3NN_result_set* = { O_1, O_3, O_2 }, but we can see
574 that there is no object that can be included in the query result apart from objects $O_1, O_2,$
575 O_3, O_4 , which have been listened to in *bcast1* (step 5).

576 The reason for this is that O_5 is not included in the 3NN query results by T_2 , which have
577 not been listened to in *bcast1*. In Figure 13, we can see that the target of *KNN* and the
578 order of *KNN* at up to T_0 - T_1 do not change. Thus, the client can return the stored value
579 in the query result queue to the same *KNN* query result without the need for a separate
580 comparison calculation during the period T_0 - T_1 . We can also see that the target has not
581 been changed, even though the order of the objects for the *KNN* query result have been
582 changed at up to T_0 - T_2 . Thus, from T_0 - T_2 , the client can obtain *KNN* query results by
583 comparing the object’s location, direction of movement, and speed, which have been stored
584 in the queue, without having to rely on the broadcast data of the server. After T_3 , the
585 client listens to the *bcast2* information that is delivered by the server, because a new object
586 (see O_5 in Figure 13(c)) can affect the 3NN query processing results (step 6). It checks

587 the information regarding the new O_5 using *bcast2* information. In T_3 , the query result of
588 $Q3NN_result_set = \{O_3, O_1, O_5\}$ is obtained by additionally reflecting the information on
589 the location, speed, and direction of O_5 to the range query and 3NN. Let t_Order be the
590 time range from the present time until the query result target object and the KNN order
591 are not changed (T_0-T_1 of Figure 13), and t_Entity be the time range from the present
592 time until the query result target object is changed (T_0-T_2 of Figure 13). Thus, the client
593 maintains information about objects as much as is allowed within the range of the queue,
594 and is able to process spatial queries on its own by using t_Entity and t_Order , without
595 having to rely on the broadcast data of the server over a period of time.

596 In order to support client's self-process for spatial query of a mobile object, we propose
597 two heuristics, each defining the unchangeable time period in which the target object of
598 KNN does not change. Let us also assume that (1) objects that are present may disappear
599 and new objects may be added in M , which denotes all service regions, in every *bcast* of
600 the server, and (2) the server delivers the information regarding all present objects in M
601 in each cycle.

602
603 **Heuristic 1.** (*The unchangeable time period in which the target object of KNN does not*
604 *change*). Let us assume that the time point that has received the object via the most recent
605 *bcast* is T_{latest} , and that any future time point is T_{future} . Let A be the set of objects
606 included in KNN at T_{latest} (e.g., O_1, O_2 , and O_3 in Figure 13(a)), B be the objects located
607 outside of the scope of KNN as T_{latest} (e.g., O_4 and O_5 in Figure 13(a)), C be the the
608 objects to be included in KNN as T_{future} (e.g., O_1, O_3 , and O_5 in Figure 13(c)), and
609 D be the objects to be located outside of the KNN scope as T_{future} (e.g., O_2 and O_4 in
610 Figure 13(c)). t_Entity denotes the time range in which all objects included in A can also
611 be included in C . Thus, $A.D$ or $B.C$ in which B and D are included are excluded from
612 consideration as a t_Entity target. That is, $A.C$ (e.g., O_1, O_2, O_3 , and O_5 in Figure 13)
613 $\ni t_Entity$, which can be entirely included in the KNN range at T_{latest} and T_{future} .

614
615 If we assume that the time range of t_Entity time is 60 minutes according to Heuristic
616 1 and that the broadcast cycle is 20 minutes, the client can process KNN by itself, without
617 tuning to the data on the server for up to a maximum of three cycles.

618
619 **Heuristic 2.** (*The unchangeable time period in which the order of the target object of*
620 *KNN does not change*). Let E be the set of objects that do not change in the order of KNN
621 from t_Entity to T_{future} . Let F be the set of objects that change in the order of KNN from
622 t_Entity to T_{future} (e.g., O_1, O_2 and O_3 for a period of time T_0-T_1 in Figure 13). Let F
623 be the set of objects that change in the order of KNN from t_Entity to T_{future} . T_Order
624 defines the time range in which all objects that satisfy $A.C$ maintain the same in the order
625 of KNN. That is, it becomes $A.C.E \ni T_Order$.

626

627 If we assume that the time range of t_Entity is 60 minutes for Heuristic 2 and a
628 broadcast cycle of 20 minutes, and for T_Order for a cycle of 20 minutes, the client can
629 return the value stored in the query result queue to the KNN query result as it is for 20
630 minutes, without having to listen to the server’s data for a maximum of three cycles.

631 7. Performance Evaluation

632 BBS [29] and DSI [13] are both major indexes, which exploit the wireless data broadcast
633 environment’s characteristics for spatial queries. BBS provides excellent AT because it
634 delivers the data sequentially based on the object’s location, without separating the index
635 information. On the other hand, DSI is the most widely used index for the selective
636 tuning of clients in a wireless data broadcast environment. Unlike Broadcast Grid Index
637 (BGI) [2] is designed for continuous queries, both DSI and BBS are basically designed
638 for snapshot queries. Moreover, the BGI method is an extended version of the existing
639 DSI method that is suited for continuous spatial queries. Similarly to the previous spatial
640 indexing methods, BGI suffers from the fast-growing index size problem related to the index
641 overlapping pointer. Thus, in this paper, we compare our methods with DSI and BBS.
642 The client’s self-processing algorithm is not included in our experimental study. This is
643 because DSI and BBS, the main indexing techniques against which the proposed technique
644 is compared, do not contain the client’s self-processing algorithm. Instead of carrying out
645 an experimental study, we defined two heuristic methods (Heuristic 1 and Heuristic 2)
646 regarding the client’s self-processing algorithm, and we predicted its performance in the
647 assumed query processing conditions.

648 7.1. Performance Analysis

649 In this section, we analyze DZI and DSI. DSI uses distributed exponential pointers
650 from each data item. With the exponential pointer, each data object contains pointers
651 that point the IDs of the data items that will subsequently be broadcast. In section 7.1.1
652 and section 7.1.2, we assume that the average time taken to find a single object in either
653 KNN queries (i.e., $K = 1$) or range queries.

654 7.1.1. Access Time for DSI and DZI

655 Let us compare the access times of the DZI and DSI methods. Because DZI and DSI
656 algorithms have a distributed index structure, the probe wait time can be ignored. We
657 now compare the access time of DSI and DZI.

