UNIVERSITY OF OKLAHOMA

GRADUATE COLLEGE

SEQUENCING GEOGRAPHICAL DATA FOR EFFICIENT QUERY PROCESSING ON AIR IN MOBILE COMPUTING

A Dissertation

SUBMITTED TO THE GRADUATE FACULTY

In partial fulfillment of the requirements for the

Degree of

Doctor of Philosophy

By

JIANTING ZHANG

Norman, Oklahoma

2004

SEUQENCING GEOGRAPHICAL DATA FOR EFFICIENT QUERY PROCESSING ON AIR IN MOBILE COMPUTING

A Dissertation APPROVED FOR THE SCHOOL OF COMPUTER SCIENCE

By

Dr. Le Gruenwald (Chair)

Dr. Mohammed Atiquzzaman

Dr. Changwook Kim

Dr. Pakize Pulat

Dr. Krishnaiya Thulasiraman

Acknowledgment

My deepest appreciation goes to all who have helped me during my entire Ph.D. study. My special thanks to the committee members, Dr. Le Gruenwald, Dr. Mohammed Atiquzzaman, Dr. Chang-wook Kim, Dr. Pakize Pulat, and Dr. Krishnaiya Thulasiraman. I appreciate their time, advice and patience as I have gone through this process.

I would like to express my sincere gratitude to my committee chair, Dr. Le Gruenwald. She inspired me to begin my career in computer science four years ago and has been guiding me ever since. Her guidance and supports, both mentally and financially, are essential to the completion of this dissertation.

I acknowledge the generous financial supports from Dr. V. Lakshmanan at the Cooperative Institute for Mesoscale Meteorological Studies (CIMMS), who has provided me with a research assistantship through his NSF grant. The flexible RA work schedule has helped me greatly in my dissertation research, conference travels as well as overcoming family hardships.

I also would like to acknowledge the support from my wife, Guomei Cao. I am indebted to her for the numerous nights that I stayed in my office while leaving her alone at home during her pregnancy and taking care of our baby thereafter. My one-year-old son, John Zhongyao Zhang, has provided me with a much needed refuge from the stress.

Finally I am indebted to my parents, Shijiang Zhang and Deqin Miao, for their enthusiastic encouragement and unconditional support. Although they have suffered from some significant medical problems, they have never stopped encouraging me to complete my studies. May this dissertation be dedicated to them.

Table of Contents

List of Figures

List of Tables

Abstract

Geographical data broadcasting is suitable for many large scale dissemination-based applications due to its independence of number of users, and thus it can serve as an important part of intelligent information infrastructures for modern cities. In broadcast systems, query response time is greatly affected by the order in which data items are being broadcast. However, existing broadcast ordering techniques are not suitable for geographical data because of the multi-dimension and rich semantics of geographical data. This research develops cost models and methods for placing geographical data items in a broadcast channel based on their spatial semantics to reduce response time and energy consumption for processing spatial queries on point data and graph data.

Three cost models are derived to measure Data Broadcast Wait (DBW), Data Access Time in the multiplexing scheme (AT_{Data}^{Mul}) where both data and indices are broadcast in the same channel, and Data Access Time in the separate channel scheme (AT_{Data}^{Sep}) where data and indices are broadcast in two separate channels. Hyper-graph representations are used to represent the spatial relationships of both point data and graph data. The broadcast data placement problem is then converted to the graph layout problem. A framework for classifying ordering heuristics for different types of geographical data is presented. A low-polynomial cost approximation graph layout method is used to solve the DBW minimization problem. Based on the proven monotonic relationship between AT_{Data}^{Sep} and DBW, the same approximation method is also used for AT_{Data}^{Sep} optimization. A novel method is developed to optimize

AT_{Data}^{Mul}. Experiments using both synthetic and real data are conducted to evaluate the performance of the ordering heuristics and optimization methods. The results show that R-Tree traversal ordering heuristic in conjunction with the optimization methods is effective for sequencing point data for spatial range query processing, while graph partition tree traversal ordering heuristic in conjunction with the optimization methods is suitable for sequencing graph data for network path query processing over air.

Chapter 1

Introduction

Analog radio broadcast has played important roles in modern society during the past decades. The last decade saw great expansions and interconnections of digital information, the World Wide Web for example. While the client/server architecture of the Web and the underlining point-to-point communication infrastructure of the Internet work fine for moderate traffic, they do not scale well when millions of people request similar information from a website. The problem is even severe as more and more information systems are extending to wireless and mobile networks to allow information access anytime and anywhere. Due to the limited nature of wireless bandwidth, scalability in such large systems is very likely to be a big issue.

Broadcast is suitable for dissemination-based applications with the following characteristics (Aksoy, 1998): large scale, high overlapped demands among users and the asymmetric data flow from sources to users. Broadcast is a promising alternative to point-to-point access in many cases since resource consumption in a broadcast system is independent of the number of users in the system. Geographical information has been widely used in our everyday lives. Geographical information broadcasting can serve as an important component of intelligent information infrastructures for modern cities.

