# Optimizing Data Placement Over Wireless Broadcast Channel For Multi-Dimensional Range Query Processing

Jianting Zhang Le Gruenwald
The University of Oklahoma, School of Computer Science, Norman, OK, 73019
Contact author email: ggruenwald@ou.edu, Phone: 1-405-325-3498

## Abstract

Data broadcasting is well known for its excellent scalability. Multi-dimensional range queries, such as spatial range queries of geographical information for location dependent services, are very popular queries in mobile computing. Query response time is greatly affected by the order in which data items are being broadcast. This paper proposes a non-greedy, low polynomial time cost optimization method to place data over a wireless broadcast channel for multi-dimensional range query processing. Experimental results show that the method, together with proper global constraints based on application semantics, can greatly reduce total access time to the data channel. Compared with the heuristic data placement methods purely based on the access frequencies, the reduction of access time can be as much as 54%.

## 1. Introduction

Data broadcasting is well known for its excellent scalability (*Imielinski, 1997*). Previous work on data broadcasting has focused on retrieving a single data item from a broadcast channel while little work has been done on complex queries where a query result set has multiple data items (Lee 2002). It is obviously inefficient to retrieve the data items in a complex query by accessing the broadcast channel multiple times and retrieving one data item at a time. In this study, we consider a typical broadcast scenario introduced in (*Imielinski, 1997*), where data and its index are multiplexed into a single channel and index is placed ahead of data.

Range query processing has been extensively studied in disk-resident databases (Gaede, 1998). A range query often returns multiple data items . Although it is straightforward to map one-dimensional data items onto a broadcast channel for range queries based on their values, it is not a trivial issue to make such a mapping for multi-dimensional data items for efficient range query processing. The multi-attribute indexing method (*Imielinski, 1997*) proposed to fragment the first attribute and build an index for each fragment of the first attribute based on the values of a second attribute. These indices are distributed sequentially on a broadcast sequence. Data items that are indexed are placed right after the indices. As far as indexing is concerned, this is actually using one-dimensional indexing methods consecutively for multidimensional indexing and thus is inefficient (Kriegel, 1984). We are not convinced that the sequence of the data items generated by the method is efficient for range queries either since no access frequencies of data items are considered. As discussed in (Tan, 1998), unlike queries that involve only a single data item, replicating data items, as in many broadcast-disk based techniques, generally will not help reducing access time in processing range queries. Thus we assume that both data and index are broadcast only once.

In this study, we are interested in reducing total access time to the data channel for processing multi-dimensional range queries by better placement of data items on a broadcast channel. We represent all possible complex query result sets as hyper-edges in a hyper-graph where the data items are represented as nodes. We treat the access frequencies of the query result sets as the weights of the corresponding hyper-edges. An efficient non-greedy data placement method based on an approximation method for the graph Minimum Linear Arrangement (MinLA) problem (Bar-Yehuda, 2001) is proposed. Experimental results show that the method, together with proper global constraints imposed on the hyper-graph based on application semantics, can greatly reduce total access time to the data channel. Compared with the heuristic data placement methods purely based on the access frequencies, the reduction of access time can be as much as 54%.

The rest of this paper is arranged as follows. Some background in data broadcast is introduced in Section 2 and then related work is reviewed in Section 3. Section 4 describes the optimization method. Section 5 presents the experiments comparing the proposed method and other heuristic methods. Finally Section 6 is conclusions and future work directions.

## 2. Background

In a broadcast system, a minimum logical unit in a broadcast sequence is called a bucket/frame, and a set of continuous buckets (either index or real data) is called a segment. Different from main memory or disk resident data accesses, accesses to a broadcast sequence are essentially one-dimensional. There are two important parameters in evaluating the performance of a broadcast

Proceedings of the 2004 IEEE International Conference on Mobile Data Management (MDM'04)
0-7695-2070-7/04 $20.00 © 2004 IEEE

system, namely Tune-in Time (TT) and Access Time (AT, or latency). TT is the amount of time a client spent listening to the broadcast channel. AT is the average time elapsed from the time a client requests data to the time when the client downloads all the required data . In Fig. 1, TT is equal to the summation of the lengths of the required data items (shaded) while AT is the duration between the initial access position and the last required data item.