658 - DSI: let e be the exponential base e , N be the total number of data, and f be the
659 total index tuple size of the next pointers in each index table⁵, h be the Hilbert-curve
660 order, D be the size of data, T_{e1} be the total size of the exponential pointer during a
661 broadcast cycle, where $k = \log_e N$ and ID be the object ID. Then T_{e1} is obtained by:

⁵Each index table in DSI has the next pointers that increase exponentially within the range of N , the total r amount of data. For example, if $N=1024$, a single index table has 9 next pointers of 1, 2, 4, 8, and up to 1024.

$$T_{e1} = \sum_{f=0}^k N \quad (1)$$

662 Then, the access time for DSI is

$$AT_DSI = \frac{T_{e1} + N \times D + N \times h + N \times ID}{2} \quad (2)$$

663 Let \mathfrak{R} be the number of objects that are contained in the grid nodes that belong to
 664 the root grid R , p^n be the child node of the root grid node R_n , z^n be the number of grid
 665 nodes that belong to R_n , GID be the grid identifier included in R_n , and z'^n be the number
 666 of grid-nodes included in the R_n (because the number of leaf nodes in \mathfrak{R} is 0, it is not
 667 necessary to indicate it, and therefore, the lowest grid node is excluded from the number).
 668 Then, the access time for DZI is

$$AT_DZI = \frac{\sum_{R^n=0}^3 \sum_{p^n=0}^{z^n} GID + \sum_{R^n=0}^3 \sum_{p^n=0}^{z'^n} \mathfrak{R} + D \times N}{2} \quad (3)$$

669 7.1.2. Tuning Time for DSI and DZI

670 Energy consumption of the indexing techniques can be indirectly compared by mea-
 671 suring how long the client has to stay in active mode to acquire the desired data objects.
 672 Measuring the actual energy consumption requires consideration of various factors, such
 673 as the size of a single data item and the channel bandwidth. This paper simply uses the
 674 client's tuning time as a measure of energy consumption. We now compare the tuning
 675 time for DSI and DZI to prove the energy efficiency of our proposed algorithms during
 676 spatial query processing. While DSI uses the exponential pointer, which has a redundant
 677 structure, our algorithm using DZI allows selective tuning with the ID and the number of
 678 objects without the exponential pointer.

679 Let N' be the tuned total number of data and T_{e2} be the number of indexes from f'
 680 to k' , where $k' = \log_e N'$ which the client wakes up and listens to until the desired data are
 681 obtained. We obtain:

$$T_{e2} = \sum_{f'=0}^{k'} N' \quad (4)$$

682 Then, the tuning time (TT) for DSI:

$$TT_DSI = h + T_{e2} + D \quad (5)$$

683 Let L^n be the number of grid nodes in R_n up to the grid level n , until the query
 684 processing is completed and $L^{n'}$ be the number of grid nodes excluding the leaf grid node

685 in the R_n up to the grid level n , until the query processing is completed. Let L^m be the
686 beginning grid node for tuning, and $L^{m'}$ be the beginning grid node for tuning excluding
687 the leaf grid node.

688 Now, we have:

$$TT_DZI = \sum_{R^n=L^m}^{L^n} \sum_{p^n=L^m}^{L^n} GID + \sum_{R^n=L^{m'}}^{L^{n'}} \sum_{p^n=L^{m'}}^{L^{n'}} \mathfrak{R} + D \quad (6)$$

689 7.2. Performance Experiments

690 Let us now evaluate the efficiency of our algorithms through experimentation. We
691 implemented all methods in C++ and conducted experiments on a machine with a Pentium
692 3.16 GHz CPU and 3 GByte of RAM. We assume that the client’s mobility pattern follows
693 the Random Waypoint Mobility Model [28], which is widely used.

694 In our experiments, each object begins by pausing for a fixed number of seconds. Then,
695 object selects a random destination in the simulation area and a random speed between
696 0 and the maximum speed. The object moves to this destination and again pauses for a
697 fixed period before another random location and speed. This behaviour is repeated for the
698 length of the simulation. We now describe the ratio of energy consumption for these states.
699 A mobile client begins by staying in one location for a certain period of time t . We assume
700 two energy states: sleep and active mode. In a processor, the doze mode has extremely low
701 power consumption. We assume that the wireless data broadcast channel has a bandwidth
702 of 2 Mbps as the same applied in [2, 29]. We use a uniform dataset (hereafter called
703 $\mathcal{D}1$) with 50,000 points and a real dataset (hereafter called $\mathcal{D}2$) containing 39,231 data
704 objects of MBRs for Montgomery County roads⁶ (see Figures 15(a) and 15(b)). We assign
705 8 bytes for each pointer and 8 bytes for 2D coordinates. The default parameter setting
706 in our synthetic data set test is: number of objects = 10,000, size of data = 256 bytes,
707 exponential value $e = 2$, moving distance = 100m, the value of K for KNN search is 3,
708 and a search range for range query is 300m \times 300m. Assuming that the data objects are
709 distributed on a 10km \times 10 km area.

710 The total exploration time from when the client reads an index at a certain time point
711 after receiving a user’s request to obtain the desired final result is assumed as AT. The
712 total amount of time to be awake in order to obtain the desired final result is assumed as
713 TT.

714 First, we make a comparison between AT for BFD and DFD, according to the increase
715 in the size of the user data for KNN query. As can be seen in Figure 16, DFD has excellent
716 AT compared to BFD. The user data in BFD is delivered in batches after the index data
717 delivery. Thus, all the index information must be tuned to in order to listen to the user
718 data that will be delivered to the next order. Because the index search method also uses
719 BFD, all index information from the top of the tree must be tuned to first, in order to

⁶The dataset available at <http://chorochronos.datastories.org/?q=node/17>

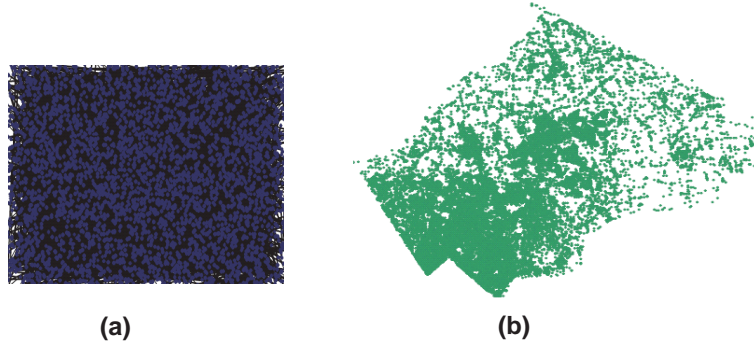


Figure 15: Datasets for Performance Evaluation. (a) UNIFORM dataset ($\mathcal{D}1$). (b) REAL dataset ($\mathcal{D}2$).