Due to the sequential nature of a data broadcast system, query processing over air medium is significantly different from that in a disk or main memory resident

database system. The ordering of a broadcast sequence plays an important role in the query performance. However, existing broadcast ordering techniques are not suitable for geographical data because of the multi-dimensional and rich semantics characteristics of geographical data. *The objectives of this study are to provide cost models and techniques for ordering geographical data in broadcast channels that improve spatial query processing on air.*

In this chapter, we first introduce some background on data broadcast, geographical information and geographical information broadcast. We then discuss some application areas and point out the research challenges concerning geographical information broadcasting. Finally we state our research objectives and present the dissertation outline.

1.1 Data Broadcast

Data broadcast can be performed on either wired or wireless network using either a single-hop or a multi-hop communication infrastructure. An excellent example of single-hop data broadcast is the Datacycle project at Bellcore more than 15 years ago where a database circulates on a high bandwidth optical network (140 Mbps) (Herman, 1987). From the application perspective, the current Internet multicast can be treated as multi-hop broadcast to a user group on fixed networks. Disseminating data from a node to all the other nodes in a wireless sensor network is a good example of multi-hop broadcast on wireless network. Multi-hop broadcast is more energy-efficient than single-hop broadcast since the received signal power decreased much faster than the communication distance ($p'=p*r^{-\alpha}$, where p is the

transmission power, *p'* is the received power, *r* is the distance and α is a parameter typically between two and four) (Wieselthier, 2002). However, when there are special nodes in wireless networks that are free from energy constraints, it is advantageous to use single-hop broadcast as discussed shortly.

In this study we are interested in geographical data broadcast to support location dependent services. We adopt single-hop wireless data broadcast for several practical reasons. First, cellular networks, the most popular form of wireless mobile communication at present, use wireless broadcast at their last hop where the base stations are the special nodes that are generally thought to be free from energy constraints. It is beneficial to utilize cellular networks by setting broadcast servers at the base stations. Second, even in wireless ad-hoc networks, it is very likely that there are some mobile units have more power supplies and computing powers than others. It is beneficial to tradeoff energy consumption with coverage and mobility management overheads. For the rest of this dissertation, we refer "single-hop digital wireless data broadcast" as "broadcast" or "data broadcast".

Data broadcast can be classified into two main categories, pull-based and push-based (Aksoy, 1998). In pull-based broadcast, the broadcast server receives explicit requests from clients and schedules a broadcast sequence based on the requests. In this case there are no unwanted data in the broadcast sequence which can improve channel utilization. In push-based broadcast, the data access patterns are assumed to be fixed and the broadcast sequence is pre-determined. It is possible that there are data items in the broadcast channel that are not needed by any clients at

particular time slots. Although the broadcast channel might not be fully utilized in push-based broadcast, it has two advantages. The first is that it does not need on-demand scheduling which could be very expensive. The second is that no up-link communication between clients and the broadcast server is needed which makes it suitable for light-equipped and inexpensive handsets.

In addition to the excellent scalability as discussed earlier, there are several additional advantages for single-hop wireless and push-based broadcast. First, data communication through broadcast consumes less energy since users are in receiving mode instead of sending mode. Second, there is no mobility management problem for the broadcast server when users are in the receiving range of the server while there are significant overheads in mobility management in cellular or ad-hoc mobile networks. Third, since handsets in such broadcasts systems do not need up-link communication components to send data, their sizes/weights and manufacturing cost can be significantly reduced. The reduction of sizes and/or weights can further reduce power consumption.

Compared with analog radio broadcast, digital broadcast allows automatic data filtering and integration of multiple resources to provide targeted and personalized data without having to physically tuning to radios. Digital broadcast of newspapers to individual subscribers can be traced back as early as 1985 when personal computers are still not powerful enough to accommodate several Kbps data transfer rate (Gifford, 1985). Several standards have been proposed for digital broadcast, such as the ATSC data broadcast in North America (Chernock, 2001),

digital audio broadcast (Hoeg, 2001) and digital video broadcast (Reimers, 2001) standards in Europe. However, such techniques are mostly designed for streamed multimedia broadcast and do not support interactive queries over broadcast data. It is worth to mention that these multimedia broadcast standards are not specially designed for wireless broadcast. Actually they are currently more suitable to apply to cable networks. Although multimedia broadcast and database broadcast can share the same broadcast techniques at the physical level for broadcasting data bits, unlike audio/video broadcast which has a predefined order based on time sequence, orderings of the data items (and their indices as well) in database broadcast will affect the performance of query processing significantly.

The digital audio broadcast standard (Reimers, 2001) has defined data services and applications which allow broadcasting data other than audio and video, such as "Broadcast Web Site" (TS 101 498). Although the standard suggests prioritizing data objects based on their individual access frequencies similar to our preliminary work in (Zhang, 2002), it does not take the case in which multiple data items are accessed together into consideration. Further discussions on this problem will be provided in Section 1.5 and Section 2.1 in Chapter 2.