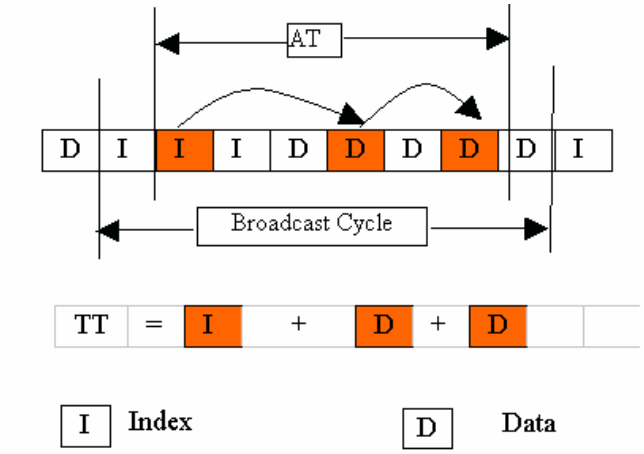


**Fig. 1. Illustration of TT and AT**

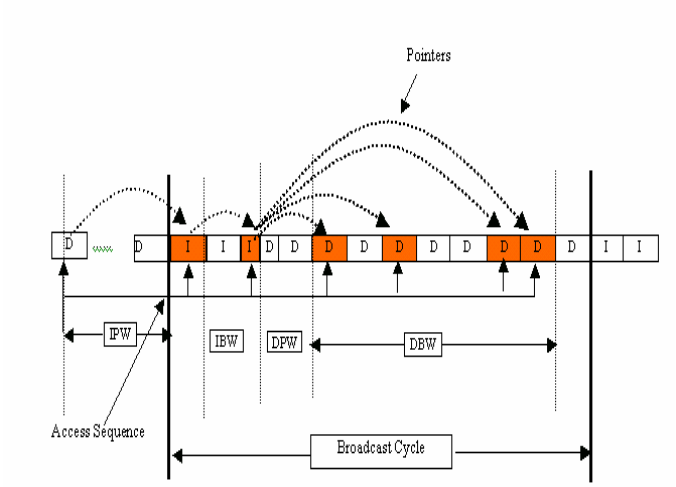


**Fig. 2. The Four Components in Access Time**

In (Imielinski, 1997), AT is the sum of the Probe Wait (PW) and the Bcast Wait (BW) where the former is the average duration for getting to the next index segment and the latter is the average duration between the time when the index segment is encountered and the time when all the required data items are downloaded. We argue that it might be more appropriate to divide AT into four components: Index-Probe Wait (IPW), Index-Bcast Wait (IBW), Data-Probe Wait (DPW) and Data-Bcast Wait (DBW). IPW is the same as PW. IBW is the time duration from the time when the first index segment is met to the time when the last index segment is met. DPW is defined as the duration from the time the last index segment is reached to the time when the first data segment is reached. DBW is defined as the duration from the time when the first data segment is reached to the time when the last data segment is downloaded. The summation of IBW, DPW and DBW is equivalent to BW. Using the four components allows us to compute access time to index and data separately. We assume a client has already had an ordered set of pointers to the data items in the broadcast channel by performing a range query on the index segments which are either in the same channel with the data or in a separate index channel. The scenario we consider is the one in which the index and data are multiplexed into a single channel where a client needs to go to the beginning of the multiplexed channel before retrieving data items. These four components under this scenario are illustrated in Fig 2.

By using these four components we are able to separate the access time to index and the access time to data. In this study we focus on the access time to data, i.e., $AT_{Data}$=DPW+DBW. Let function $\pi(u)$ map data item u to its position in the broadcast sequence, $AT_{Data}$ for processing a single complex query that contains k data items $n_1$, $n_2$ … $n_k$ is $\max\{\pi(n_1),\pi(n_2),...\pi(n_k)\}$. Consequently the total $AT_{Data}$ for processing all possible complex queries is

$$\sum_{q\in Q} w(q) * \max\{\pi(n_1),\pi(n_2)...\pi(n_k)\}$$

where $Q$ is the set of all possible complex queries and $w(q)$ is the weight of a single complex query $q$.

## 3. Related Work

There have been several studies on data broadcast. Many of them have focused on indexing techniques to make tradeoffs between TT and AT, such as tree-indexing (Imielinski, 1994a), hashing (Imielinski, 1994b), signature (Lee, 1996) and hybrid (Hu, 2001a). They can support only queries on one-dimensional data and can search only one data item in a query result. Although (Imielinski, 1997) proposed to chain data items that have the same values in different meta-segments in its non-clustering index and multi-index methods, it cannot be applied to data items that have different values but are often in the same query results (e.g. range queries on attributes that have continuous values). Furthermore, in its performance analysis, it assumed that it takes a whole broadcast cycle to retrieve non-clustered data items of a particular value. That is an unnecessary

overestimation. The issue of multi-attribute data broadcast and query was first addressed in (Hu, 2001b). However, this work can handle only conjunction/disjunction queries that involve fewer than three attributes. They are not suitable for multi-dimensional range queries.

Recent works on object-oriented database broadcast (Chehadeh, 1999) and relational database broadcast (Lee, 2002) allow multiple data items to be accessed in a query. However, they assumed the access to data items has predefined orders. They are not suitable for range queries since data items in a query result do not necessarily have a predefined order. Furthermore, the ordering heuristics proposed in (Chehadeh, 1999; Lee, 2002) are greedy. The work presented in (Chung, 2001) handles both DPW and DBW. However, it assumed that a user begins to access the data channel randomly which is different from the scenario we are considering where a user always goes to the beginning of a broadcast channel to retrieve data items. The data placement method proposed in in (Chung, 2001) is also greedy and the data items having scheduled in a previous round will not change their orderings in the following rounds. Our method differs from all these works in that we try to examine an exponential number of orderings in a low polynomial complexity time. The exhaustive searching nature makes our method tend to produce orderings with high quality.

# 4. The Optimization Method

## 4.1. Basic Ideas

Our method is motivated by the approximation algorithm for the graph Minimum Linear Arrangement (MinLA) problem proposed in (Bar-Yehuda, 2001). The goal in MinLA is to find an ordering that minimizes the weighted sum of the edge lengths in a sequence. Formally, the weighted sum of the edge lengths with respect to an ordering is defined as

$$la(G) = \sum_{(u,v)\in E} w(u,v) * | \pi(u) - \pi(v) |$$

where w(u,v) is the weight of the edge (u,v) and π(u) is the position mapping function as defined in Section 2. By relating w(u,v) with w(q) and relating |π(u)- π(u)| with $\max\{\pi(n_1),\pi(n_2),...\pi(n_k)\}$ we can see that the problem of optimizing the total access time is structurally similar to the MinLA problem. However, there are several significant differences. First, there are multiple data items in a complex query result set (q) while there are only two nodes associated with an edge (u and v). This difference may be solved by extending a regular edge that only consists of two nodes to a hyper-edge that consists of a subset of the vertex set of the graph. We can define π(u)= $\max\{\pi(n_1),\pi(n_2),...\pi(n_k)\}$ and π(v)= $\min\{\pi(n_1),\pi(n_2),...\pi(n_k)\}$. Second, the length of an edge is defined as $\max\{\pi(n_1),\pi(n_2),...\pi(n_k)\}$ in our problem while it is |π(u)- π(u)| in the MinLA problem. Third, rather than computing |π(u)- π(u)| for an edge directly, (Bar-Yehuda, 2001) computes it as a serial summation of the sub-tree sizes of the Binary Decomposition Tree (BDT) of a graph (see Section 4.2 below for a detailed description of BDT) to achieve its efficiency while this is not possible in our problem.