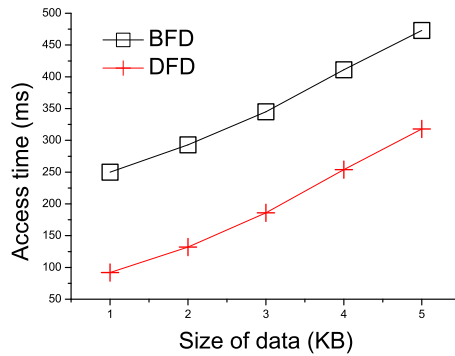


Figure 16: AT for the BFD and the DFD, according to the increase in the size of the user data.

720 access the sub-index information. The user data is applied to the same process. For this
 721 reason, the time consumed in order to obtain the desired information increases according
 722 to the increase in the index pointer and data sizes. On the other hand, DFD can obtain
 723 the final query result even in the status that involves only listening to the index from the
 724 upper part of the tree and the user data. The reason for this is that the index data and
 725 the user data are delivered one after another in DFD.

726 Figure 17(a) shows the results of the measurement of AT according to the increase in
 727 KNN . As the KNN value increases, BFD and DFD must navigate the index data and
 728 user data in a broader range. Thus, the search range and AT increase. In particular, we
 729 can see that BFD, which can confirm the final query result only after listening to all the
 730 index information, shows poor AT results as compared to those of DFD when the K value
 731 increases.

732 Figure 17(b) shows the evaluation of AT in response to an increase in the navigation
 733 area of the range query. Similar to the case of KNN , in BFD, AT increases sharply,
 734 because it must wait until all the index data has been transmitted in order to obtain the
 735 user data, and the number of user data required to be confirmed increases as the search
 736 region increases. On the other hand, DFD shows far superior AT than BFD, because it is
 737 possible to do the query processing via depth-first search, without waiting for the lowest
 738 level for no reason.

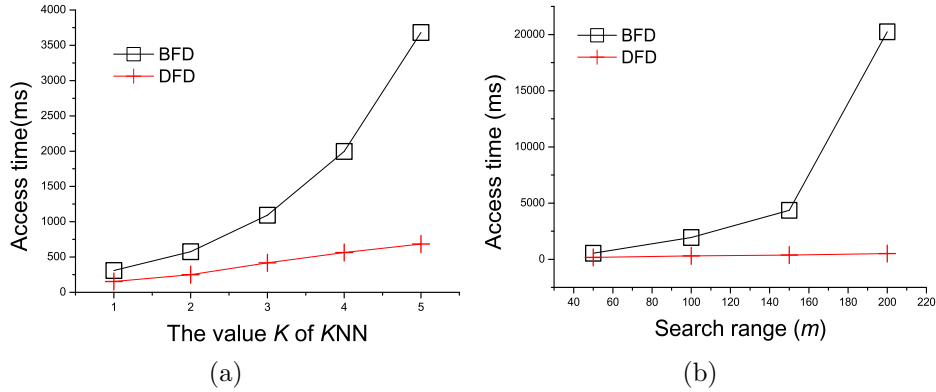


Figure 17: (a) AT according to the increase in KNN. (b) AT in response to an increase in the navigation area of the range query.

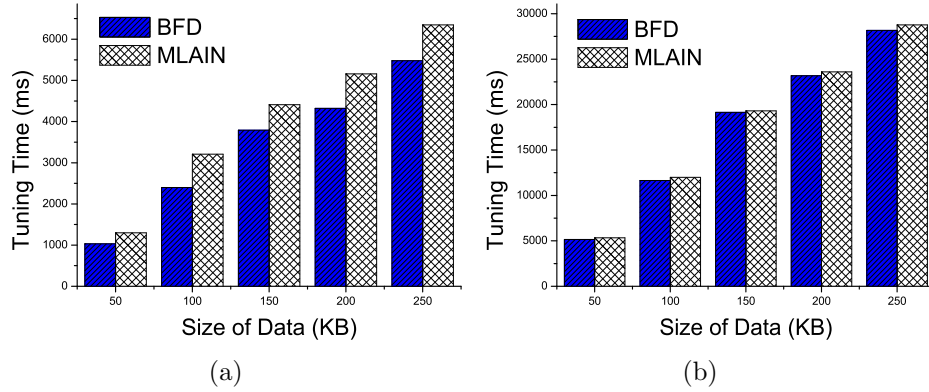


Figure 18: TT results for the case in which three index techniques are applied to the range query. (a) applying in $\mathcal{D}1$ (b) applying in $\mathcal{D}2$.

739 In the experiment, the energy efficiency was evaluated by comparing the time during
740 which clients stay in active mode. The proposed method enables clients to choose the DFD
741 algorithm to increase search speed or the BFD algorithm to improve energy efficiency.
742 In this experiment, the performance of the proposed method using BFD was compared
743 with that of the MLAIN method. The MLAIN method adopts the depth-first traversal
744 strategy to process a query. In MLAIN, clients stay in active mode while listening to each
745 and every sub-node of a target node G_i . This increases the active mode time and thus
746 decreases the energy efficiency. On the other hand, BFD can enhance energy efficiency via
747 selective listening the nodes that are not necessary for acquiring query results are pruned
748 in advance, at a higher level. Figure 18(a) and 18(b) show the energy efficiency of the
749 proposed and MLAIN methods with regard to data size changes. As can be seen in the
750 figures, the proposed method with BFD has a shorter active mode time than MLAIN,
751 gaining higher energy efficiency. Figure 19(a) and 19(b) present the energy efficiency with
752 search range changes in kNN and range queries. Here, the proposed method also achieves
753 better energy efficiency than MLAIN.

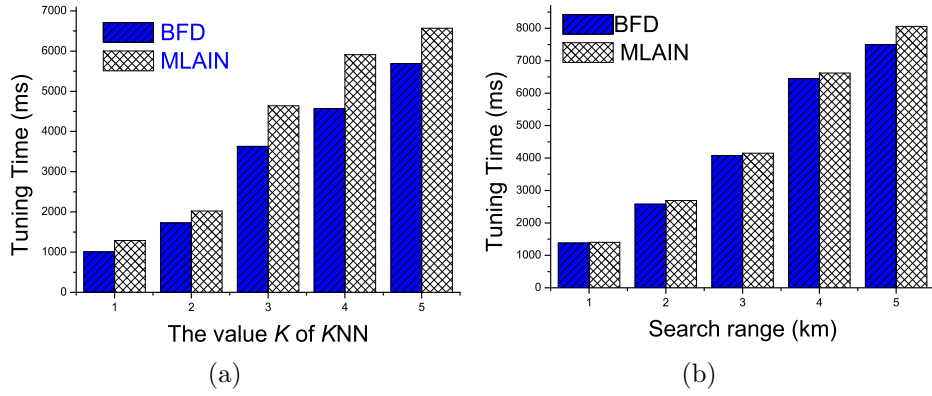


Figure 19: TT results for the case in which three index techniques are applied to the range query. (a) applying in $\mathcal{D}1$ (b) applying in $\mathcal{D}2$.