1.2 Geographical Information

Geographical information has been widely used in our everyday lives. It has been used in applications such as finding service locations (e.g. restaurants and ATM machines) and getting traffic and travel information. The National Academy of Sciences estimates that 80 percents of the information on the Internet have a spatial

component ([HREF 1]). The importance of geographical information has been recognized in mobile computing in the context of location management in cellular and ad-hoc networks (Wong 2000), position-based routing protocols (Mauve, 2001) and location based services (Virrantaus, 2001), etc.

Geographical Information Systems (GIS) have been used for geographical data management. In the database community, research on geographical data falls into the category of spatial databases (Rigaux, 2002; Shekhar, 2003). Geographical data types, such as point, polyline and polygon, are often modeled as objects, thus research on geographical data management is also related to object-oriented databases. ORACLE versions 8 through 10 define various geographical data types and use its object-relational data model to manage geographical data ([HREF 2]). Oracle version 9 and higher support spatial window (range), spatial join, nearest neighbor and other spatial queries ([HREF 2]).

Almost all the existing research on geographical data management assumes the underlining access medium is disk and much effort has been put on reducing I/Os. We envision that non-disk based spatial databases will attract more and more research interests in the areas such as main-memory spatial databases and spatial databases over air. Broadcasting spatial databases over air allows an unlimited number of users to access the spatial databases simultaneously using simple and cheap receiver any time and anywhere.

1.3 Geographical Information Broadcast

Geographical data are especially suitable for broadcasting. It serves a great number of users, such as users in metropolitan areas. It is public and has no or very few privacy concerns. It is mostly read-only and changes relatively slowly. Most importantly, it is distributed in nature which can eliminate the biggest disadvantage of broadcast, i.e., limited broadcast range. This is because most of geographical data accesses are local, i.e., people are more likely to access the geographical data that are near to them. We can adopt the cellular structure and distribute geographical data to the base stations for distributed broadcast. Fig. 1-1 illustrates the idea of geographical information broadcast for mobile computing at different levels of wireless networks. Geographical data at a global scale can be broadcast over satellite channels, while those at the country or state scales can be distributed to local broadcast servers through wired or wireless Wide Area Network (WAN) and those at the local scales (such as urban areas, communities or buildings) can use base stations in cellular networks as broadcast servers.

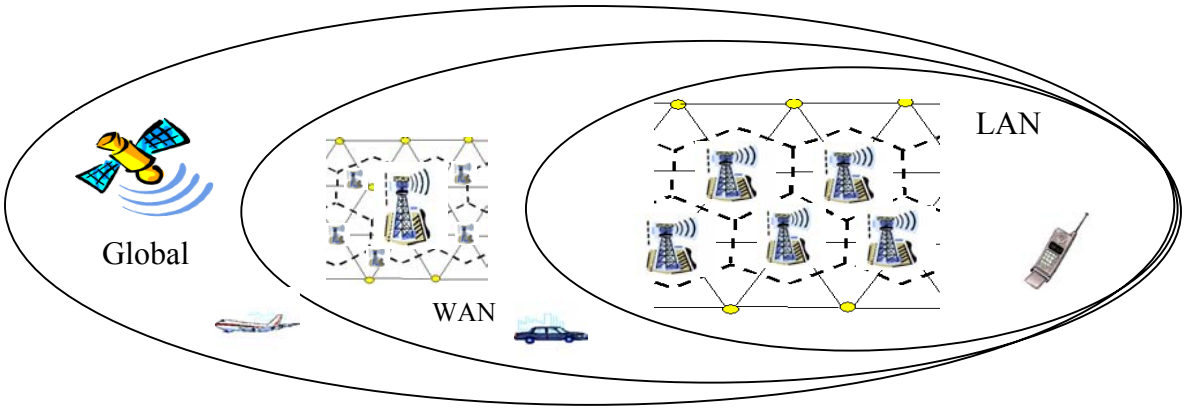


Fig. 1-1. Geographical Data Broadcasting for Mobile Computing

We are particularly interested in push-based geographical data broadcast since the expected number of users in our applications is very big and it is too expensive to schedule a broadcast as that done in pull-based broadcast. For example, there could be millions of people who request traffic data at the same time in peak traffic time in metropolitan areas. The capability of allowing inexpensive mobile handsets to perform spatial queries over broadcast geographical data is a plus for push-based broadcast.

1.4 Possible Application Areas

We envision that geographical data broadcast over air has a broad scope of application areas, ranging from location dependent services in metropolitan areas, unusual event warnings in remote areas, disaster rescuing and military related applications.