In this study, we adopt the divide-and-conquer strategy and several efficient graph data structures that have been used in (Bar-Yehuda, 2001). We propose a new method to minimize the total access time for all possible multi-dimensional query result sets under the broadcast scenario that data and index are multiplexed into a single channel. Like (Bar-Yehuda, 2001), our method checks all possible $2^{n-1}$ orderings that can be derived from a BDT in $O(n^2)$ time complexity.

We next introduce the BDT which is the global constraints imposed on the ordering of data items. We then present the optimization method and illustrate the process through a simple example. Before discussing the complexity analysis of our proposed method, we discuss several efficient graph data structures that are crucial to achieve low complexity.

## 4.2. The Binary Decomposition Tree

A BDT T (Fig. 3) is a binary tree that has all the nodes in a hyper-graph as its leaf nodes. For each tree $t \in T$ that has two sub-trees $t_1$ and $t_2$, we have two options in placing the nodes under it into a broadcast channel, i.e., either the nodes under $t_1$ are placed ahead of the nodes under $t_2$ (called 0-orientation), or the nodes under $t_2$ are placed ahead of the nodes under $t_1$ (called 1-orientation). The orientations at each intermediate node of the BDT form a tree that has the same structure as that of the BDT. An orientation tree determines an ordering sequence of all the nodes in a graph.

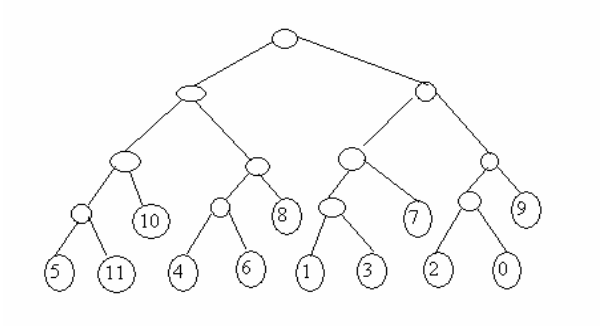


**Fig. 3. A Binary Decomposition Tree**

Since t has two orientations and the orientations of its two sub-trees, $t_1$ and $t_2$, are independent of each other, it is easy to prove that there are $2^{n-1}$ orderings for a full and balanced BDT. The efficiency is achieved by examining $2^{n-1}$ orderings in $O(n^2)$ time as shown below.

### 4.3 The Proposed Method and an Example

The proposed method is shown in Fig. 4. We postpone the discussions of Step 3.b and Step 3.c until Section 4.4 since we need several efficient graph data structures. However, there should not be any problem in understanding the optimisation process through the following example.

1 Set the positions of all nodes to the specified initial order, or the natural order of {1,2…n} if no initial order is available. Set the starting BDT node *t* to the root of BDT. Do step 2 and step 3 recursively.

2 If *t* is an intermediate node of BDT:
   a) Test the two orientations of the sub-trees $t_1$ and $t_2$ it points to by adding the access time of $t_1$ and $t_2$ under the orientations.
   b) Set the orientation of *t* to the one that has less access time.

3 If *t* is a leaf node of BDT:
   a) Set the access time associated with the node to zero.
   b) Compute the position of the node.
   c) Retrieve all the queries that contain this node and their corresponding weights.
   d) For each query that has the node as the ending node in the broadcast sequence, add position*weight to the access time associated with the node.

**Fig. 4. The Process of the Optimization Method**

The example we use to illustrate the optimization method has 4 data items and 11 queries (hyper-edges). The queries and their access frequencies (hyper-edge weights) are listed in Table. 1. The BDT we use is shown in Fig 5. For illustration convenience, we also use "+" to denote the1-orientation and "-" to denote the 0-orientation.

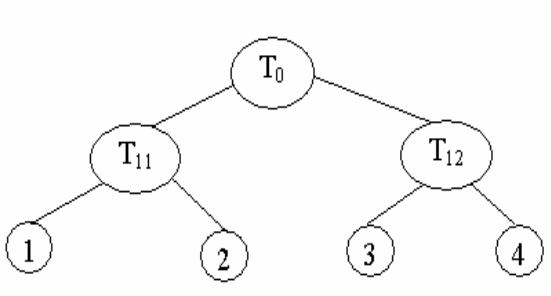


**Fig. 5. The BDT of the Example**

**Table 1. The Hyper-graph of the Example**

| | |
|---|---|
| 1 | 62 |
| 2 | 19 |
| 3 | 34 |
| 4 | 85 |
| 1,3 | 2 |
| 2,3 | 38 |
| 1,2 | 22 |
| 3,4 | 8 |
| 2,4 | 3 |
| 1,2,3 | 14 |
| 2,3,4 | 4 |