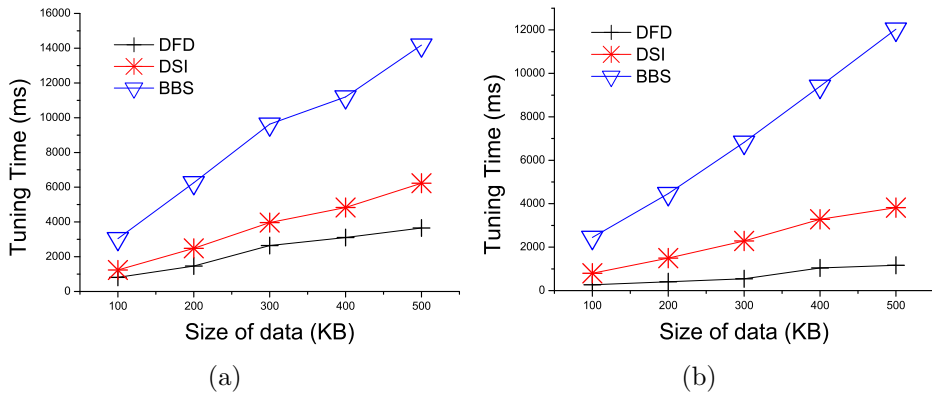


Figure 20: TT for DFD, DSI, and BBS to KNN queries. (a) applying in $\mathcal{D}1$ (b) applying in $\mathcal{D}2$.

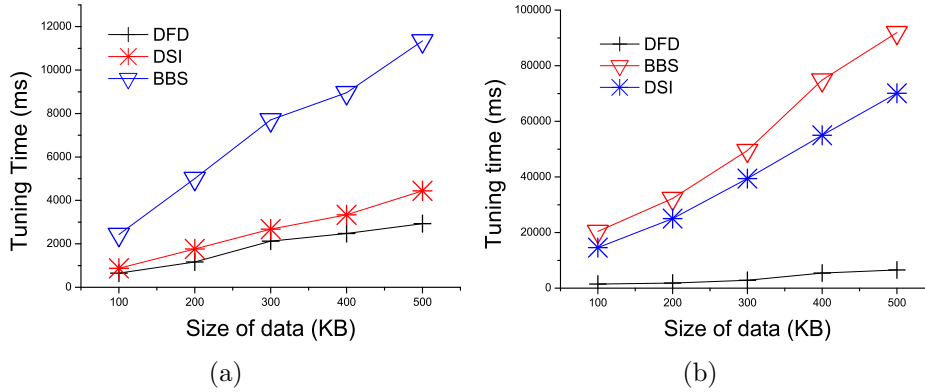


Figure 21: TT results for the case in which three index techniques are applied to the range query. (a) applying in $\mathcal{D}1$ (b) applying in $\mathcal{D}2$.

754 We compare DFD and DSI according to the increase in data size, and evaluate the
755 performance of TT and AT with regard to the BBS techniques. BFD, which displayed
756 relatively poor AT in the previous experiment is excluded from this evaluation. First, TT
757 in $\mathcal{D}1$ is compared by applying three index techniques to *KNN* queries. As can be seen in
758 Figure 20(a), DFD shows quite excellent TT. The reason is that in DSI, a client repeatedly
759 wakes up and falls asleep in the course of the search process on a large number of duplicate
760 pointers, while, in DFD, a client can wait in sleep mode for a longer time because it is
761 possible to listen selectively, using only the identification number of the grid. On the other
762 hand, we find that BBS displays the relatively worst TT, because even though the index
763 size is the smallest because it does not have a pointer that indicates the data location, it
764 does not have a pointer that supports selective tuning, so the time spent in active mode
765 is longer. Figure 20(b) shows the TT results for the three indexes were applied to a *KNN*
766 query in $\mathcal{D}2$. We find that DFD shows the relatively best TT for the same reason as in $\mathcal{D}1$.
767 On the other hand, we can also see that BBS, which must remain awake for the longest
768 time, displays the poorest TT.

769 Figures 21(a) and 21(b) show the TT results for the case in which three index techniques
770 are applied to the range query in $\mathcal{D}1$ and $\mathcal{D}2$. We find that DFD, also for similar reasons
771 as in the previous experiments, displays quite excellent TT in the range query.

772 Next, we compare AT by applying three indexes to a *KNN* query in $\mathcal{D}1$ and $\mathcal{D}2$. As
773 can be seen in Figures 22(a) and 22(b), unlike in the previous experimental results, DSI
774 displays the poorest AT, and BBS displays relatively good AT. DSI performs the worst
775 during the time it takes to obtain the final query, due to the burdens of the pointer size and
776 the search cost of a Hilbert-curve. On the other hand, DFD shows excellent AT because its
777 pointer size is relatively small, and it can quickly determine the region not to be listened
778 to, due to the hierarchical tree structure that considers the location. BBS also displays
779 good results compared to the previous TT test, because the burden of the index pointer is
780 on the low end, and it can obtain the final query result without having to listen to all the
781 data. BBS that minimizes the size of the index information reduces the broadcast cycle,
782 which allows for a faster access time in spatial query processing. However, BBS doesn't

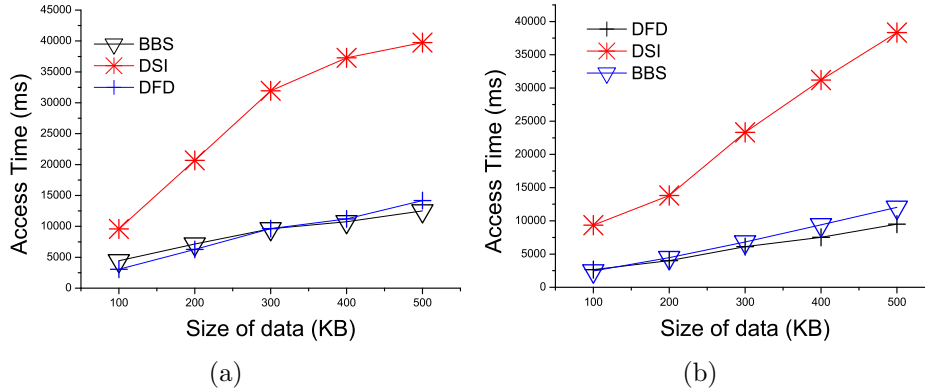


Figure 22: AT by applying three indexes to a KNN query. (a) applying in $\mathcal{D}1$ (b) applying in $\mathcal{D}2$.

