PAPER Special Section on Info-Plosion

Energy-Efficient Distributed Spatial Join Processing in Wireless Sensor Networks

Min Soo KIM[†a)], **Jin Hyun SON**[††], ***Members***, **Ju Wan KIM**[†], ***and*** **Myoung Ho KIM**[†††], ***Nonmembers***

SUMMARY In the area of wireless sensor networks, the efficient spatial query processing based on the locations of sensor nodes is required. Especially, spatial queries on two sensor networks need a distributed spatial join processing among the sensor networks. Because the distributed spatial join processing causes lots of wireless transmissions in accessing sensor nodes of two sensor networks, our goal of this paper is to reduce the wireless transmissions for the energy efficiency of sensor nodes. In this paper, we propose an energy-efficient distributed spatial join algorithm on two heterogeneous sensor networks, which performs in-network spatial join processing. To optimize the in-network processing, we also propose a Grid-based Rectangle tree (GR-tree) and a grid-based approximation function. The GR-tree reduces the wireless transmissions by supporting a distributed spatial search for sensor nodes. The grid-based approximation function reduces the wireless transmissions by reducing the volume of spatial query objects which should be pushed down to sensor nodes. Finally, we compare naïve and existing approaches through extensive experiments and clarify our approach's distinguished features.

key words: *distributed spatial index, distributed spatial join, in-network query processing, wireless sensor network*

1. Introduction

A sensor network is an ad hoc network composed of tiny sensor nodes having restricted battery, communication, and computation capabilities. These restrictions have made a sensor network system managed differently, compared with traditional mobile or distributed systems. A database community has viewed a sensor network as a virtual database system and has proposed various query processors such as Cougar [1], TinyDB [2], SINA [3], DSWare [4] and COSMOS [20]. The community also has proposed much work [2], [5]–[11] for energy-efficient query processing.

Spatial search and spatial join methods have been proposed in works of a distributed Spatial IndeX (SPIX) [12] and energy-efficient spatial join processing [15] in sensor networks. However, spatial queries for two or more different sensor networks, which we call heterogeneous sensor networks, have not been much discussed. Heterogeneous sensor networks generally come together with different kinds of sensor nodes and different communication protocols. A typical spatial query example for two heterogeneous sensor networks is as follows:

(Q1) Consider two heterogeneous sensor networks, where one network consists of sensor nodes having physical sensors for humidity and thermometer monitoring and the other consists of sensor nodes having chemical sensors for CO, CO_2, and SO_2 monitoring. We assume two sensor networks are laid down in the same region. However, sensor nodes of two sensor networks were produced by different companies, were installed in different time and have been managed by different organizations. Therefore, the two heterogeneous sensor networks are unable to communicate with each other in wireless networking. Here, we want to collect a set of 6-tuples of the form $<h, l_1, id_1, c, l_2, id_2>$ (which indicates humidity h of sensor node id_1 in location l_1 and CO density c of node id_2 in location l_2) that satisfies the following selection predicates and a spatial join predicate: $h \geq 60\%$, $c \geq 8$ ppm and distance $(n_1, n_2) \leq 30$ m. Here, distance (n_1, n_2) denotes a physical distance between nodes n_1 and n_2. Note that node n_1 and node n_2 are in different sensor networks. Informally speaking, this query finds a set of small areas together with their humidity and CO density where both humidity and CO density exceed certain threshold values.

To answer the query Q1, distributed spatial join processing should be performed to materialize a proximity relationship between sensor nodes n_1 and n_2.

In conventional spatial databases, the distributed spatial join query has been extensively studied for join processing of the large volume and high complexity of spatial data. However, conventional spatial join algorithms [13], [14], [21], [22] cannot be straightforwardly applied to distributed spatial join processing on sensor networks. Most of the conventional algorithms are based on centralized spatial indexes such as the R-tree and its variants (R^+-tree, R*-tree). However, the centralized indexes cannot be built on distributed sensor nodes. Moreover, the performance of the conventional algorithms are compared on the total time (composed of CPU time, IO time, and transmission time of spatial query objects), while the performance in sensor networks are compared on the number of wireless transmissions among sensor nodes. In other words, the distributed spatial join algorithm in sensor networks focuses on raising

Manuscript received August 4, 2009.
Manuscript revised January 4, 2010.

[†]The authors are with the Dept. of Telematics Research, ETRI, 138, Gajeongno, Yuseong-gu, Daejeon, 305–700, Republic of Korea.

[††]The author is with the Dept. of Computer Science and Engineering, Hanyang University, 1271 Sa-3dong, Sangnok-gu, Ansan, Kyeonggi-do 426–791, Republic of Korea.

[†††]The author is with the Dept. of Computer Science, KAIST, 335 Gwahangno, Yuseong-gu, Daejeon, 305–701, Republic of Korea.

a) E-mail: minsoo@etri.re.kr

DOI: 10.1587/transinf.E93.D.1447

energy efficiency of sensor nodes by reducing the wireless transmissions.

In this paper, we propose an energy-efficient distributed spatial join algorithm on two sensor networks. We also propose two optimization schemes for performance improvement of the algorithm. They are a Grid-based Rectangle tree (GR-tree) and a grid-based approximation function which can reduce the number of wireless transmissions. We conducted extensive experiments on the proposed algorithm. The experimental results show that the proposed algorithm is effective and available in comparison with existing algorithms.

The remainder of this paper is organized as follows: In Sect. 2, we survey researches related to spatial search and spatial join methods that have been announced in conventional spatial databases and sensor networks. In Sect. 3, we propose a distributed spatial join algorithm and we also propose the GR-tree and a grid-based approximation function. In Sect. 4, we present the experimental results of our algorithm. In Sect. 5, we discuss the scalability of our algorithm and finally, we present our conclusions in Sect. 6.

2. Related Work

Lots of distributed spatial join algorithms have been proposed on conventional spatial database management systems (SDBMS). Abel et al. [14] proposed a distributed spatial join algorithm which is based on the spatial semijoin approach. Tan et al. [13] suggested a similar approach. Their experimental study shows that the semijoin-based algorithm provides useful reductions in the total processing time of a spatial join operation. However, the algorithms [13], [14] are not directly related with our problem which focuses on reducing energy consumption of sensor nodes.

Several in-network join methods in sensor networks have been presented. Abadi et al. [10] presented REED, a system for Robust and Efficient Event Detection in sensor networks. The key idea behind REED is to form a condition table for event detection, and then to distribute the table throughout the sensor network. Once the table has been disseminated, each node performs in-network joins between the table and its sensor readings. Here, if the table is too large to reside in any node's memory, REED distributes fragments of the table into a sensor group which is composed of several nodes. The in-network join is cooperatively performed within the sensor group. REED shows significant energy savings through the in-network join, but it does not work well when the table is too large or the density of a sensor network is not sufficiently high. For an extension of REED, Jeon et al. [24] proposed HIPaG, called the Hybrid In-network join with join Paths and join Groups. Here, a join group is the same as a sensor group in REED. A join path is a sequence of nodes in a routing tree such that every two nodes of the sequence have a parent-child relationship and the union of fragments stored at all the nodes consists of the condition table. HIPaG also performs the in-network join between sensor readings and fragments of the condition table in such a way that it utilizes join groups in high density region and join paths in low density region of a sensor network. Gil et al. [23] proposed Scoop, a system for adaptive indexing and querying stored data in sensor networks. In Scoop, a base station dynamically builds a storage index using historical statistics about a rate of queries and distribution of sensor readings. A storage index maps sensor reading to node ID. After generating a storage index, a base station disseminates the index to all nodes and each node stores specific sensor readings according to the index. Using this index, queries can be answered from specific nodes, without flooding the queries throughout the network. Zhu et al. [16] proposed an algorithm for join processing of multiple sensor readings in a sensor network. They insist that their perpendicular approach should be load-balanced, communication-efficient, and should incur near-optimal transmission cost for the special case of binary joins in grid networks. Although these algorithms [10], [16], [23], [24] show good performance through the in-network join, they cannot be directly applied to our problem in the following reasons. First, the algorithms do not present a solution for spatial join processing. In other words, the algorithms never consider important features of SDBMS such as spatial filtering and distributed spatial indexing. Second, the algorithms do not support a geographical routing, which influences on the performance of spatial filtering. They assume that a certain routing protocol exists.