A. Location Dependent Services

There are several ways for users to be aware of their locations. The Global Position System (GPS) provides very accurate position information. An inexpensive hand-held GPS receiver can provide an accuracy of 10 meters or better (Leonhardi, 2002). The infrastructures of most cellular networks can at least tell which cell a mobile user is currently in; this is a part of location/mobility management in the networks (Wong 2000). With the help of the neighboring base stations, the networks have the capability to tell the users their positions more accurately. In many cases, the position information provided by GPS, network infrastructures or their combinations (Konig-Ries, 2002) are accurate enough to perform Location

Dependent Queries (LDQ) and request Location Dependent Services (LDS) (Seydim, 2001). Two examples of such queries are "find all the ATM machines within 2 miles of my current location" and "tell me the shortest path from the White House to University of Maryland campus". These services can be very useful for users in unfamiliar places. Furthermore, intelligent navigation systems can be built on top of LDS over broadcast geographical data, such as shopping guidance in big malls, transferring flights in busy airports, finding books in a library and locating rooms in skyscrapers. By issuing LDQs continuously over broadcast geographical data, the users' intelligent agents will lead the users to their destinations. Comparing with using point-to-point communication for such services, all the advantages of data broadcast we discussed before apply.

B. Unusual Event Monitoring

Unusual events, such as traffic jams, storms and hurricanes, affect our everyday lives greatly. Some of them are matters of life and death. A public warning system is extremely useful in these situations. Traffic jams and road accidents have been broadcasting in analog form during the past decades and are going to be broadcast digitally ([HREF 3]). A new industry called Telemetrics that explore digital data broadcast technologies is coming into being (Xu, 2000). Energy consumption in those applications is usually not a problem since such events happen infrequently and users usually have continuous power supply, such as in cars. The reason of using data broadcast technologies from the sender's perspective is primarily for its scalability and wide coverage. From the receiver's perspective, it is crucial to reduce query

response time for queries that inquire whether there are or there are no such unusual events within a spatial range of some specific locations. This is especially important for the events that are broadcast through satellites to wide regions in remote areas. Since the number of such events is large while the available satellite bandwidths are limited, the broadcast cycle can be long and it is crucial to reduce response time by careful data placement.

C. Disaster Rescue

The power supply of a handset is usually very limited when a disaster happens. If the disaster happens far away from base stations, in a dessert for example, it is quite possible that the handset power might be quickly depleted after several unsuccessful connections. An alternative way might be to broadcast the geographical information and other related information in the disaster area. By using such information, people that are trapped by the disasters might be able to make right decisions. Power consumption is the primarily concern in such cases.

D. Military Operations

Communications in battlefield are crucial. One of the advantages of data broadcast in battlefield is safety. Since a soldier does not interact with the server by only listening to broadcast geographical and other types of data, he/she cannot be detected based on signal his/her handset emits. Data broadcasting is also advantageous when a soldier is isolated and has very limited power left and cannot afford active communication. Geographical data broadcast can also be used for group dispatch or guidance. For example, a group of soldiers in a particular region should

move to another region or follow a particular route. A broadcast server can also broadcast road networks and topography in a particular area, updated information to data stored on the CD or other medium that go with soldiers, etc.

1.5 Research Challenges

Most existing geographical information systems are disk-resident. Spatial indexing and query processing techniques are mostly designed for reducing the number of I/Os. However a broadcast channel as an access medium is essentially one-dimensional and only allows sequential access which is quite different from disk or main memory based data access. The difference between disk-resident data access and broadcast channel data access is illustrated in Fig. 1-2. In disk resident data access, the read/write arm first moves the read/write head to the desired disk track, and the disk then rotates to the desired sector. Although the sequence of data items still plays an important role in performance as explained in Chapter 2, disk resident data access as well as main-memory data access can be generally treated as random. In broadcast data access, although only some data items (including both index and data) are needed (those that are shaded in Fig. 1-2), a client will have to wait between two needed data items (those that are non-shaded in Fig. 1-2). More detailed explanations for broadcast channel based data access are given in Section 3.1.

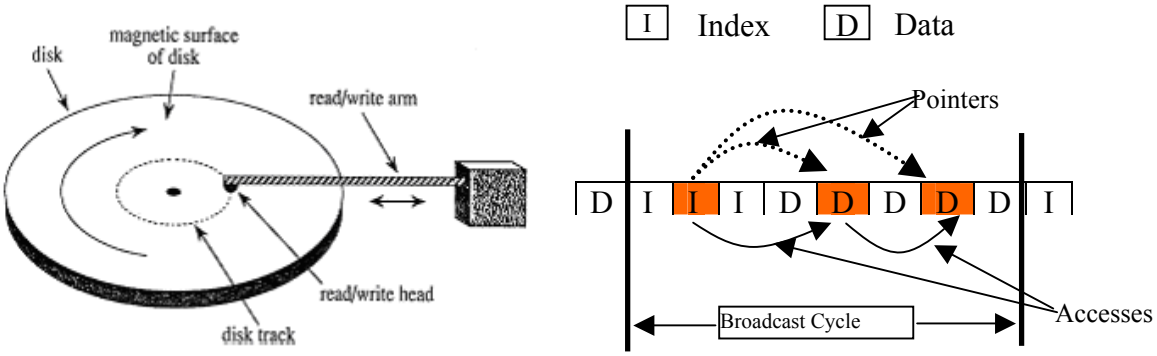


Fig. 1-2. Disk Based (The Left Figure) And Broadcast Channel Based (The Right Figure) Data Access