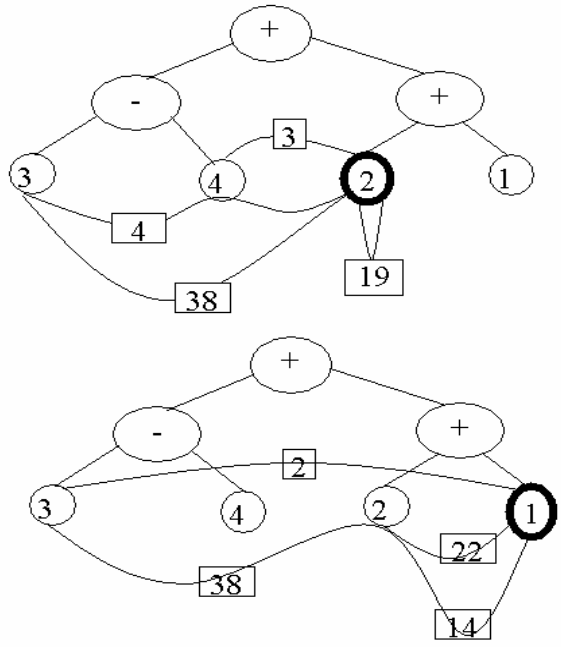


**Fig. 6. Computing AT(2) and AT(1) Under 1-Orientation of $T_{11}$**

For $T_{11}$ in 1-orientationas shown in Fig. 6, the position of node 2 is 2 and the position of node 1 is 3, thus AT(2) = 2*( 4 + 38 + 19 + 3) = 128, AT(1)=3*(2+14+22+62) =300 and $AT(T_{11}^{1})$=128+300=428. Similarly we can also compute $AT(T_{11}^{0})$=428. According to our convention, we will choose 0-orientation if the two orientations have the same cost. We next compute the cost of $T_{12}$ under both orientations.

For the $T_{12}$ in 1-orientation as shown in Fig 7, the position of node 3 is 1 and the position of node 4 is 0, thus AT(3)=1*(8+34)=42, AT(4)=0 and $AT(T_{12}^{1})$=42+0=42. Similarly for $T_{12}$ in 0-orientation, AT(3)=0, AT(4)= 1*(8+85)= 93, thus $AT(T_{12}^{0})$=0+93=93. Since 42 is less than 93, 0-orientation of $T_{12}$ is the winner. Thus the total cost of $T_0$ under 1-orientation is $AT(T_0^{1})$= $AT(T_{11}^{1})$+ $AT(T_{12}^{0})$ = 428+42= 470. Similarly the $AT(T_0)$ under 0-orientation is 517. Thus the 1-orientation of $T_0$ is the winner and the final optimized ordering is [4,3,1,2] whose access time is 14.9% better than the access time of the natural ordering of [1,2,3,4].

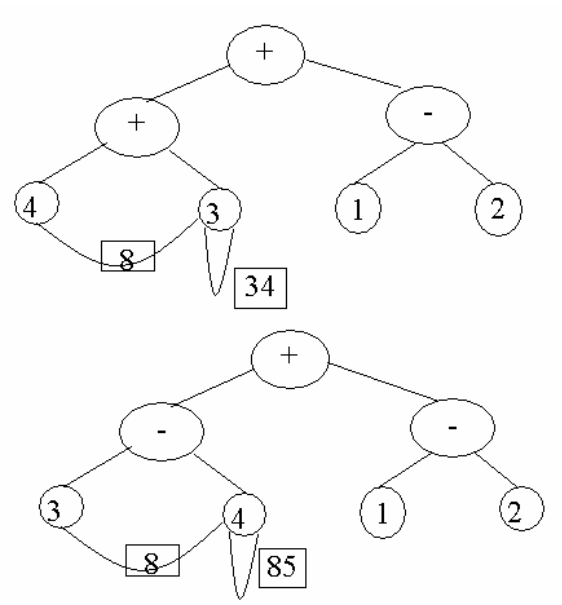


**Fig. 7 Computing $AT(T_{12})$ Under 1- and 0-Orientations**

## 4.4 Hyper-Graph Data Structures

Since we represent all possible query result sets as a hyper-graph, we next introduce several efficient graph data structures that are crucial in achieving efficiency for our method. Some of them have also been used in the implementation of (Bar-Yehuda, 2001).

First, the nodes of the hyper-edges are stored sequentially in an array and the positions that mark the endings of the hyper-edges in the array are stored in another array. These positions serve as the indexes to the nodes in the hyper-edges. The weights of the hyper-edges are stored in a third array. The fourth array associated with the hyper-graph contains the pointers to the leaf nodes in the BDT. An inverse hyper-graph is also built for the graph. It has an array to store sequentially the hyper-edge IDs that contain the nodes in the original graph. Similar to the index array used in a hyper-graph, the positions that mark the endings of the nodes are stored in the second array of the inverse hyper-graph.

By using the arrays of the hyper-graph and the corresponding inverse hyper-graph, we can perform the following operations efficiently. First, we can retrieve all the nodes in a hyper-edge of ID *e* by first retrieving the ending position of *e*-1 and the ending position of *e* and then retrieving all the nodes in the node array of the hyper-graph. Second, we can retrieve all the hyper-edges that contain a node *v* by first retrieving the ending position of *v*-1 and the ending position of *v* and then retrieving all the IDs of the hyper-edges between the two positions in the index array of the inverse hyper-graph. Note that we can retrieve the weight of a hyper-edge by accessing the weight array of the hyper-graph using the hyper-edge's ID directly.

We next discuss the BDT in more detail. Each node in the BDT contains a pointer to its parent and two pointers to its two children. The parent pointer of the root of the BDT is empty and the two children pointers of a leaf node of the BDT are also empty. Each node also contains an orientation flag and the size of the sub-tree of the BDT that has the node as the root. Now we are ready to show how to compute the position of a node under a BDT efficiently.