Recently, there have been several works for spatial search or join processing in sensor networks. Soheili et al. [12] proposed SPIX over a sensor network by constructing the spatial index of the R-tree [18] and its variant [19]. They also presented a distributed way of optimizing the SPIX to reduce energy consumption of sensor nodes during spatial search operation. The SPIX forms a geographical routing tree with distribution on sensor nodes. Their experimental study shows that the SPIX is efficient and scalable in processing spatial search. However, it is likely to incur much overlap among MBRs of intermediate sensor nodes when applying construction strategies of the R-tree to a sensor network as it is. Meka et al. [25] proposed a distributed spatio-temporal index structure for sensor networks called DIST to trace a moving object. In DIST, a sensor network is hierarchically decomposed into levels and there is a quad-tree like partitioning at each level. Each partitioning (called cell) has a leader and the leader is connected to its four quadrant leaders. Index updates and range query executions are executed by the leader. The leader dynamically updates time interval information about a moving object and propagates the information to its parent. DIST is efficient in a spatio-temporal range query for a moving object. However, it cannot be directly applied to our problem in several reasons. First, DIST's assumption that each leader is connected to its quadrant leaders is not realistic for a large-scale sensor network. Second, DIST uses the existing greedy perimeter stateless routing (GPSR) algorithm, without presenting a geographical routing tree. Third, temporal index is dynamically optimized for a moving object,

but spatial index is fixed to the quad-tree. Yiu et al. [15] proposed an energy-efficient approach for in-network computation of a spatial join predicate in a single sensor network. This approach shows good performance for low-selectivity join queries having a short distance constraint less than a transmission distance of a sensor node. However, it is not directly related with our problem which handles distributed join among heterogeneous networks.

There have been many works on efficient in-network join or spatial query processing in a single sensor network. However the spatial join query between heterogeneous networks, which considers both distributed spatial index and a geographical routing, has not been much discussed.

3. Distributed Spatial Join Processing

A straightforward approach to perform a distributed spatial join between two sensor networks is to process selection predicates and a spatial join predicate at a server after acquiring all sensor readings. This approach called centralized approach, though simple, incurs high transmission cost in accessing all sensor nodes.

A simple optimization scheme for the centralized approach is to process a selection predicate in a network. This scheme pushes down a selection predicate to a sensor network like ACQP (ACquisitional Query Processor) [17] and the filtered sensor readings which satisfy the predicate are acquired to a server. Then, the server performs a spatial join predicate. Although this may reduce transmission cost spent in acquiring sensor readings through in-network processing of the selection predicate, it still spends high transmission cost in pushing down the predicate to nodes.

The second optimization scheme is to process a spatial join predicate as well as selection predicates in sensor networks. This scheme pushes down a spatial join predicate as well as selection predicates and the filtered sensor readings from spatially filtered nodes are acquired. In this scheme, we can find two research challenges.

- The first challenge is to find out an efficient in-network spatial search method which can filter out sensor nodes which never participate in the spatial join processing.
- In order for an in-network processing of the spatial join predicate, the spatial objects of one sensor network should be sent to the other network. However, this incurs high transmission cost. The second challenge is to find out an efficient method which can reduce the volume of the spatial objects.

In this section, we propose a distributed spatial join algorithm based on the second optimization scheme. We also propose the GR-tree and a grid-based approximation function for the two research challenges.

3.1 Distributed Spatial Join Algorithm

A distributed spatial join query between two sensor networks is defined as follows:

Definition 1: Distributed Spatial Join Query (*DQ*). Let SN_1 and SN_2 be two sensor networks. There is one tuple for a sensor node n_i ($n_i \in SN_i$), where the tuple schema consists of an attribute for a sensor location and one or more attributes for senor readings (i.e., temperature, humidity, CO_2, CO, etc.). We will use n_i to denote a tuple for a sensor node of a sensor network SN_i if there is no ambiguity. A distributed spatial join query for SN_1 and SN_2 is defined by $DQ = \{<n_1, n_2> \mid n_1 \in SN_1,\ n_2 \in SN_2$ such that $f_1(n_1)$, $f_2(n_2)$ and $g(n_1, n_2)$ are true.$\}$. Here, f_1 and f_2 are selection predicates for sensor readings and g is a spatial join predicate for sensor locations. For instance, in the query Q1 described in Sect. 1, $h \geq 60\%$ is f_1, $c \geq 8$ ppm is f_2 and distance $(n_1, n_2) \leq 30$ m is g.

Our proposed distributed spatial join algorithm which uses a spatial semijoin [14] concept is as follows:

Algorithm 1: Distributed Spatial Join Algorithm Using Spatial Semijoin

Input: Two sensor networks SN_1 and SN_2 with sensor locations L_1 and L_2, respectively.

Output: A set of $<n_1, n_2>$ that satisfies selection predicates f_1 and f_2 and a spatial join predicate g.

Step 1: First, a server acquires all sensor tuples R from SN_1, which satisfy a selection predicate f_1. $R = \{n_1 \mid n_1 \in SN_1$ such that $f_1(n_1)$ is true.$\}$.

Step 2: Then R is projected on a spatial attribute L_1 at a server and a result set R' is created. $R' = \{r.L_1 \mid r \in R\}$.

Step 3: Each record $r.L_1$ is mapped into a two-dimensional object in order to apply a spatial join predicate g. $R^T = \{T(r') \mid r' \in R'\}$. T is a one-to-one mapping function such that each record of R' is mapped into a record in R^T.

Step 4: Then records of R^T are mapped into a spatially approximated object for spatial semijoin processing. $R^P = \{P(r^T) \mid r^T \in R^T\}$. P is a many-to-one spatial approximation function such that one or several records of R^T are mapped into a spatial query object of R^P.

Step 5: A server pushes down R^P and f_2 to SN_2 and then acquires sensor tuples S from SN_2, which satisfy a selection predicate f_2 and a spatial semijon predicate. $S = \{n_2 \in SN_2 \mid r^P \in R^P$ such that $g(r^P, n_2)$ and $f_2(n_2)$ are true.$\}$.

Step 6: A server performs a refinement phase of a spatial join predicate g between R and S and a final result F is created. $F = \{<r, s> \mid r \in R,\ s \in S$ such that $g(r, s)$ is true.$\}$

Figure 1 shows a processing sequence of this algorithm when processing the query Q1.