Geographical data is multi-dimensional spatial data that has rich semantics which renders existing broadcasting techniques not suitable for its broadcasting. In this study we mostly target the first and the second application scenarios discussed above, i.e., location dependent query and unusual events monitoring. We are interested in two major geographical data types that are widely used in mobile computing, i.e., point data and graph data. Point data has explicit geometric coordinates and the spatial semantics among them are implicit. For graph data, the spatial semantics are explicitly expressed in terms of the weights of edges between the nodes of a graph. In this study, we assume graph data are two-dimensional geometric network and thus their vertices are also points. A typical application scenario of point data broadcast is a spatial range query that retrieves all the gas stations within 2 miles of a user's current location over a broadcast channel. A typical graph data broadcast scenario is a network path query that finds the shortest path from location A to location B over a broadcast channel. In these queries, there may be more than one data items (restaurants or locations) in the query results. We use the

term “Complex Query” (Lee, 2002a) to denote the queries whose result sets have multiple data items.

Query response time is greatly affected by the order in which geographical data items are being broadcast. Suppose there are six data items {1,2,3,4,5,6} to broadcast and there are two data items {2,5} in a spatial query result set. It only takes two units of time to retrieve the query result if the data items 2 and 5 are placed next to each other. However, it would take four units of time to retrieve them in the natural ordering. The placement is complicated when there are many such complex queries with different access frequencies over broadcast data.

1.6 Research Objectives and Dissertation Outline

Using air as an access medium for geographical data broadcast, or spatial databases on air, requires a new scheme for data organization and query processing. T**he objectives of this study are to develop cost models and methods for placing geographical data items onto a broadcast channel based on their spatial semantics to reduce the response time and energy consumption for processing spatial queries over broadcast channels.** In order to achieve the objectives, this dissertation performs the following tasks:

- Derive the cost models of computing the data access time for processing spatial queries over broadcast geographical data under different scenarios.
- Provide hypergraph representations for spatial relationships of both point data sets and graph data sets and relate the broadcast data placement problem with graph layout problems.

- Present a coherent framework for classifying ordering heuristics and discuss their applicability for different types of geographical data.
- Develop efficient and effective optimization methods to reduce data access time under different cost models.
- Perform experiments on both ordering heuristics and the optimization methods using both synthetic and real data sets.

This dissertation is outlined as in Fig. 1-3 where arrows show the dependencies between chapters. We first review the related work in Chapter 2. We then present our three cost models for spatial range queries and network path queries under two different scenarios in Chapter 3. We propose to use a hypergraph to represent the spatial semantics of a data set in our applications in Chapter 4. In Chapter 5, we discuss several heuristics to generate the orderings of broadcast sequences for both point data and graph data. The orderings based on the heuristics can be used as initial orderings for optimization. We provide several methods to solve the optimization problems efficiently in Chapter 6 under different scenarios. Chapter 7 presents experiments on the heuristics and optimization methods based on our cost models using real and synthetic data.