As shown in Fig. 8, starting at a leaf node (6 for example), we use the parent pointer associated with the BDT node to travel from the leaf node all the way to the root of the BDT. We check the orientation of the BDT nodes along the path. If the sub-tree rooted at the node is the right sub-tree of its parent then we add the sub-tree size of its sibling to the position, otherwise we just skip. In the BDT shown in Fig. 8, in the first step, since node 6 is the right child of $T_1$ we add 1 to the position. In the second step, since $T_1$ is the left child of $T_2$, we just skip. In the third step, since $T_2$ is the right child of $T_3$, we add 3, which is the size of the left sub-tree of $T_3$, to the position. Since $T_3$ is the left child of $T_4$, which is the root of the BDT, we skip again. Finally we get the position of node 6 as 3+1=4. The cost of computing the position of a leaf node is in the order of $\log^n$ if the BDT is balanced.

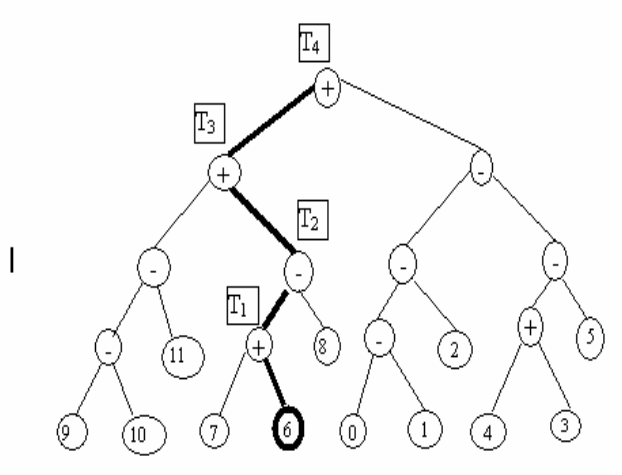


**Fig. 8. Illustration of Computing the Position of a BDT Node**

We next show how to determine whether a node is the ending node of a hyper-edge. We need some pre-processing. For each hyper-edge we build a tree called the Least Common Ancestor (LCA) tree. The process of constructing a LCA tree is as follows. For each of the node in the hyper-edge, we travel from the node all the way to the root of the BDT as we did before and assign the ID of the hyper-edge to the flags of all the intermediate nodes on the paths. Next, starting from the root of BDT, we first try to find a node of the BDT whose both children's flags are the ID of the hyper-edge and travel down to only the child whose flag is the ID of the hyper-edge. We repeat this process recursively until we reach the leaf nodes of the BDT. We add a BDT node to the LCA tree if its both children's flags are the ID of the hyper-edge or it is the leaf node of the BDT. Fig. 9 shows the process where the ID represents the hyper-edge of {1,4,5,6} and the dashed lines show the correspondence between the nodes in the BDT and the LCA tree. Note that each LCA tree is topologically a sub-tree of the BDT. Since each LCA tree node stores a pointer to a copy of the corresponding BDT node, the actual order of the nodes in a hyper-edge represented by a LCA tree will also change when the orientation of a related BDT node is switched.

By using the LCA tree, we can determine efficiently whether a leaf node of the BDT is the ending node of a hyper-edge. We first start with the root of the LCA tree of the node. We then follow the right children until we reach a leaf node of the LCA tree to find the last node in the hyper-edge under the current orientations of the BDT. We compare the last node's ID with the ID of the BDT node to determine whether the node represented by the leaf BDT node is the ending node of the hyper-edge. The cost of the traversal is in the order of $log^m$ where $m$ is the number of nodes in the hyper-edge. We assume the maximum number of the nodes in the hyper-graph is bounded by a constant, thus the determination can be made in small constant time.

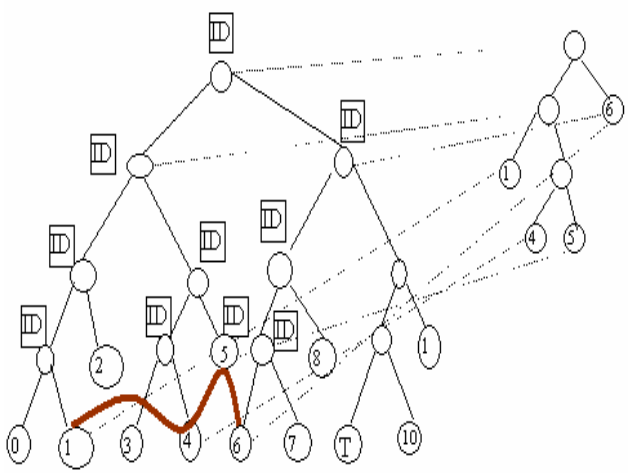


**Fig. 9. Illustration of Determining the Ending Node of a Hyper-Edge**

## 4.5 Complexity Analysis

We are now ready to analyse the computation complexity of the proposed method. Let S(N) be the total computation cost for an N-nodes hyper-graph. Assuming the corresponding BDT T is balanced. At each $t \in T$ having $n$ nodes (i.e. $n=|t|$) we need to calculate the costs of its two children under the two orientations which result in $4*S(n/2)$ We also need one addition for each orientation (to add the costs of $t_1$ and $t_2$) and one comparison (to compare the costs of the two orientations). Thus the complexity analysis of the total access time in terms of the number of data items $n$ is shown as in Fig. 9.