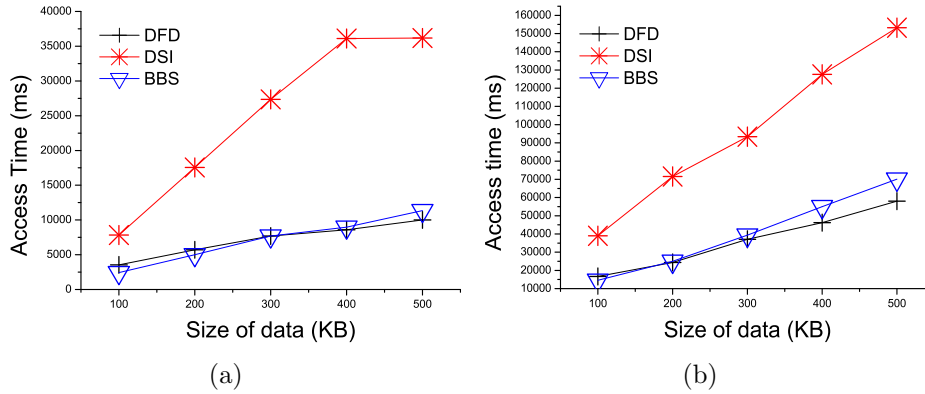


Figure 23: AT for three index techniques applying to a range query.(a) applying in $\mathcal{D}1$ (b) applying in $\mathcal{D}2$.

783 perform well because of the limitation of selective tuning ability. In general, the client's
 784 access time decreases and tuning time increases as the index information decreases.

785 Figures 23(a) and 23(b) show the results of a comparison between AT in $\mathcal{D}1$ and $\mathcal{D}2$, in
 786 which three index techniques are applied to a range query. In this case as well, DSI displays
 787 the poorest AT, due to the burden of pointer size and the inefficiency of the search method,
 788 because the increase in data size increases the number of data that must be listened to.

789 Next, we perform a test comparing TT according to an increase in KNN (see Figures
 790 24(a) and 24(b)). As can be seen in Figures 24(a) and 24(b), DFD shows excellent results,
 791 because an increase in the KNN value incurs an increased cost for searching the index
 792 data and user data. Figures 25(a) and 25(b) both show the results in $\mathcal{D}1$ and $\mathcal{D}2$ for a
 793 comparison of AT, in which the value of KNN is increased from 1 to 5. As shown in the
 794 previous experiment's results, BBS minimizes the index size and improves the access time.
 795 DFD shows excellent results, because an increase in the KNN value incurs an increased
 796 cost for searching the index data and user data. Also, there is the least amount of data to
 797 be listened to in the active mode, because of the relatively low burden on the index pointer,
 798 and the least burden of the mapping calculation of the user data from the index. In BBS,
 799 the distribution of objects in a dataset has a direct influence on the data delivery order

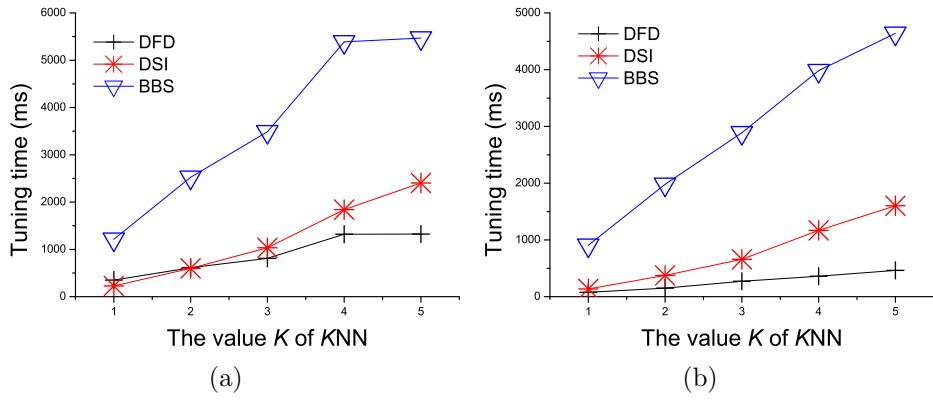


Figure 24: A comparison according to an increase in KNN. (a) applying in \mathcal{D}_1 (b) applying in \mathcal{D}_2 .

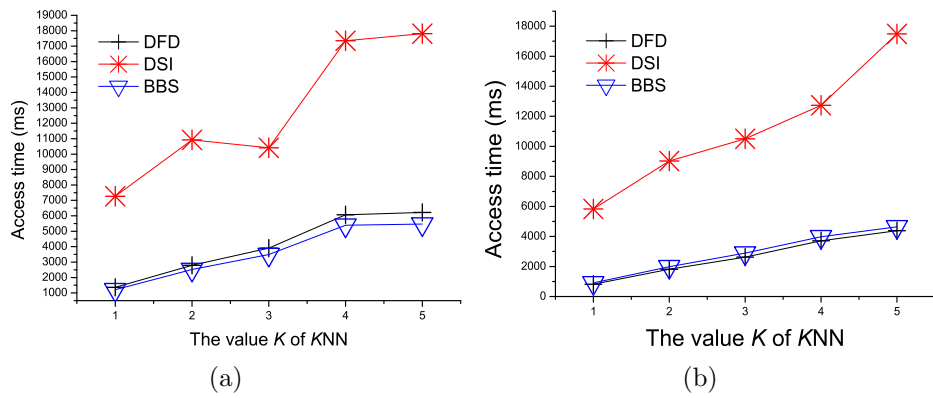


Figure 25: A comparison of AT, in which the value of KNN is increased from 1 to 5. (a) applying in \mathcal{D}_1 (b) applying in \mathcal{D}_2 .