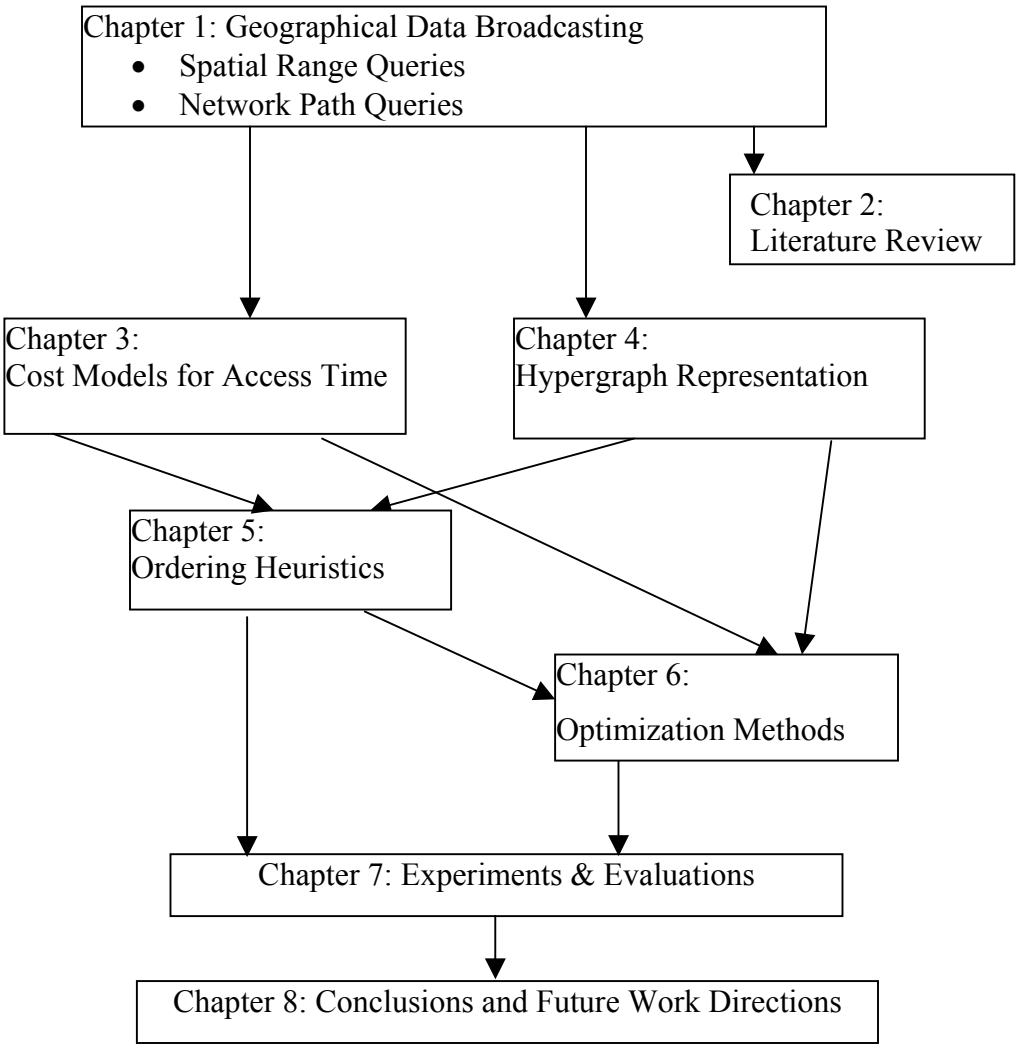


Fig. 1-3. Dissertation Outline

Chapter 2

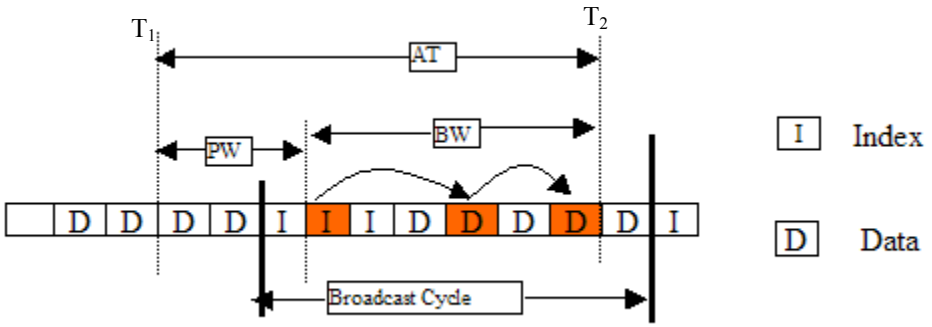
Literature Review

In this chapter, we first review the existing work on generic data broadcast and we then turn to spatial indexing and spatial query processing in disk-resident spatial databases. Although extensive work has been done in both fields, we believe we are the first to address the problem of sequencing geographical data items on broadcast channels for efficient spatial query processing. As shown in Chapter 3, we transform the sequencing problem to a graph layout problem based on spatial semantics. Thus our work is also related to the page ordering research in disk-resident database systems and graph layout problems from a theoretical perspective. Finally, we provide brief reviews of other related work, such as graph partition, spatial clustering, location-based services and moving object data management. It is our vision that geographical data broadcasting can be used for location-based services as explained in the introduction chapter. We also envision that moving object queries over air is a very promising alternative to current disk-resident databases on top of point-to-point communication infrastructures. Graph partition and spatial clustering are used for generating heuristic orderings and optimizations as discussed in Chapter 5 and Chapter 6.

2.1 Data Broadcast

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) are called a segment. The time to broadcast a segment can be calculated as the volume of a data segment divided by the bandwidth allocated for broadcasting. For the sake of simplicity, it is generally assumed that all the broadcast data items have the same volume and it takes a unit time to broadcast a data item. Based on this assumption we can measure the broadcast parameters discussed below using the positions of the data items in a broadcast sequence and their intervals (lengths or durations) as the measurements of access time.

Different from disk resident data accesses, accesses to broadcast sequence are essentially one-dimensional. There are two important parameters in evaluating the performance of a broadcast system, namely Tune-in Time (TT) and Access Time (AT, or latency). TT is the amount of time spent by a client listening to the channel. AT is the average time elapsed from the time a client requests data to the time when all the required data are downloaded by the client. 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 nearest index segment and the later is the average duration between the time the index segment is reached till all the required data items are downloaded. In Fig. 2-1, TT is equal to the summation of the lengths of the required data items (shaded) while AT is the duration between the initial access time and the time to access the last required data item.



T_1: Time to begin accessing the broadcast channel
T_2: Time when all requested data items are downloaded

Shaded boxes: required data items
Non-Shaded boxes: not required data items

Fig. 2-1. Illustration of TT and AT

There is a well-known tradeoff between AT and TT based on the assumption that the energy consumption in TT is far greater than that in AT (250mW versus 50μW for AT&T Hobbit chip, Imielinski, 1997). Two extreme strategies exist for data broadcasting (Imielinski, 1997). For the *Access Opt* strategy, there will be no index and only data is broadcast to minimize AT. While for the *Tune Opt* strategy, extensive indexing is used to minimize unnecessary active channel listening (TT). Strategies that combine the two are often adopted. Although TT is determined by the number of data items in a query result and the number of index segments that need to be accessed, AT is determined by the length of a broadcast cycle and the ordering of data items in a broadcast sequence. The purpose to reduce TT is primarily for energy consumption while reducing AT will reduce both energy consumption and query response time.