S(1) has the following components. It takes $O(\log^n)$ to compute the position of a node in Step 2.b as discussed in Section 4.4. It takes constant time to retrieve all the hyper-edges that contain the node by using the inverse hyper-graph in Step 2.c. Also as discussed in Section 4.4, it takes constant time to determine whether a node is the ending node of a hyper-edge, thus the total cost in Step 2.d is also bounded by a constant. Furthermore, $n$ varies from 100 to 10000 in many real applications, thus $log^n$ varies from 7 to 14, which is comparable to the multiplication of the average number of hyper-edges that contain a node (10-20) and the average depth of a LCA tree (3-5). We treat $log^n$ as a bounded constant for the practical values of $n$. Thus we can claim that our method has the complexity of $O(n^2)$.

$$S(n) = 4 * S(\frac{n}{2}) + 3$$

$$= 4 * [4 * S(\frac{n}{2^2}) + 3] + n + 3$$

$$= 4^2 * S(\frac{n}{2^2}) + 4 * 3 + 3$$

$$= \ldots$$

$$= 4^k [S(\frac{n}{2^k}) + 3] + \ldots + 4 * 3 + 3$$

$$= 4^k * S(\frac{n}{2^k}) + \ldots + 4^k * 3 + 4 * 3 + 3$$

$$= 4^k * S(1) + (4^k + 4^{k-1} + \ldots 1) * 3$$

$$= (2^k)^2 * S(1) + \frac{4^{k+1} - 1}{4 - 1} * 3$$

$$= n^2 * S(1) + 4 * n^2 - 1$$

**Fig. 9. Complexity Analysis**

## 5 Experiments for Performance Evaluation

### 5.1 Generating Data Sets

In addition to the two parameters in generating random data sets, i.e., the number of data items (M) and the number of queries (Q), we also use the parameter of the maximum number of data items in a query (N) to comply with our assumption in the complexity analysis. In this study, we set M=100, Q=100 and N=10. We assume our data are two- dimensional whose data space is (0,1) by (0,1) and the access frequency for each query is between 0 and 100. Instead of generating query ranges directly, we first pick seed data items and generate random numbers between 1 and N. We then retrieve the N-nearest data items of the seed data items and use their Minimum Bounding Rectangles (MBR) as the query ranges. By doing so we free ourselves from the time-consuming try-and-error processes to make sure that the number of data items in a generated query result set is less than N. We generate 100 such data sets and perform evaluation of our proposed method on them in the following sub-sections. All our experiments are performed on a Dell Dimension 4100 personal computer with 866MHZ processor and 512M memory. For the rest of this section, when we say "A is *c*% better than B" we compute *c* as the percentage of the reduced access time from the broadcast ordering given by method B to the broadcast ordering given by method A divided by the access time of the broadcast ordering given by method A, i.e., *c*=(B-A)/A=B/A-1.

### 5.2 Optimization of R-Tree Ordering

R-Tree (Guttman, 1984), R+-Tree (Sellis, 1987) and R*-Tree (Beckmann, 1990) are well-known multi-dimensional indexing techniques. They are designed to put data items close to each other into the same branch while put data items far away from each other into different branches. Since data items that are close to each other in multi-dimensional space are more likely to be queried together in multi-dimensional queries, using R-Tree as the base is a good candidate to construct a BDT. We replace an *m*-branches R-tree node with a small binary tree and combine all such small binary trees to build a BDT as shown in Fig. 10. The total access time in the 100 data sets for the ordering of traversal of the R-Tree and the optimized ordering are compared in Fig. 12. The average percentage of the reduced access time for the 100 data sets is 9.8% and the maximum of that is 19.2%.

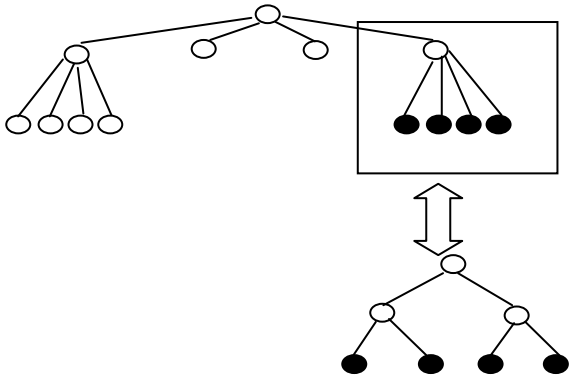

**Fig. 10. Constructing a BDT by Replacing a R-Tree Node with a BDT Sub-Tree**

### 5.3 Optimization of Graph Partition Tree Ordering

Since we represent all possible query result sets as a weighted hyper-graph, it is natural to use graph partition techniques to generate a BDT as used in (Bar-Yehuda, 2001). An illustration of using graph partition to construct a BDT is shown in Fig. 11.

Although graph partition is a NP-complete problem , some heuristics and approximation algorithms are fast enough for our problem where the number of nodes in a graph is typically 100-10000. In this study we use a hyper-graph partition package called HMETIS ([HREF 1]) to perform recursive bisection and generate a BDT accordingly. By taking access frequencies (edge weights) into consideration explicitly, the quality of the BDT and the optimization result are expected to be better than the R-Tree based results. This has been confirmed by our experiments. The results of the same 100 data sets are shown in Fig. 13.

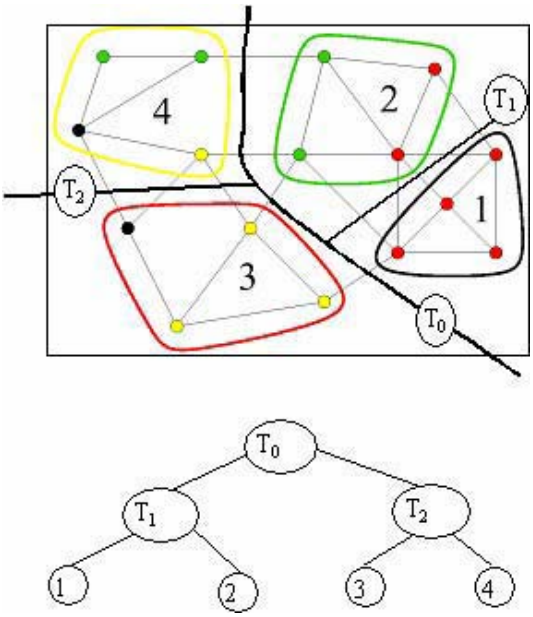


**Fig. 11. Constructing a BDT by Graph Partition**

### 5.4 Comparison with Two Heuristics

We also generate two broadcast orderings using two popular heuristics. The first one is based on the decreasing order of summarized node weights, i.e., we add up all the weights of the hyper-edges that contain a node to get the summarized weight of the node. The data items that are corresponding to the nodes are sequenced according to the decreasing order of their weights. The second heuristic is edge-based. We sort the hyper-edges according to their deceasing weights. All the nodes in the hyper-edge that has the largest weight are placed at the beginning of the broadcast sequence. We then check the hyper-edge that has the second largest weight and place its nodes that have not been placed onto the broadcast sequence. This process is repeated until all the nodes in the hyper-graph representation have been placed onto the broadcast sequence.