In Step 3, for example, each record l_1 in the query Q1 having a spatial join predicate of distance $(n_1, n_2) \leq 30$ m is mapped into a circle object whose center is l_1 and radius is 30 m. In Step 4, a representative spatial approximation is MBR-based approximation which maps high complexity of spatial objects into MBR (Minimum Bounding Rectangle)

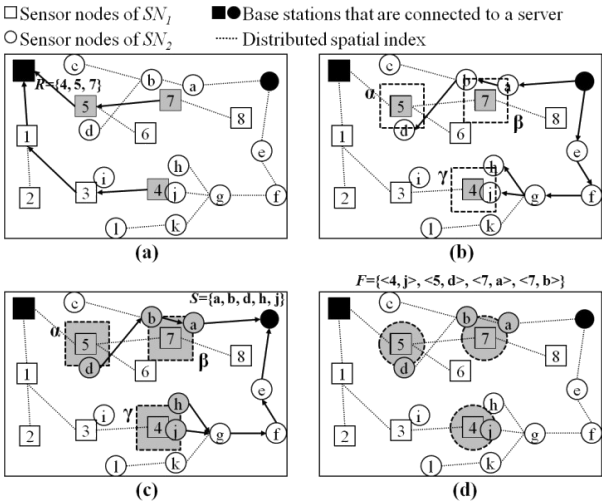

Fig. 1 Processing sequence of the distributed spatial join algorithm: (a) a server acquires $R = \{4, 5, 7\}$ from SN_1, which satisfies f_1, (b) a server makes a spatially approximated objects $R^P = \{\alpha, \beta, \gamma\}$ and then pushes down R^P and f_2 to SN_2, (c) a server acquires $S = \{a, b, d, h, j\}$ from SN_2, which satisfies f_2 and a spatial semijoin predicate, and (d) a server performs a refinement phase of a spatial join predicate g between R and S and gives a final result set $F = \{<4, j>, <5, d>, <7, a>, <7, b>\}$. Here, an element h of S is filtered out in the final result.

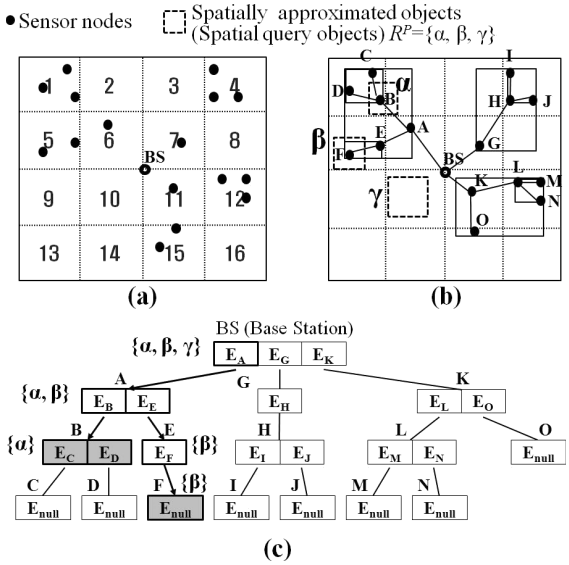

Fig. 2 A GR-tree index structure and spatial search process: (a) sensor nodes and their fixed grid addresses, (b) GR-tree index structure and spatial query objects (spatially approximated objects) $R^P = \{\alpha, \beta, \gamma\}$, and (c) spatial search process using the GR-tree.

like Fig. 1 (b). In Step 5, in-network spatial searches by the approximated objects r^P should satisfy the spatial semijoin predicate $g(r^P, n_2)$ like Fig. 1 (c). The spatial approximation in Step 4 and the spatial searches in Step 5 are important for the performance of our algorithm. In the following sections, we propose the GR-tree for Step 5 and a grid-based approximation function for Step 4.

3.2 Proposed Distributed Spatial Index

In this section, we propose the GR-tree for the optimization of the in-network spatial search in Step 5. In a sensor network, all routing paths converge into a base station. Such the convergence of the routing paths is likely to incur much overlap among MBRs of intermediate sensor nodes when a geographical routing tree such as SPIX is used. The closer the routing paths reach a base station, the more overlap happens. Therefore, the main design policy of the GR-tree is to minimize the overlap among such MBRs with maintaining the tree depth minimized at the same time. The minimization of the overlap and the tree depth may decrease the number of geographical routing to be traversed in the in-network spatial search. Unfortunately, the overlap minimization technique of the R*-tree cannot be applied to a sensor network since it never considers a geographical routing topology. Therefore, we propose a new minimization technique for the GR-tree.

3.2.1 The GR-Tree Index Structure

The GR-tree is a distributed spatial index structure built with distribution on sensor nodes. The GR-tree essentially forms a geographical routing tree in a sensor network. Using the geographical routing tree, a spatial query is selectively transmitted from a base station to sensor nodes and its result is returned. Figure 2 illustrates a GR-tree index structure and spatial search process using the GR-tree. Similar to the R-tree [18], each sensor node in the GR-tree structure maintains entries information as many as the number of child nodes below it. Each entry consists of a child's node ID, grid address, tree depth, and MBR, where the MBR bounds location of the child node and MBRs of grandchildren. The entry is used when a sensor node determines whether a query needs to be applied locally and pushed down to its children. As shown in Fig. 2, an entry E_A of a base station has a node ID A, a grid address 6, a tree depth 1, and an MBR that minimally bounds MBRs of E_B and E_E and location of the child A.

3.2.2 Spatial Searching Using the GR-Tree

The spatial search algorithm starts from a base station in a manner similar to the R-tree [18] except that it is performed with distribution among sensor nodes.

Algorithm 2: Spatial Search
Input: Spatial query object s, a GR-tree, a sensor node n
Output: Sensor nodes which are intersected with s
Step 1: Searching intermediate sensor nodes
If n is an intermediate sensor node, check each entry E to determine whether each entry's MBR intersects with s. If so, the spatial search is forwarded to child sensor nodes pointed by all intersected entries. Also, the spatial search checks whether the n's location intersects with s. If so, the n's location and ID are returned to its parent sensor

node. Here, the parent has to wait until its child nodes return their result, before it returns its result.
Step 2: Searching leaf sensor nodes
If n is a leaf sensor node in the GR-tree, check the n to determine whether the n's location intersects with s. If so, the n's location and ID are returned to its parent node.

As shown in Fig. 2 (b) and 2 (c), spatial searches with the query objects $\{\alpha, \beta, \gamma\}$ can be performed using the Algorithm 2. A base station checks that MBRs of E_A, E_G, E_K intersect with α, β, and γ. Then, α and β are forwarded to sensor node A through the GR-tree. As the same way, α and β are forwarded to sensor node B and E, respectively. Finally, the base station receives a result {B, F}. Here, we can observe that our spatial searches reduce the wireless transmissions by visiting only {A, B, E, F} instead of all sensor nodes.

3.2.3 Building the GR-Tree

In construction of the GR-tree, we assume that sensor nodes are stationary and location-aware. We also assume all sensor nodes are deployed at a fixed area which is divided into equal-sized fixed grids. The GR-tree is built in a different manner compared with the R-tree because of the following properties.

- Base station doesn't know locations of all sensor nodes. Each sensor node only knows its own location.
- GR-tree is built with considering geographical routing topology as well as spatial proximity among nodes.
- GR-tree index is stored with distribution on nodes.

The GR-tree is built using an advertisement and a parent selection phase. In the advertisement phase, each sensor node broadcasts an advertisement message to other sensor nodes in order to find its candidate parents and candidate children. The advertisement message M includes a node ID (nid), a tree depth (d), and location (l_a) of the advertiser and location (l_b) of the base station. In the parent selection phase, each sensor node selects its parent node among the candidate parents, which best fits both the spatial proximity and the geographical routing.

Advertisement phase starts from a base station. The base station broadcasts an advertisement message to sensor nodes in its transmission range. After receiving the message, the sensor nodes advertise themselves to other sensor nodes. This advertisement phase continues until all sensor nodes receive advertisement messages. In this phase, each sensor node performs the following works:

Algorithm 3: Advertisement
Input: Two sensor nodes n_1, n_2 and n_1's advertisement message $M = <nid, d, l_a, l_b>$
Output: n_1's candidate children and n_2's candidate parents
Step 1: A sensor node n_2 receives an advertisement message M from a sensor node n_1. n_2 reviews if there is an equal sensor node ID in its candidate child list for a given nid. If there exist, n_2 completes this phase.
Step 2: Otherwise, n_2 computes its grid address and tree depth. The grid address is easily computed using n_2's location, the fixed grid size and the overall area size. The tree depth is computed by adding one to d.
Step 3: Then, n_2 adds M to its candidate parent list and returns an acknowledgement including its node ID to n_1.
Step 4: n_1 adds n_2's node ID to its candidate child list.