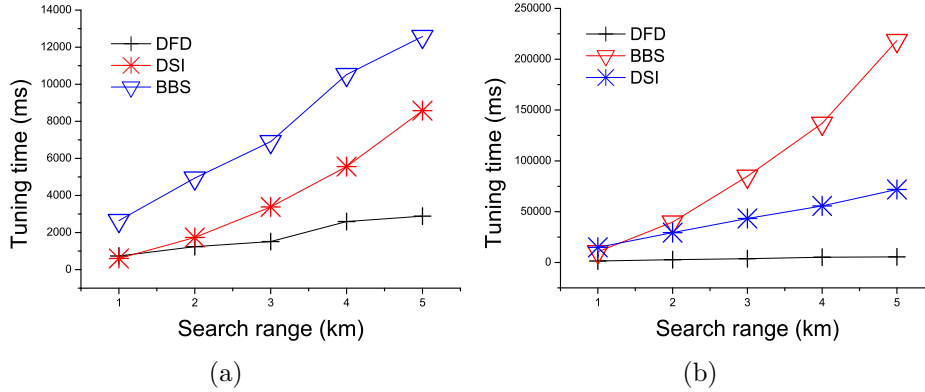


Figure 26: A comparison for TT in which the search range is increased from 1 km to 5 km. (a) applying in $\mathcal{D}1$ (b) applying in $\mathcal{D}2$.

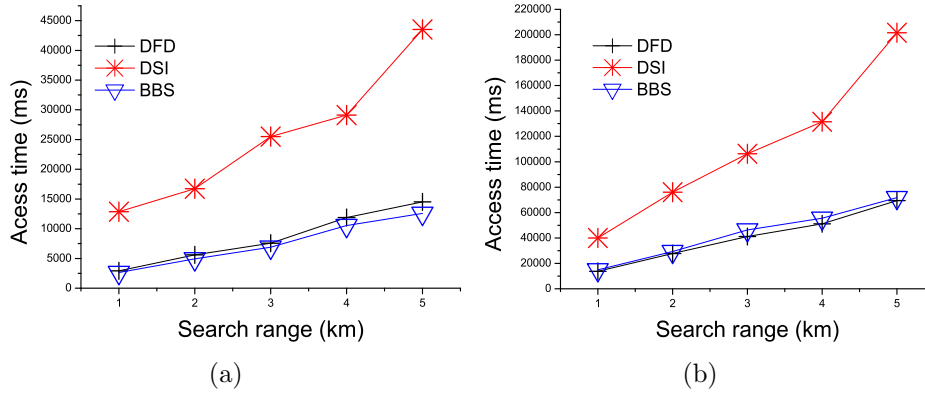


Figure 27: A comparison according to increases in the search range in the range queries. (a) applying in $\mathcal{D}1$ (b) applying in $\mathcal{D}2$.

800 and search efficiency the objects at the leftmost are delivered first, followed by delivering
 801 those at the right-hand side. In the experiment results shown in Figure 25(a), BBS has
 802 faster access time than DFD. On the other hand, DFD has a slightly better access time
 803 than BBS in Figure 25(b) because of the difference in object distributions.

804 Figures 26(a) and 26(b) both show the results of a comparison for TT in $\mathcal{D}1$ and $\mathcal{D}2$
 805 in which the search range is increased from 1 km to 5 km. We can see that DFD shows
 806 that, as the search range increases, the highest level of performance is also in TT, for the
 807 same reason as in the previous test.

808 Next, we perform a test comparing AT according to increases in the search range in the
 809 range queries. Figures 27(a) and 27(b) both show the results of a comparison of AT in $\mathcal{D}1$
 810 and $\mathcal{D}2$, in which the search range is increased from 1 km to 5 km. The results show that
 811 DSI had the worst result when the search range is increased, for reasons similar to those
 812 in the KNN query processing.

813 We now compare AT in $\mathcal{D}1$ in terms of an increase in the size of objects. An increase in
 814 the number of objects leads to an increase in the size of the pointer, which represents the

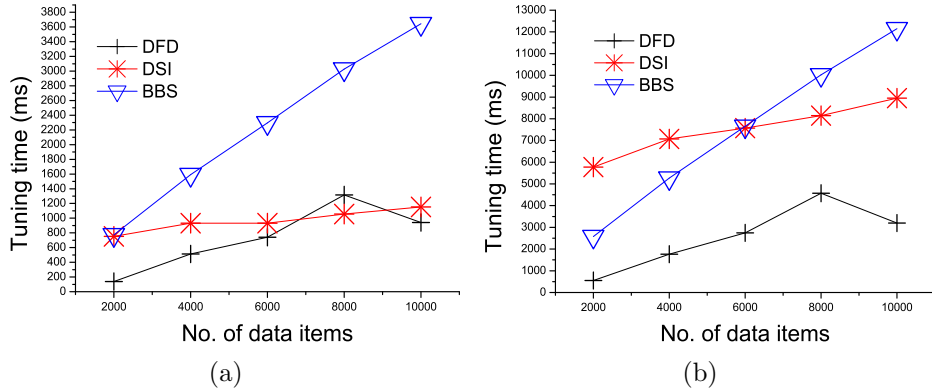


Figure 28: A comparison and measurement of TT in terms of KNN query. Both (a) and (b) applying in $\mathcal{D}1$.

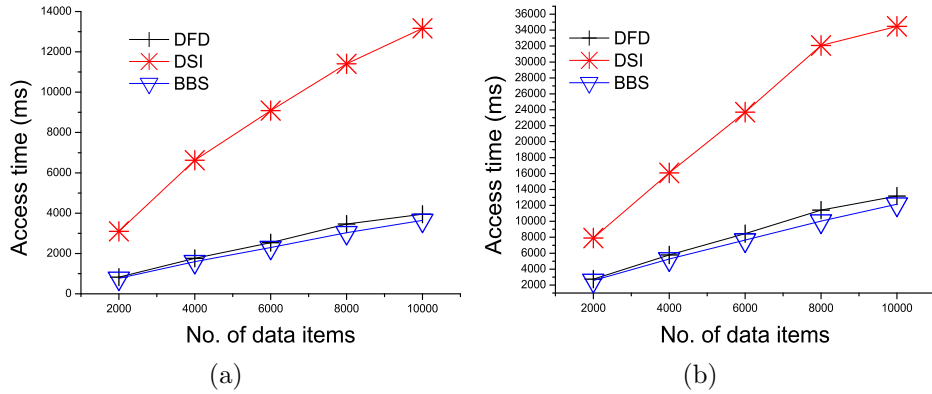


Figure 29: A comparison and measurement of AT in terms of KNN query. Both (a) and (b) applying in $\mathcal{D}1$.

815 object location and delivery time. In the case of DSI, even though a pointer that increases
 816 by index e (exponential) is used, the amount of information to be listened to becomes
 817 quite large, because the duplicate pointer increases as much as $\log_e N$ in each index frame
 818 every time the number N of data increases. On the other hand, the listening data and
 819 computation time in DZI become relatively small compared to those in DSI, due to the
 820 reduction in the cost of duplicate indexing pointers.