Comparisons of the experiment results of the two heuristic orderings with the optimized orderings using R-Tree and graph partition tree as BDTs are shown in Fig. 14. The average total access time of the 100 data sets are almost identical for the two heuristics. The optimized ordering using R-Tree as BDT is on average about 22% better than the two heuristic orderings. Furthermore, optimized ordering using a graph partition tree as the BDT is on average about 54% better than the two heuristic orderings using the 100 data set. Optimized ordering is the best data placement method in our experiments.

## 6 Conclusions and Future work Directions

In this study, we considered the problem of data placement on a broadcast channel for efficient multi-dimensional range query processing. We proposed an efficient optimization method using a divide-and-conquer strategy and several efficient graph data structures that can examine $2^{n-1}$ orderings in $O(n^2)$ time for practical $n$ values that vary from 100 to 10000. Our experiments using 100 synthetic data sets on the two heuristic methods and the optimization method (using R-Tree and graph partition tree as BDTs) suggest that using graph partition tree as the BDT for optimization achieves the best data access time that could be as much as 54% better than the heuristic orderings.

For future work, we plan to take access time to the index channel into consideration and explore more ordering heuristics as well as exact and/or approximation optimization methods. Finally we plan to do more experiments using both synthetic and real data sets with different sizes and distributions to examine the effectiveness and scalabilities of the optimization method.

## References


1. Norbert Beckmann, Hans-Peter Kriegel, Ralf Schneider, Bernhard Seeger: The R*-Tree: An Efficient and Robust Access Method for Points and Rectangles. SIGMOD Conference 1990: 322-331

2. Y. C. Chehadeh, A. R. Hurson, Mohsen Kavehrad: Object Organization on a Single Broadcast Channel in the Mobile Computing Environment. Multimedia Tools and Applications 9(1): 69-94 (1999)

3. Josep Daíz and Jordi Petit and María Serna: A Survey on Graph Layout Problems. *ACM* Computing Surveys, 34(3): 313-356 (2002)

4. Yon Dohn Chung, Myoung-Ho Kim: Effective Data Placement for Wireless Broadcast. Distributed and Parallel Databases 9(2): 133-150 (2001)

5. V.Gaede, O.Günther: Multidimensional access methods. ACM Computing Survey, 30(2):170-231 (1998)

6. A.Guttman, R-trees: A dynamic index structure for spatial searching. SIGMOD Conference, 1984:47-54

7. Qinglong Hu, Wang-Chien Lee, Dik Lun Lee: A Hybrid Index Technique for Power Efficient Data Broadcast. Distributed and Parallel Databases, 9(2): 151-177 (2001)

8. Qinglong Hu, Wang-Chien Lee, Dik Lun Lee: Indexing Techniques for Power Management in Multi-Attribute Data Broadcast. MONET 6(2): 185-197 (2001)

9. T. Imielinski, S. Viswanathan, B. R. Badrinath: Energy Efficient Indexing On Air. SIGMOD Conference, 1994:25-36

10. T. Imielinski, S. Viswanathan, B. Badrinath: Power Efficient Filtering of Data on Air. EDBT, 1994: 245-258

11. T. Imielinski, S. Viswanathan, B. R. Badrinath, Data on Air: Organization and Access. IEEE Transactions on Knowledge and Data Engineering, 9(3): 353-372 (1997)

12. Hans-Peter Kriegel: Performance Comparison of Index Structures for Multi-Key Retrieval. SIGMOD Conference 1984: 186-196

13. Guanling Lee, Shou-Chih Lo, Arbee L. P. Chen: Data Allocation on Wireless Broadcast Channels for Efficient Query Processing. IEEE Transactions on Computers 51(10): 1237-1252 (2002)

14. Wang-Chien Lee, Dik Lun Lee, Using Signature Techniques for Information Filtering in Wireless and Mobile Environments. Distributed and Parallel Databases, 4(3): 205-227 (1996)

15. Bernd-Uwe Pagel, Hans-Werner Six, Heinrich Toben, Peter Widmayer: Towards an Analysis of Range Query Performance in Spatial Data Structures. PODS 1993: 214-221

16. T. Sellis, N. Roussopoulos and C. Faloutsos. The R+-Tree: A Dynamic Index for Multi-Dimensional Objects. VLDB Journal, 1987:507-518

17. Kian-Lee Tan, Xu Yu, Generating broadcast programs that support range queries, TKDE 10(4):668-672:1998

18. Reuven Bar-Yehuda, Computing an optimal orientation of a balanced decomposition tree for linear arrangement problems. Journal of Graph Algorithms and Applications, 5(4): 1-27 (2001)