The idea of using air as a data access medium is first proposed in (Imielinski, 1993). (Imielinski, 1997) and its related work (Imielinski, 1994a; Imielinski, 1994b) presented both a framework and several indexing methods for data broadcasting over the air. However, their work only takes one-dimensional tree indexing (B-Tree) into consideration. For the multiple-attribute case, they proposed to build multiple indices for each fragment of the first attribute. This is actually using one-dimensional indexing methods consecutively for multidimensional indexing which is inefficient (Kriegel, 1984). Using signature techniques for information filtering in wireless and mobile environments was presented in (Lee, 1996). A hybrid method by combining tree indexing and signature methods was proposed with demonstrated advantages (Hu, 2001a). However, signature based index methods only work for categorical data. They cannot be used for geographical data that is multidimensional and continuous in nature. The issue of multi-attribute data broadcast and query was explicitly addressed in (Hu, 2001b). However, it can only handle conjunction/disjunction queries that involve fewer than three attributes. Except for (Hu, 2001b), none of them considers different access frequencies of data items. Their focus is to trade TT with AT by using indices and reduce PW by index replication.

The Broadcast Disk technique (Achrya, 1995) was proposed to broadcast data over multiple channels to reduce total access time by allocating bandwidth to disks (channels) based on the access frequencies of the data items placed in the channels. (Shivakuma, 1996) proposed an alphabetic Huffman tree based scheme to broadcast the index over multiple channels. In this scheme, the number of channels has to be

equal to the levels of the constructed alphabetic Huffman tree. In (Peng, 2000), a heuristic algorithm VFK was proposed to assign data items to multiple data channels based on the constructed imbalanced index tree which was again based on access frequencies. In this scheme, the number of channels can be arbitrary. Recently (Hsu, 2002) extended the distributed indexing technique proposed in (Imielinski, 1997) to multiple channels. It first assigned Broadcast Segments (BS) to channels in the decreasing order of the summation of access frequencies of data items in the BSs. For the BSs in the same channel, it replicated the BSs based again on the summation of frequencies and distributed the replicated BSs on the channel as evenly as possible. This technique was applied again for the data items within each BS based on their access frequencies. Considering data access frequencies will generally improve the average access time. However, all these techniques only consider retrieving a single data item from broadcast channels. It is obviously inefficient to retrieve multiple data items in a query result set by applying these methods multiple times.

To the best of our knowledge, none of the above studies addresses the ordering problem of data items to be broadcast to reduce access time incurred in accessing multiple data items in a query result set. Although (Imielinski, 1997) proposed to chain data items that have the same values in different meta-segments in its nonclustering index and multi-index methods, it cannot be applied to data items that have different values but are often in the same query result (such as spatial range queries and network path queries). Furthermore, in the performance analysis, their work assumed that it would take a whole broadcast cycle to retrieve non-clustered

data items of a particular value which is an overestimation. As we illustrate in Chapter 7, we can improve query response time and clients' energy consumption by carefully ordering data items in the broadcast channel before they are broadcast.

We believe that (Gondhalekar, 1997) was the first to address the problem of retrieving multiple data items from wireless broadcast channels. They showed that the problem is in general NP-complete for both a non-indexed data layout and indexed data layout with unit access frequency. They related the non-indexed data layout to the Optimal Linear Arrangement problem in graph theory. They also provided two heuristics for optimizing the broadcast sequence for the indexed layout. Their cost model for the non-indexed data layout is essentially our DBW and their cost model for indexed data layout is essentially our DPW+DBW under the multiplexing scheme as presented in Chapter 3. We will further discuss the two heuristics in Chapter 5.

The recent works on object-oriented database broadcast (Chehadeh, 1999) and relational database broadcast (Si, 1999; Lee, 2002; Lee 2003) allow multiple data items to be accessed in a query. The works presented in (Chehadeh, 1999; Lee, 2002) are only applicable to the cases where initial access must be done at the beginning of a broadcast cycle. They assumed the access to data items has a predefined order and thus are not suitable for spatial range queries since data items in a query result do not necessarily have a predefined order. Furthermore, the ordering heuristics proposed in the papers are greedy. The graph representation in (Si, 1999) allows each entity type to be accessed as the first one with a certain probability. The optimal ordering is obtained through the branch-and-bound technique. However, it might be too

expensive when the number of entity types is large since the complexity of the branch-and-bound technique is exponential with respect to the number of items to be sequenced in the worst case. In (Lee, 2003), a regular graph called data access graph is first constructed. The weight of an edge (u,v) is the summation of the frequencies of all queries that include both data items u and v. It then extended the iterative node combination process for sequencing tree-structured data proposed in (Chehadeh, 1999) to sequence the data access graph. The ordered binary combination processes determine the broadcast sequence. We provide further discussions on the graph representation of both (Si, 1999) and (Lee, 2003) in Chapter 4 and the scheduling algorithm of (Lee, 2003) in Chapter 5.