821 Figures 28(a) and 28(b) both show the results for TT as the number of objects increases
 822 from 2,000 to 10,000. As can be seen in the test results, DSI displays the worst TT than
 823 DFD in the majority of cases, due to the burden of having to read more pointers with
 824 the increase in the search range, as well as the problem of having to listen to unnecessary
 825 blocks due to the nature of the Hilbert-curve.

826 Figures 29(a) and 29(b) show the results of a comparison and measurement of AT in
 827 $\mathcal{D}1$ and $\mathcal{D}2$ in terms of KNN query, when the number of data is increased from 2,000
 828 to 10,000. In this case, we can see that DSI shows the worst AT, because it receives a
 829 duplicate pointer and forms an inefficient search structure as the number of data increases.
 830 On the other hand, we find that DFD, which uses DZI, displays excellent AT, as a result
 831 of the lightweight index and fast search results.

832 As can be seen in all the experimental results, AT and TT, which are the main elements
833 of performance evaluation, are both affected greatly by the index structure and search
834 method. We find that the search method that uses DZI with DFD shows the best AT and
835 TT in *KNN* and a range query.

836 8. Conclusion

837 In this paper, we proposed a distribution-based Z-order air index and query processing
838 algorithms, such as BFD (Breadth-First-Delivery), and DFD (Depth-First-Delivery) that
839 reduce AT and TT in a wireless data broadcast environment. In DZI, we are able to reduce
840 the pointer size and search cost, which account for the largest cost in the index structure,
841 by assigning an identification number only to the space of those objects that are actually
842 present. A lightweight index structure reduces the broadcast cycle and index search time,
843 and a hierarchical index structure reduces energy consumption via the client's selective
844 tuning. BFD can improve the efficiency of TT through selective tuning by previewing and
845 pruning the part not to be listened to from the upper node by delivering the DZI tree
846 information from the upper levels to the lower levels. DFD delivers the DZI in depth-first
847 order.

848 We can improve the performance of AT by determining the final query result by listening
849 to the partial data information, without having to listen to the end node of the tree via
850 the consecutive delivery of index data and user data that belong to the same grid.

851 Next, we proposed an algorithm that allows the client to self-process a spatial query
852 of a mobile object using the object's location, speed, and direction information without
853 reliance on the server for a period of time. It has proved that DZI with BFD and DFD
854 provides fast response time and minimal energy consumption. In the future, we plan to
855 study the multi-channel broadcast for location-based services.

856

857 References

- 858 [1] Bugra G., Aameek S., and Ling L., "Energy efficient exact kNN search in wireless
859 broadcast environments", In *Proc. of ACM GIS*, pp. 137-146, 2004.
- 860 [2] Kyriakos M., Spiridon B., and Dimitris P., "Continuous Monitoring of Spatial Queries
861 in Wireless Broadcast Environments", In *IEEE Trans. Mob. Comput.*, 8(10), pp. 1297-
862 1311, 2009.
- 863 [3] J. Xu, B. Zheng, W.-C. Lee, and D. L. Lee, "Energy Efficient Index for Querying
864 Location-Dependent Data in Mobile Broadcast Environments", In *Proc. of the 19th
865 IEEE Int. Conf. on Data Engineering (ICDE '03)*, pp. 239-250, 2003.
- 866 [4] Datta, A., Celik, A., Kim, J.K., VanderMeer, D., and Kumar, V., "Adaptive broadcast
867 protocols to support power conservation retrieval by mobile users", In *Proc. of IEEE
868 International Conference Data Engineering (ICDE)*, pp. 124-133, 1997.

- 869 [5] Datta, A., VanderMeer, D.E., Celik, A., and Kumar, V., “Broadcast protocols to
870 support efficient retrieval from databases by mobile users”, *ACM Trans. Database*
871 *Syst.*, 24(1), pp. 1-79, 1999.
- 872 [6] Imielinski, T., Viswanathan, S., and Badrinath, B.R., “Energy efficiency indexing
873 on air”, In *Proc. of the ACM SIGMOD International Conference on Management of*
874 *Data*, pp. 25-36, 1994.
- 875 [7] Imielinski, T., Viswanathan, S., and Badrinath, B.R., “Data on air-organization and
876 access”, *IEEE Trans. Knowl. Data Eng.*, 9(3), pp. 353-372, 1997.
- 877 [8] Shanmugasundaram, J., Nithrakashyap, A., Sivasankaran, R.M., and Ramamritham,
878 K. “Efficient concurrency control for broadcast environments”, In *Proc. of ACM SIG-*
879 *MOD International Conference on Management of Data*, pp. 85-96, 1999.
- 880 [9] Zheng, B. and Lee, D.L., “Information dissemination via wireless broadcast”, *Com-*
881 *munic. ACM*, 48(5), pp. 105-110, 2005.
- 882 [10] Park, K. and Choo, H., “Energy-efficient data dissemination schemes for nearest neigh-
883 bor query processing”, *IEEE Trans. Comput.*, 56(6), pp. 754-768, 2007.
- 884 [11] Park, K. and Valduriez, P., “Energy Efficient Data Access in Mobile P2P Networks”,
885 *IEEE Trans. Knowl. Data Eng.*, 23(11), pp. 1619-1634, 2011.
- 886 [12] Kyriakos Mouratidis and Man Lung Yiu, “Shortest Path Computation with No Infor-
887 mation Leakage”, *PVLDB*, 5(8), pp. 692-703, 2012.
- 888 [13] Baihua Z., Wang L., Ken L., Dik L., and Min S., “A distributed spatial index for
889 error-prone wireless data broadcast”, *VLDB J.*, 18(4), pp. 959-986, 2009.
- 890 [14] Swarup Acharya, Rafael Alonso, Michael J. Franklin, and Stanley B. Zdonik, “Broad-
891 cast Disks: Data Management for Asymmetric Communications Environments”, In
892 *Proc. of SIGMOD Conference*, pp. 199-210, 1995.
- 893 [15] B. Zheng, W.-C. Lee, and D. L. Lee. “Spatial queries in wireless broadcast systems”,
894 *Wireless Networks*, 10(6), pp. 723-736, 2004.
- 895 [16] C. Gotsman and M. Lindenbaum, “On the metric properties of discrete space-filling
896 curves”, *IEEE Transactions on Image Processing*, 5(5), pp. 794-797, 1996.
- 897 [17] W.-C. Lee and B. Zheng, “DSI: A Fully Distributed Spatial Index for Location-Based
898 Wireless Broadcast Services”, In *Proc. of Int’l Conf. Distributed Computing Systems*
899 *(ICDCS)*, pp. 349-358, 2005.
- 900 [18] Guttman, A., “R-trees: a dynamic index structure for spatial searching”, In *Proc. of*
901 *Special Interest Group on Management of Data (SIGMOD)*, pp. 47-57, 1984.