In Step 1, n_2's review process is performed in order to avoid cross advertisements between n_1 and n_2. It prevents a cycle in the GR-tree. In Step 2, we assume that each sensor node already knows the overall area size and a fixed grid size. Here, we set a transmission distance of a node to be a fixed grid size through experimental experiences. After the Step 3 and 4 is completed, n_1 and n_2 are going to have a candidate child list and a candidate parent list, respectively.

Parent selection phase starts from leaf sensor nodes that have no candidate children. If a node has candidate children, it waits until its children determine their parent. This parent selection continues until all sensor nodes select their parent. The base station is selected as the last parent. In this phase, a sensor node performs the following works:

Algorithm 4: Parent Selection
Input: A sensor node n having a candidate parent list
Output: GR-tree index structure
Step 1: n finds sensor nodes S from its candidate parent list, whose locations are within n's grid and tree depths are both less than n's tree depth and a minimum among S. Here, n's grid is a fixed grid including n.
Step 2: After completing Step 1, if there is only one sensor node in S. n selects the sensor node as its parent. If there are two or more sensor nodes in S, n determines a sensor node as its parent, which has minimum distance to a base station. If S is empty, Step 1 and Step 2 are repeated using n's neighboring grids instead of n's grid.
Step 3: n stores the parent and removes the candidate parent list. Then, n sends its entry to the parent.
Step 4: The parent adds n's entry information to its child list and updates its MBR in order to include n's MBR.

In Step 1, we should note that the sensor node n first finds its parent in the grid including n. It enables the GR-tree to preserve spatial proximity among sensor nodes when building a geographical routing tree. In Step 2, when S is empty, the algorithm finds a parent from sensor nodes which are within n's neighboring grids. As shown in Fig. 3, a grid has at most eight neighboring grids and the search order for the eight grids makes a great influence on a GR-tree index. For the search order, we basically give higher priority to a grid which is near to a base station in order to reduce a depth of the GR-tree as shown in Fig. 3 (a). However, as shown in Fig. 3 (b) and 3 (c), we adapt a new parent selection criterion in the GR-tree. The new criterion gives higher priority to a grid which is adjacent to a straight line rather than a grid which is near to a base station. We adapt this criterion

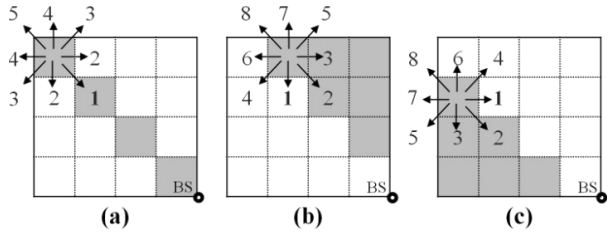

Fig. 3 Search order for the neighboring grids, where x and y is an ordinal number for x-axis and y-axis grids, respectively: (a) when a sensor node is located at a diagonal-line, i.e., $x = y$, (b) when $x > y$, and (c) when $x < y$.

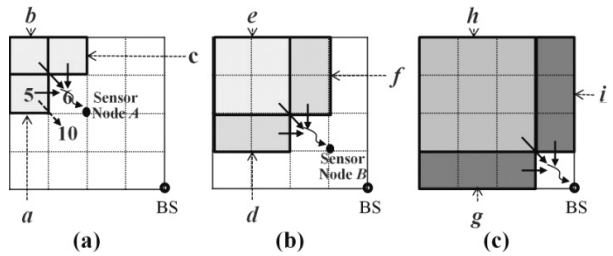

Fig. 4 Hierarchical GR-tree index structure that has no overlapped area among MBRs: (a) an entry set of A = $\{a, b, c\}$, (b) an entry set of B = $\{d, e, f\}$ and e's MBR area minimally bounds MBRs of a, b, and c, and (c) an entry set of C = $\{g, h, i\}$ and h's MBR area bounds MBRs of d, e, f.

in order to minimize the overlap among MBRs of intermediate sensor nodes in a GR-tree index structure. For example, Fig. 4 illustrates an ideal GR-tree index structure using our new criterion, which has no overlap among MBRs. Here, we assume the MBRs coincide with the borderlines of fixed grids as an ideal case of the GR-tree. In Fig. 4 (a), a node in a grid 5 originally should find its parent from the grid 10. However, our criterion finds a parent in the grid 6, according to the search order as shown in Fig. 3 (c). Then this criterion creates an MBR e in a parent level of the node, which includes MBR a, b, and c. The MBR e never overlaps with MBR d and f which are also created by the new criterion. After the parent selection is repeated to the base station, it comes to maintain children having MBR h, g, and i which have no overlap, as shown in Fig. 4 (c).

In the performance study, we will show you how the GR-tree contributes to reducing the wireless transmissions.

3.2.4 Maintaining the GR-Tree

Although we assume sensor nodes are stationary, sometimes a sensor node may fail to respond or a new sensor node may be added to a sensor network. In this case, the GR-tree should be updated through an insertion and a deletion phase. Our insertion algorithm can be easily performed using the advertisement and the parent selection algorithm as follows:

Algorithm 5: Insertion
Input: A new sensor node n and a GR-tree index structure
Output: GR-tree index structure including a sensor node n
Step 1: A sensor node n broadcasts a message to its vicinity nodes and requests for advertisement.
Step 2: Sensor nodes that hear the request sends their advertisement message M to n. n adds the messages Ms to its candidate parent list and computes its grid address and tree depth like the advertisement algorithm.
Step 3: For n's candidate parent list, the parent selection algorithm is invoked and its parent is determined.
Step 4: If the MBR of n's parent have been updated in Step 3, MBR updating is propagated from the parent to the base station until the updating doesn't happen.

However, a deletion is not simple like the insertion. First, a sensor node failure should be found. In this paper, we find a sensor node failure using a technique which is used in the SPIX [12]. In brief, we set a timeout period on sensor nodes. When the timeout period expires, the sensor nodes verify parent-child relationships. If a parent node finds a child node that doesn't respond to the verification, the parent determines that the child has failed. On the contrary, if a child finds a parent that doesn't respond to the verification, the child determines that the parent has failed. Second, if a sensor node failure happens, a deletion algorithm is performed as follows:

Algorithm 6: Deletion
Input: A sensor node n_1 and its child node n_2
Output: A GR-tree index structure excluding a failed node
Step 1: If n_2 fails to respond, n_1 updates its MBR in order to exclude n_2's MBR. Then, if n_1's MBR is changed, MBR updating is propagated to the base station.
Step 2: If n_1 fails to respond, n_2 performs the insertion algorithm. If n_2's new parent is determined from the insertion, n_2 completes the deletion algorithm.
Step 3: Otherwise, n_2 propagates an insertion message to its vicinity nodes n_{vs} and receives acknowledgements including n_{vs}'s IDs. Here, n_{vs} adds n_2 to its candidate child list and n_2 adds n_{vs}'s IDs to its candidate parent list.
Step 4: Each node of n_{vs} performs the insertion algorithm. If each node of n_{vs} finds its parent, it sends <ID, $true$> message to n_2. Otherwise, it sends <ID,$false$> message to n_2 and hierarchically propagates an insertion message to its vicinity nodes until its parent is found.
Step 5: If n_2 receives <ID,$false$>, n_2 removes a node having the ID from its candidate parent list. If n_2 receives <ID, $true$>, n_2 keeps its candidate parent list. After receiving all messages for n_2's candidate parent list, n_2 performs the parent selection algorithm.