[HREF 1] http://www-users.cs.umn.edu/~karypis/metis/hmetis/

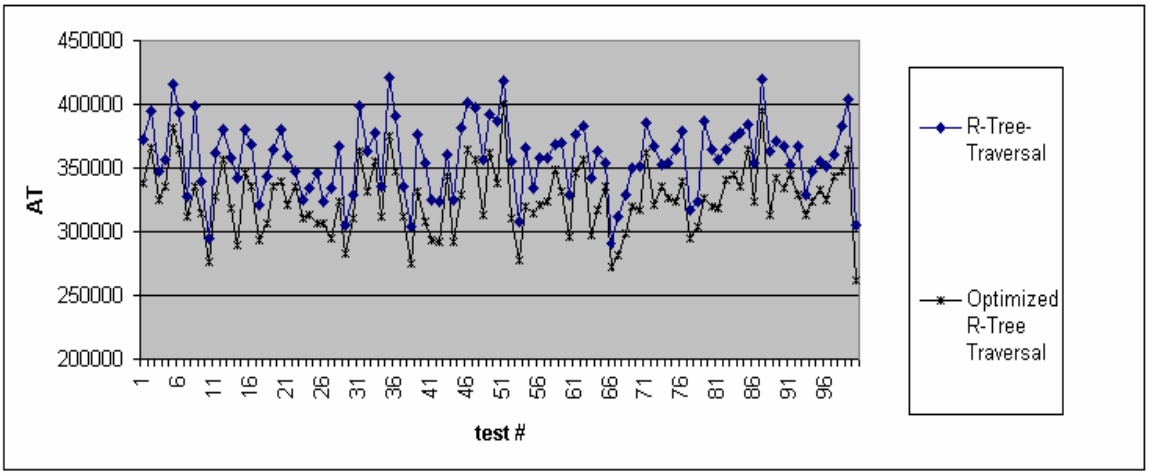


**Fig. 12. Comparison of Original and Optimized R-Tree Traversal Orderings**

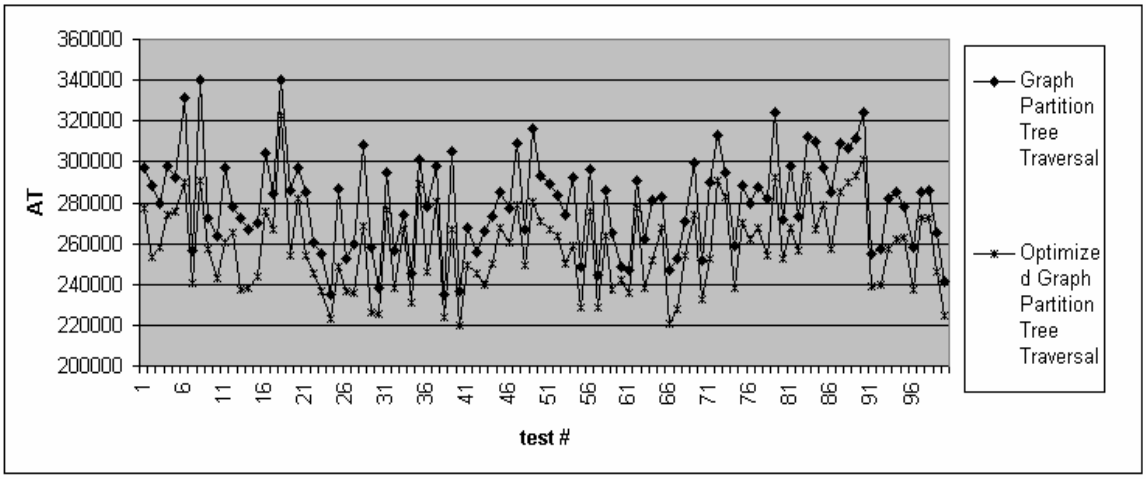


**Fig. 13. Comparison of Original and Optimized Graph Partition Tree Traversal Orderings**

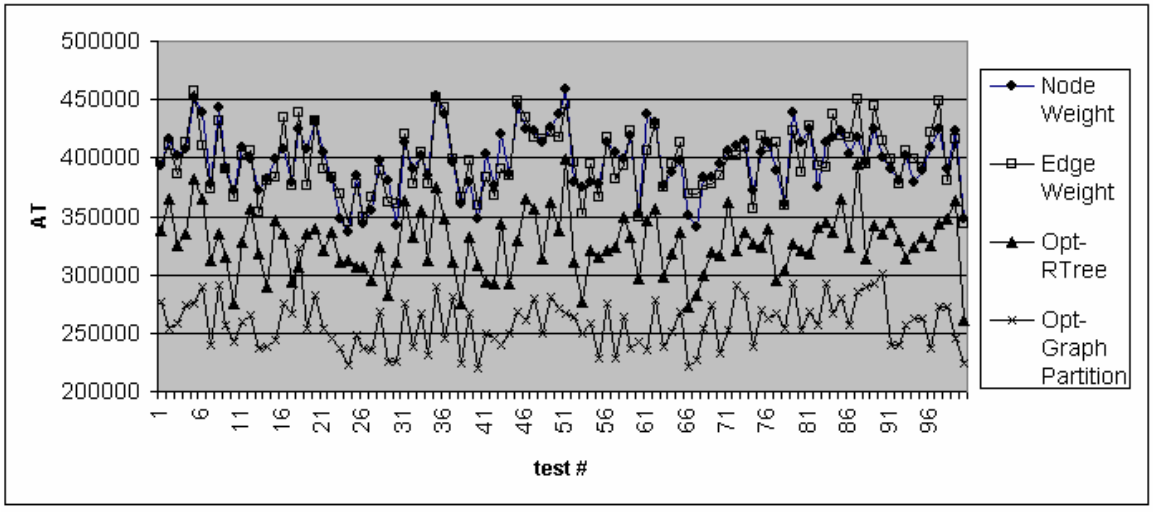


**Fig. 14. Comparisons of the Two Heuristic and the Two Optimized Orderings**