- 902 [19] Xuan, P., Sen, S., Gonzalez, O., Fernandez, J., and Ramamritham, K., “Broadcast on
903 demand: efficient and timely dissemination of data in mobile environments”, In *Proc.*
904 *of IEEE Real Time Technology and Applications Symposium*, pp. 38-48, 1997.
- 905 [20] Chuan-Ming Liu and Kun-Feng Lin, “Disseminating dependent data in wireless broad-
906 cast environments”, *Distributed and Parallel Databases*, 22(1), pp. 1-25, 2007.
- 907 [21] P.Nicopolitidis, G.I.Papadimitriou and A.S.Pomportsis, “Exploiting Locality of De-
908 mand to Improve the Performance of Wireless Data Broadcasting”, *IEEE Transactions*
909 *on Vehicular Technology*, vol.55, no.4, pp.1347-1361, 2006.
- 910 [22] Zaixin Lu, Yan Shi, Weili Wu, and Bin Fu, “Efficient data retrieval scheduling for
911 multi-channel wireless data broadcast”, In *Proc. of INFOCOM*, pp. 891-899, 2012.
- 912 [23] S.E. Hambruch, C.-M. Liu, W.G. Aref, and S. Prabhakar, “Query Processing in
913 Broadcasted Spatial Index Trees”, In *Proc. of Int’l Symp. Advances in Spatial and*
914 *Temporal Databases (SSTD)*, pp. 502-521, 2001.
- 915 [24] Glenn S. Iwerks, Hanan Samet, and Kenneth P. Smith, “Continuous K-Nearest Neighbor
916 Queries for Continuously Moving Points with Updates”, In *Proc. of VLDB*, pp.
917 512-523, 2003.
- 918 [25] G. Ghinita, P. Kalnis and S. Skiadopoulos, “PRIVE: Anonymous Location- Based
919 Queries in Distributed Mobile Systems”, In *Proc. of International World Wide Web*
920 *Conference (WWW)*, pp. 371-380, 2007.
- 921 [26] M. F. Mokbel, C.-Y. Chow, and W. G. Aref, “The New Casper: A Privacy-Aware
922 Location-Based Database Server”, In *Proc. of International Conference on Data En-*
923 *gineering Conference (ICDE)*, pp. 1499-1500, 2007.
- 924 [27] B. Gedik and L. Liu, “A Customizable k-Anonymity Model for Protecting Location
925 Privacy”, In *Proc. of International Conference on Distributed Computing Systems*
926 *(ICDCS)*, pp. 620-629, 2005.
- 927 [28] T. Camp, J. Boleng, and V. Davies, “A Survey of Mobility Models for Ad Hoc Network
928 Research”, *Wireless Comm. and Mobile Computing*, vol. 2, no. 5, pp. 483-502, 2002.
- 929 [29] Park K., Song M., and Hwang C., “Broadcasting and Prefetching Schemes for Loca-
930 tion Dependent Information Services”, In *Proc. of Int’l Workshop Web and Wireless*
931 *Geographical Information Systems (W2GIS)*, pp. 26-37, 2004.
- 932 [30] Kyriakos M., “Spatial queries in wireless broadcast environments”, In *Proc. ACM In-*
933 *ternational Workshop on Data Engineering for Wireless and Mobile Access (MobiDE)*,
934 pp. 39-44, 2012.

- 935 [31] Y. Xiong, Y. Deng, W. Wang, and J. Ma, “Phoenix: A Collaborative Location-Based
936 Notification System for Mobile Networks”, *Mathematical Problems in Engineering*,
937 Article ID 307498, pp. 12, 2014.
- 938 [32] Y. Wang, C. Xu, Y. Gu, M. Chen, and G. Yu, “Spatial query processing in road
939 networks for wireless data broadcast”, *Wireless Networks (WINET)*, 19(4), pp. 477-
940 494, 2013.
- 941 [33] Y. Li, J. Li, L. Shu, Q. Li, G. Li, and F. Yang, “Searching continuous nearest neighbors
942 in road networks on the air”, *Information Systems (IS)*, 42, pp. 177-194, 2014.
- 943 [34] W. Sun, C. Chen, B. Zheng, C. Chen, and P. Liu, “An Air Index for Spatial Query
944 Processing in Road Networks”, *IEEE Trans. Knowl. Data Eng (TKDE)*, 27(2), pp.
945 382-395, 2015.
- 946 [35] K. Park and P. Valduriez, “A Hierarchical Grid Index (HGI), spatial queries in wireless
947 data broadcasting”, *Distributed and Parallel Databases (DAPD)*, 31(3), pp. 413-446,
948 2013.
- 949 [36] P. Nagarkar, K. S. Candan, and A. Bhat, “Compressed Spatial Hierarchical Bitmap
950 (cSHB) Indexes for Efficiently Processing Spatial Range Query Workloads”, In *Pro-
951 ceedings of the VLDB Endowment (PVLDB)*, 8(12), pp. 1382-1393, 2015.
- 952 [37] Y. Li, L. Shu, R. Zhu, and L. Li, “A novel distributed air index for efficient spatial
953 query processing in road sensor networks on the air”, *Int. J. Communication Systems*
954 30(5), pp. 1-23, 2017.
- 955 [38] J. Shen and M. Jian, “Spatial query processing for skewed access patterns in non-
956 uniform wireless data broadcast environments”, *International Journal of Ad Hoc and*
957 *Ubiquitous Computing*, 25(1/2), pp. 4-16, 2017.
- 958 [39] D. Song and K. Park, “A partial index for distributed broadcasting in wireless mobile
959 networks”, *Inf. Sci.* 348, pp. 142-152, 2016.
- 960 [40] Y. Li, G. Li, J. Li, and K. Yao, “SKQAI: A novel air index for spatial keyword query
961 processing in road networks. *Inf. Sci.*, 430, pp. 17-38, 2018.
- 962 [41] M. Veerasha and M. Sugumaran, “Continuous k-Nearest Neighbor Queries in Road
963 Networks”, In *Proc. of International Conference on Inventive Systems and Control*,
964 pp. 1-4, 2017.
- 965 [42] S. Im, and J. Choi, “MLAIN: multi-leveled air indexing scheme in non-flat wireless
966 data broadcast for efficient window query processing”, *Computers and Mathematics*
967 *with Applications*, 64(5), pp. 1242-1251, 2012.