When a child node n_2 fails, n_2's deletion can be simply performed as shown in Step 1. When a parent node n_1 fails, there is no overhead if n_2's new parent is directly determined as shown in Step 2. However, if an insertion message is hierarchically propagated to n_2's vicinity nodes in order to find a new parent, the deletion algorithm is likely to spend too much transmission cost in Step 3–5. Therefore, we should note that frequent node failures may spend much transmission cost in maintaining the GR-tree.

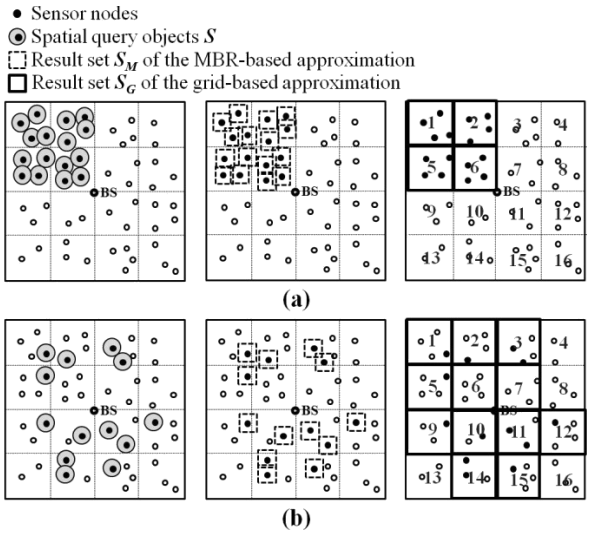

Fig. 5 Spatial approximation examples: (a) when a set S is skewed: S_M has sixteen MBRs, while S_G has grid addresses of $\{1, 2, 5, 6\}$ and (b) when S is randomly distributed: S_M has thirteen MBRs, while S_G has grid addresses of $\{1, 2, 3, 5, 6, 7, 9, 10, 11, 12, 14, 15\}$.

3.3 Proposed Spatial Approximation Function

We use a grid-based approximation function for the optimization of Step 4. The grid-based approximation function is used to map two dimensional query objects into a small volume of fixed grid addresses. Figure 5 shows an example of the grid-based approximation compared with the conventional MBR-based approximation. In Fig. 5, the left side shows raw spatial query objects S, the middle shows a result set S_M of the MBR-based approximation, and the right side shows a result set S_G of the grid-based approximation.

As shown in Fig. 5 (a), under skewed distribution of the S, we can see that the grid-based approximation is more efficient than the MBR-based approximation. When we assume that a numeric value needs 4 bytes, the former forms only 16 bytes (4 grid addresses $\times$ 4 bytes) while the latter forms 256 bytes (16 MBR $\times$ 4 coordinates $\times$ 4 bytes). However, if the S is randomly distributed as shown in Fig. 5 (b), we can see the grid-based approximation may incur high transmission cost, since the search area occupied by S_G is significantly increased. In the performance study, we will show you how the proposed approximation function can contribute to reducing the wireless transmissions.

4. Performance Study

We executed a performance study on the distributed spatial join algorithm to answer the following questions:

- Is the proposed algorithm effective in comparison with naïve and conventional algorithms?
- Is the proposed algorithm available for wireless sensor networks?

4.1 Experimental Setup

In the experiments, we performed the query Q1 having proximity relationship between two sensor networks SN_1 and SN_2. The performance was compared on the number of transmission messages. The transmission messages include messages for sending f_1 to SN_1, messages for receiving a result R from SN_1, messages for sending f_2 and R^P to SN_2, and messages for receiving a result S from SN_2. Thus, the total transmission messages are computed, as follows:

$$N_{total} = N_{1,total} + N_{2,total}$$
$$N_{1,total} = N_{1,sending} + N_{1,receiving}$$
$$N_{2,total} = N_{2,sending} + N_{2,receiving},$$

where $N_{i,sending}$ and $N_{i,receiving}$ represent the number of messages spent in sending and receiving, respectively ($i = 1, 2$). In the experiments, we measure only $N_{2,total}$ as our performance metric, since all join algorithms spend the same $N_{1,total}$ in visiting all sensor nodes of SN_1.

In order for a performance study, we compared our algorithm with the following algorithms.

- DJ-GT-M: This is the proposed join algorithm that uses the GR-tree and the MBR-based approximation.
- DJ-GT-G: This is the proposed join algorithm that uses the GR-tree and the grid-based approximation.
- DJ-CA (Centralized Approach): This is the most naive algorithm which performs a distributed spatial join query at a server. This algorithm has to visit all sensor nodes of two sensor networks. If this algorithm runs without a distributed index, its transmission cost will be increased exponentially due to the flooding of the transmission messages. So, we assume this algorithm runs on the GR-tree.
- DJ-SP-A (Area)/DJ-SP-P (Perimeter): This is an algorithm that uses the existing SPIX and the MBR-based approximation. In [12], SP-A means the SPIX that uses the least MBR area enlargement and SP-P means the SPIX that uses the least MBR perimeter enlargement as a parent selection criterion.
- DJ-CPS (Closest Parent Selection): This is an algorithm that uses the distributed spatial index that focuses on reducing an average tree depth. In this index, a node chooses a candidate as its parent, which has the minimum distance to a base station among candidates.

Figure 6 illustrates several distributed spatial indexes that are used in the above algorithms for the same sensor nodes.

In the experiments, we intentionally created various spatial query objects (results of SN_1) which are pushed down to SN_2. The spatial query objects were created by reflecting the following experimental constraints: number of sensor nodes, distribution of spatial query objects, size of spatial query objects, number of spatial query objects, packet size, and transmission distance of a sensor node. Detail descriptions about the constraints are summarized in Table 1. We also created a simulated sensor network for SN_2.

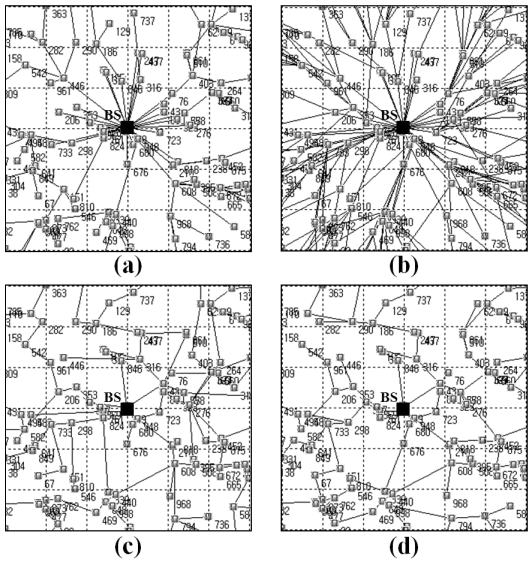

Fig. 6 Distributed spatial indexes for a sensor network: (a) the GR-tree, (b) the CPS, (c) the SPIX based on MBR area enlargement, (d) the SPIX based on MBR perimeter enlargement.

Table 1 Experimental constraints.

Constraints	Description
Number of sensor nodes (no_i, i means a sensor network identifier)	Sensor nodes of SN_2 are randomly placed in a whole area of 2,000 x 2,000 meters. The number of sensor nodes is varied from 1,000 to 10,000. Unless otherwise stated, the default number is set to 5,000.
Distribution of spatial query objects (Area ratio)	In the expereriments, the distribution of the spatial query objects (R^P) means an area ratio which can be maximally occupied by the objects without overlapping. The area ratio to the whole area is varied from 1/256 to 1. Unless otherwise stated, two distributions are used. The first is skewed distribution having the ratio of 1/16 where the query objects should be placed within one-sixteenth area of whole area. The second is random distribution having the ratio of 1 where the query objects can be placed anywhere of whole area.
Size of spatial query objects (Distance)	In the query Q1, the size of the spatial query objects is determined by $distance(n_1, n_2) < d$. For example, the size becomes d meters in the Q1. In the experiments, the size is varied from 25 to 200 meters. Unless otherwise stated, a default size is set to 50 meters.
Number of spatial query objects (Selectivity)	In the query Q1, the number of the query objects is influenced by selectivity of the selection predicate f_1. For example, if the selectivity is 10% and number of nodes in SN_1 is 5,000, the number of the query objects will be 500. In the experiments, the number is varied from 50 to 500. Unless otherwise stated, a default number is set to 100.
Packet size	Packets that contains the R^P are pushed down to SN_2, as described in Step 5 of Algorithm 1. In the experiments, a packet size is varied from 32 to 2,048 bytes. Unless otherwise stated, the default size is set to 128 bytes in consideration of a wirless communication protocol such as the IEEE 802.15.4 specification.
Transmission distance of a sensor node	In the experiments, we assume that the transmisssion distances of all sensor nodes are the same. The transmission distance is varied from 50 to 275 meters. Unless otherwise stated, a default distance is set to 100 meters.

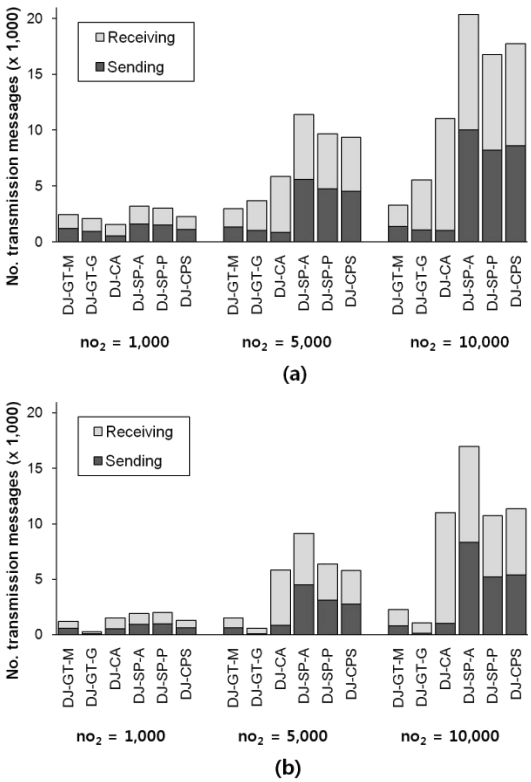

Fig. 7 Comparisons of distributed spatial join algorithms with varying number of sensor nodes of SN_2: (a) random distribution and (b) skewed distribution of R^P.

In SN_2, sensor nodes are randomly placed in a whole area of $2{,}000 \times 2{,}000$ meters.

4.2 Experimental Results for Effectiveness of the Proposed Algorithm

To study the effectiveness of the proposed algorithms, we conducted four experiments using the following constraints: number of sensor nodes, distribution of spatial query objects, size of spatial query objects, and number of spatial query objects.

The first experiment was performed in order to show that the proposed algorithms are effective in terms of the scalability on the number of sensor nodes. We measured $N_{2,\,total}$ with random and skewed distribution when the number of sensor nodes are 1,000, 5,000, and 10,000. The results are summarized in Fig. 7, where the lower and upper bars represent $N_{2,\,sending}$ and $N_{2,\,receiving}$, respectively.

As shown in Fig. 7, the proposed DJ-GT-M and DJ-GT-G perform best in most cases. We can also observe that the performance gap between the proposed algorithms and the others is sharply increased when no_2 is increased. This is because the GR-tree can support efficient spatial search by minimizing overlap between MBRs of sensor nodes, while the other algorithms degrades performance of spatial search in proportion to no_2. In Fig. 7 (a), we can observe that DJ-CA tends to be more efficient than DJ-GT-M and DJ-GT-G, when no_2 is 1,000. This is because

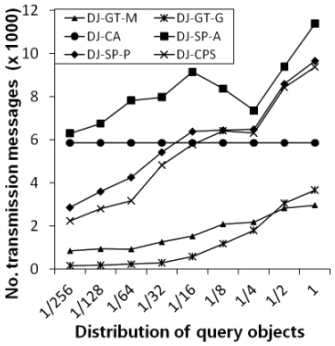

Fig. 8 Comparisons of distributed spatial join algorithms with varying distributions of R^P.

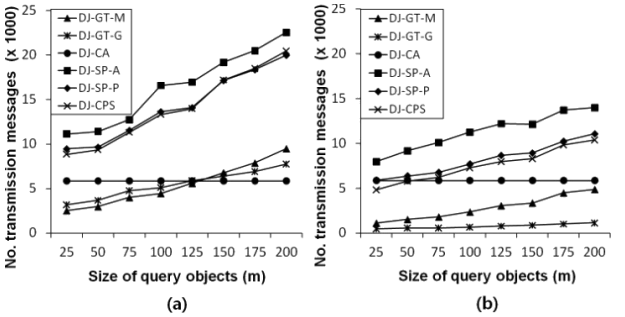

Fig. 9 Comparisons of distributed spatial join algorithms with varying size of spatial query objects: (a) random distribution and (b) skewed distribution of R^P.

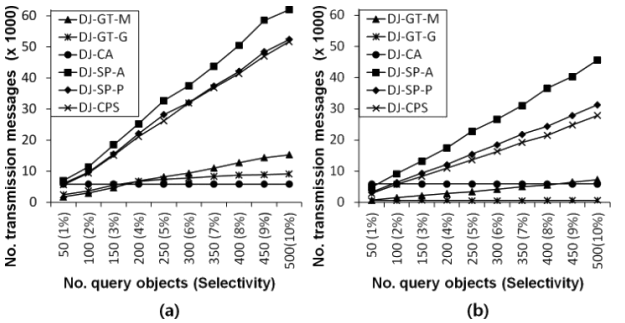

Fig. 10 Comparisons of distributed spatial join algorithms with varying number of spatial query objects: (a) random distribution and (b) skewed distribution of R^P.

performance loss caused by multiple packets transmission overwhelms performance gain obtained by spatial searches of the GR-tree. DJ-CA transmits only a packet (composed of 128 bytes) which includes a selection predicate f_2, while the others transmit at least thirteen packets which include not only f_2, but also R^P composed of 100 rectangle objects (100×4 coordinates $\times$ 4 bytes). In Fig. 7 (b), we can observe that DJ-GT-G outperforms the others. Note that the grid-based approximation is more efficient than the MBR-based approximation in skewed distribution. From this experiment, we can find that DJ-GT-M and DJ-GT-G has better scalability to the number of sensor nodes.

To study the effect of the distribution of query objects, we conducted the second experiment with varying the distribution from 1/256 to 1. Figure 8 shows the result.

As shown in Fig. 8, DJ-CA shows the same performance. This is because DJ-CA always has to visit all sensor nodes, irrespective of varying distributions of R^P. We can also observe that DJ-GT-M and DJ-GT-G shows good performance compared with the others. Here, the size and the number of spatial query objects were set to 50 m and 100, respectively. It means that the small size and small number of spatial query objects compared with the whole area of $2{,}000 \times 2{,}000$ meters will not have a bad effect on the performance of DJ-GT-M and DJ-GT-G, although the query objects are widely distributed throughout the whole area. This is because the GR-tree can support outstanding spatial filtering by minimizing overlap between MBRs of sensor nodes, unless the query objects occupy too much area. If the area occupied by the query objects is getting close to the whole area, we think DJ-GT-M and DJ-GT-G performs worse than DJ-CA. In the next experiments, we will show you what is going on this issue.

To study the effect of the increase of the area occupied by the query objects, we conducted two experiments with varying the size of the query objects from 25 to 100 meters and with varying the number of the query objects from 50 to 500. Figure 9 and Fig. 10 show the results.

From Fig. 9 and Fig. 10, we can observe that two results have a somewhat similar pattern. In random distribution, the performance of all algorithms is getting worse than DJ-CA, when the total area occupied by the query objects is greatly increased. Even DJ-GT-M and DJ-GT-G is getting worse than the DJ-CA. This is because a spatial query on wide area cannot obtain performance gain from spatial filtering, no matter how the GR-tree supports outstanding spatial filtering as R*-tree in SDBMS does. In skewed distribution, however, DJ-GT-M and DJ-GT-G generally performs better than DJ-CA. Also, we can observe that DJ-GT-G outperforms the others. This is because the total query area can be maximally increased up to only 1/16 to the whole area in skewed distribution, as it was defined in Table 1. Moreover, DJ-GT-G consumes lower cost in $N_{2,\,sending}$ than DJ-GT-M. Note that the grid-based approximation is more efficient than the MBR-based approximation in skewed distribution.

From the four experiments, we can find that DJ-GT-M and DJ-GT-G largely outperform the others. We can also find that DJ-GT-G performs best in skewed distribution. When considering practical situations such as military or environmental applications where lots of sensor nodes are widely deployed, selectivity for a spatial query is low and spatial query objects are in skewed distribution, we should note that DJ-GT-M and DJ-GT-G perform better than the others. On the contrary, in case of less sensor nodes deployment at a small region, high selectivity of a spatial query and wide distribution of spatial query objects, we should note that DJ-CA may perform better than DJ-GT-M and DJ-GT-G. Healthcare application for defined patients can be an example for this case.

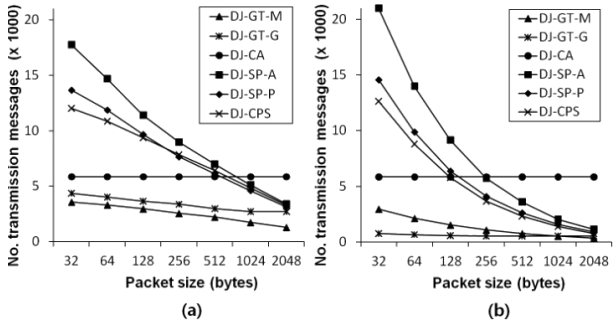

Fig. 11 Comparisons of distributed spatial join algorithms with varying packet sizes: (a) random distribution and (b) skewed distribution of R^P.

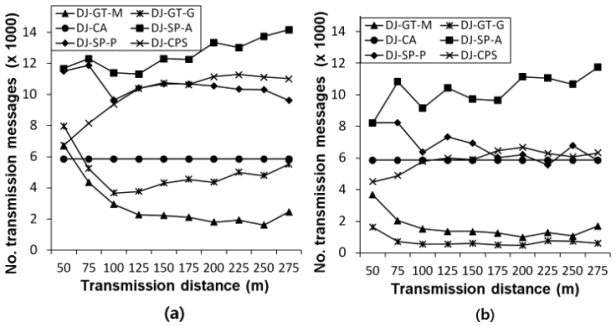

Fig. 12 Comparisons of distributed spatial join algorithms with varying transmission distance of a sensor node: (a) random distribution and (b) skewed distribution of R^P.

4.3 Experimental Results for Availability of the Proposed Algorithm

To study the availability for a sensor network of the proposed algorithms, we conducted two more experiments using the following constraints: packet size and transmission distance of a sensor node.

We performed the fifth experiment to show that our algorithm is available for change of a packet size. In this experiment, we measured $N_{2,total}$ with varying the packet size from 32 to 2,048 bytes. Figure 11 shows the results.

From Fig. 11, we can see two interesting observations. First, all algorithms which build a distributed spatial index gradually perform better than DJ-CA, as the packet size is getting increased. This is because a large-sized packet which can contain many spatial query objects can reduce $N_{2,sending}$. In practical situations, the packet size, however, cannot be infinitely increased, since its error rate will be exponentially increased. For example, the error rate of a packet in wireless networking is $1-(1-p)^n$, where p is the error rate of a bit and n denotes the number of bits composing a packet. As defined in Table 1, a packet of 128 bytes is generally used. Second, we can observe that DJ-GT-M and DJ-GT-G always perform better than the others. This is because the performance gain obtained by the GR-tree overwhelms the performance loss caused by multiple packets transmission, when default values for other constraints are applied. Note that the small size and small number of spatial query objects will not have an effect on the degradation of the performance.

To study the availability of our algorithm when transmission distance of a sensor node is changed, we conducted this experiment. The increase of the transmission distance is likely to incur reduction of a depth of the GR-tree and such reduction may improve performance of a spatial search. On the other hand, the increase may destroy spatial proximity of the GR-tree index structure and such destruction may degrade performance of a spatial search. Therefore, concerned with the transmission distance, we can find that there is a tradeoff between the tree depth and the spatial proximity. In other words, we cannot predict $N_{2,total}$ for the changes of the transmission distance. Figure 12 shows the results of this experiment with varying the transmission distance from 50 to 275 meters.

As shown in Fig. 12, DJ-GT-M and DJ-GT-G performs best in most cases fortunately, although there is a tradeoff between the tree depth and the spatial proximity. However, we can observe that DJ-CA tends to be more efficient than DJ-GT-M and DJ-GT-G when the distance is too short (below about 50 meters). This is because the short transmission distance prevents building an optimized GR-tree index structure. In other words, it means that a node cannot select a parent which optimizes the GR-tree index, since the node has few candidate parents. Here, the GR-tree is likely to form a general routing tree rather than a geographical routing tree.

From the two experiments, we can find that DJ-GT-M and DJ-GT-G are generally efficient than the others. Especially, when we consider practical situations of the packet size of 128 bytes and the transmission distance of 100 meters, we can observe that DJ-GT-M and DJ-GT-G can be directly applied to specific applications which deploy many sensor nodes. Here, we refer to a standard such as IEEE 802.15.4 specification for the practical situations. However, in case the transmission distance becomes too short in random distribution of spatial query objects, DJ-CA is more applicable to applications. This is because the parent selection for GR-tree optimization is closely related with the transmission distance.

5. Discussion

Until now we showed that our algorithms can be used at a distributed spatial join processing between two sensor networks. In addition, the algorithms can be applied to a distributed spatial join processing between a network and a spatial database. The example query is as follows:

(Q2) Consider a sensor network and a spatial database, where the network consists of sensor nodes for CO density monitoring and the database consists of a cadastral map having factories information. We want to collect a set of tuples of the form $<c, l, id_1, f, g>$ (which indicates CO density c of sensor node id_1 in location l and factory geometry g

of factory name f) that satisfies the following conditions: $c \geq 8$ ppm and within (g, l). Here, within (g, l) means that the node location l is contained in the factory geometry g. Informally speaking, this query finds a set of factories with their CO density where CO density exceeds certain threshold value.

This query can be computed by replacing SN_1 with a spatial database in Step 1 and by omitting Step 3 in the Algorithm 1. In the query Q2, we should note that the spatial approximation function of Step 4 plays more important role in processing a join query, since the factory information has more complex type of spatial objects.

6. Conclusion

In this paper, we presented the distributed spatial join algorithm using a spatial semijoin concept. In order to optimize the algorithm, we also proposed the GR-tree and the grid-based approximation function. The GR-tree can determine whether any of the sensor nodes have to participate in the in-network spatial join. The grid-based approximation can reduce the volume of spatial query objects which are pushed down to a sensor network.

Our experimental results showed that DJ-GT-M and DJ-GT-G generally outperform the others in situations where many sensor nodes are widely distributed, the selectivity for a spatial query is low, and the packet size and the transmission distance of a node is not too small. Therefore, we can apply the proposed algorithms to environmental and military applications which require wide deployment of many sensor nodes. However, our proposed algorithms cannot be applied to applications which incur the following situations:

- Too small number of nodes are sparsely deployed,
- Many packets transmissions are required owing to too small packet size,
- Too many results are selected owing to large-sized and large number of spatial query objects,
- Frequent sensor nodes failures happen in a network.

We plan to extend this work in two directions. First, we would like to extend our algorithms to continuous spatial join processing. We expect that a dynamic update scheme which is related to the GR-tree will be needed for the continuous join processing. Second, we will modify the GR-tree algorithm to support mobility of sensor nodes. We think that a new method that can preserve the GR-tree consistently on mobile nodes at low cost will be needed.

Acknowledgments

This work was supported by the grant (07KLSGC05) from Cutting-edge Urban Development — Korean Land Spatialization Research Project funded by MLTM.

References

[1] P. Bonnet, J. Gehrke, and P. Seshadri, "Towards sensor database systems," Proc. 2nd Int'l Conf. Mobile Data Management (MDM 01), LNCS 1987, pp.3–14, 2001.
[2] S. Madden, M.J. Franklin, and J.M. Hellerstein, "TinyDB: An acquisitional query processing system for sensor networks," ACM TODS, vol.30, no.1, pp.122–173, 2005.
[3] C. Srisathapornphat, C. Jaikaeo, and C. Shen, "Sensor information networking architecture," Proc. Int'l Workshop Parallel Processing, pp.23–30, IEEE CS Press, 2000.
[4] S. Li, S. Son, and J. Stankovic, "Event detection services using data service middleware in distributed sensor networks," Proc. 2nd Int'l Workshop Information Processing in Sensor Networks (IPSN 03), LNCS 2634, pp.502–517, Springer, 2003.
[5] S. Madden, M.J. Franklin, J.M. Hellerstein, and W. Hong, "TAG: A tiny aggregation service for ad-hoc sensor networks," Proc. 5th Symposium on Operating Systems Design and Implementation, 2002.
[6] J. Considine, F. Li, G. Kollios, and J. Byers, "Approximate aggregation for sensor databases," Proc. 20th Int'l Conference on Data Engineering, pp.449–460, 2004.
[7] A. Deligiannakis, Y. Kotidis, and N. Roussopoulos, "Hierarchical in-network data aggregation with quality guarantees," Int'l Conference on Extending Database Technology, pp.658–675, 2004.
[8] A. Deligiannakis, Y. Kotidis, and N. Roussopoulos, "Compressing historical information in sensor networks," ACM SIGMOD, pp.527–538, 2004.
[9] M.A. Sharaf, J. Beaver, A. Labrinidis, and P.K. Chrysanthis, "Balancing energy efficiency and quality of aggregate data in sensor networks," VLDB Journal, vol.13, no.4, pp.384–403, 2004.
[10] D.J. Abadi and S. Madden, "REED: Robust, efficient filtering and event detection in sensor networks," Proc. 31st Int'l Conference on VLDB, pp.769–780, 2005.
[11] U. Srivastava, K. Munagala, and J. Widom, "Operator placement for in-network stream query processing," Proc. 24th ACM PODS, pp.250–258, 2005.
[12] A. Soheili, V. Kalogeraki, and D. Gunopulos, "Spatial queries in sensor networks," Proc. 13th ACM GIS, pp.61–70, 2005.
[13] K.L. Tan and B.C. Ooi, "Exploiting spatial indexes for semijoin-based join processing in distributed spatial databases," IEEE Trans. Knowl. Data Eng., vol.12, no.6, pp.920–937, 2000.
[14] D.J. Abel, B.C. Ooi, K.L. Tan, R. Power, and J.X. Yu, "Spatial join strategies in distributed spatial DBMS," Proc. 4th Int'l Symposium on Large Spatial Databases, pp.348–367, 1995.
[15] M.L. Yiu, N. Mamoulis, and S. Bakiras, "Retrieval of spatial join pattern instances from sensor networks," 19th Int'l Conference on Scientific and Statistical Database Management, pp.25–34, 2007.
[16] X. Zhu, H. Gupta, and B. Tang, "Join of multiple data streams in sensor networks," Technical Report, http://www.cs.sunysb.edu/~hgupta/pubs.html, 2007.
[17] S. Madden, M.J. Franklin, and J.M. Hellerstein, "The design of acquisitional query processor for sensor networks," Proc. ACM SIGMOD, pp.491–501, 2003.
[18] A. Guttman, "R-trees: A dynamic index structure for spatial searching," Proc. ACM SIGMOD Int'l Conf. Management of Data, pp.47–57, 1984.
[19] T. Brinkhoff, H. Kriegel, R. Schneider, and B. Seeger, "The R*-tree: An efficient and robust access method for points and rectangles," Proc. ACM SIGMOD Int'l Conf. Management of Data, pp.322–331, 1990.
[20] M. Kim, J.W. Lee, Y.J. Lee Kim, and J.C. Ryou, "COSMOS: A middleware for integrated data processing over heterogeneous sensor networks," ETRI Journal, vol.30, no.5, 2008, pp.696–706.
[21] T. Brinkhoff, H. Kriegel, and B. Seeger, "Efficient processing of spatial joins using R-trees," Proc. ACM SIGMOD Int'l Conf. Management of Data, pp.237–246, 1993.

[22] O. Gunther, "Efficient computation of spatial joins," Proc. IEEE Int'l Conf. Data Eng., pp.50–59, 1993.
[23] T.M. Gil and S. Madden, "Scoop: An adaptive indexing scheme for stored data in sensor networks," Proc. 23rd Int'l Conference on Data Engineering, pp.1345–1349, 2007.
[24] J.H. Jeon, K.Y. Lee, J.S. Yoo, and M.H. Kim, "HIPaG: An energy-efficient in-network join for distributed condition tables in sensor networks," J. Syst. Softw., vol.82, no.7, pp.1073–1086, 2009.
[25] A. Meka and A. Singh, "DIST: A distributed spatio-temporal index structure for sensor networks," Proc. ACM CIKM, pp.139–146, 2005.

Min Soo Kim received the B.S. and M.S. degrees in Computer Science from Pusan National University, Pusan, Korea in 1994 and 1996, respectively. Since 1996, he has been a research member of Electronics and Telecommunications Research Institute (ETRI). He is also a Ph. D. student in the Department of Computer Science, KAIST, Daejeon, Korea. His research interests include Wireless Sensor Networks, GeoSensor Networks, sensor network database, spatial database and ubiquitous computing.

Jin Hyun Son received Ph.D degree in Computer Science from KAIST, Daejeon in 2001 and M.S. degree in Computer Engineering from KAIST in 1998. He received B.S. degree in Computer Science from Sogang University in 1996. He is currently a professor in the Department of Computer Science and Engineering, Hanyang University, Ansan, Korea. His research interests include business process management, database, distributed systems, and ubiquitous systems.

Ju Wan Kim received Ph.D degree in Computer Science from Chungnam National University, Daejeon in 2004 and the B.S. and M.S. degrees in Computer Engineering from Pusan National University, Pusan, Korea in 1993 and 1995, respectively. He has joined Electronic and Telecommunications Research Institute (ETRI) from 1995. He is currently a team leader in ETRI. His research interests include Wireless Sensor Networks, GIS, LBS, Telematics, and GeoSensor networks.

Myoung Ho Kim received Ph.D degree in Computer Science from Michigan State University, East Lansing, Michigan, USA and the B.S. and M.S. degrees in Computer Engineering from Seoul National University, Seoul, Korea in 1982 and 1984, respectively. He is currently a professor in the Department of Computer Science, KAIST, Daejeon, Korea. In 1995, he was a visiting professor of University of Virginia. His research interests include database, distributed systems, XML, multimedia, sensor networks. He is a member of KISS and IEEE.