The work presented in (Chung, 2001) is most similar to our cost model under the broadcast scheme that data and index are broadcast in different channels. Its QEM scheduling method and its extensions in (Lee, 2003) are essentially greedy, which is quite different from the optimization philosophy we adopt in this study. We recently found that the proof of the relationship between the average AT presented in (Chung, 2001) and the concept of Query Distance (QD) on which the methods in (Chung, 2001; Lee, 2003) are based is incorrect. The first author of (Chung, 2001) has recognized our finding (email communication, 2003). We discuss the cost model of (Chung, 2001) in Chapter 3 and the QEM-alike scheduling algorithms (Chung, 2001; Lee, 2003) in Chapter 5.

(Tan, 1998) presented an algorithm (NrBP) to generate broadcast sequences that facilitate range queries without sacrificing much on the advantages of

nonuniform broadcast for single data item retrieval. The basic idea is to range partition the data file into disjoint key ranges. Their analytical results showed that the uniform broadcast program (UBP) is the best for range access while the classic nonuniform broadcast program (NBP) is best for single data item retrieval which is based on the broadcast disk technique. They concluded that NrBP was a better choice for both single data item access and range access. However, the proposed algorithm is only applicable to one-dimensional data. An additional problem with their analytical study on the average access time for accessing items in a given range is as follows. Suppose there are n data items to broadcast and i is the ith data item, they assumed that the range a user can specify varies from 1 to n-i and any range between 1 to n-i are equally accessed. While the assumption might be reasonable if the data items are uniformly distributed, usually it does not hold for arbitrarily distributed data such as the distribution of service locations.

To the best of our knowledge, the only previous work on geographical data broadcast is (Hambrusch, 2001). It studied the execution of spatial queries on broadcast tree-based spatial index structures. It assumed that the client has such a limited memory that the whole R-tree cannot fit into the client memory and the client has to discard some retrieved R-Tree nodes to hold more useful ones during the query process. Their work focused on reducing extra access time incurred by having to access multiple broadcast cycles due to the replacement of the R-tree nodes in the client's memory. Our work differs from theirs in that we aim to generate a good sequence of geographical data items for spatial queries after we have already had

indices, either from a separate index channel or from the same channel that broadcasts both data and indices, and a client only needs to access one broadcast cycle at most to retrieve desired data items. Since memory is getting cheaper and cheaper, we assume that the client memory can hold the entire index segments related to a query and we believe this assumption is more practical.

Using multiple broadcast channels for data (not including indices) that supports complex queries was addressed in (Huang, 2003). This work proposed genetic algorithms for broadcast sequencing. Initially random sequences are generated for each query and their fitness (defined as the average access time) are computed. In the selection phase the selection probability of each sequence is set to be the weights of their fitness. In the crossover phase, the Partially Mapped Crossover (PMX) is performed and new sequences are generated. The best sequence is then selected according to the fitness. The process is repeated for a predefined number, i.e., the number of generations for genetic evolution. The philosophy of their method is quite different from that of graph-theoretic approaches.

2.2 Spatial Indexing and Query Processing

Traversal of spatial indexing trees and Space Filling Curves (SFC) generate orderings of data items. Structures of spatial indexing trees reflect spatial relationships between data items and can be used to produce better orderings.

A comprehensive overview of multidimensional indexing and access methods was presented in (Gaede, 1998). The original R-Tree method was proposed in

(Guttman, 1984). Two R-Tree variations, namely R+-Tree (Sellis, 1987) and R*-Tree (Beckmann, 1990) were introduced later. SFC, such as row-wise enumeration of the cells (Samet 1990), Peano curve (Morton, 1966) or Z-Ordering (Orenstin, 1984), Hilbert-Ordering (Faloutsos, 1989; Jagadish 1990) and Gray-Ordering (Faloustsos 1988) could be used to transform multi-dimensional data into one-dimensional data, and consequently, one-dimensional index techniques such as B-Tree can be used for spatial indexing. Several characteristics such as jumps, continuity, reverse order and bias towards a particular dimension were studied theoretically and experimentally in (Aref, 2000). The mean-variance analysis of the performance of spatial ordering methods was studied in (Kumar, 1998) for disk-resident spatial data. Since they used a block factor of 1 (i.e., there is only one data item in a block) and the measurement is the number of clusters (a cluster is a group of consecutive disk blocks according to a SFC ordering) of a range query result set in a SFC ordering, their studies can also be interpreted as the measurement on the number of mode switches (between active and doze/sleep) in a spatial range query over a broadcast sequence.

Since we assume the geographical data items to be broadcast are known prior to broadcast, they can be treated as static data. In disk-resident database systems, several methods were proposed for building indices for static spatial data, such as NEXT-X (Roussopoulos, 1985), STR (Sort-Tile-Recursive, Leutenegger, 1997), TGS (Top-down Greedy Splitting Algorithm, García, 1998), Small-Tree-Large-Tree Approach (Chen 1998; Chen 2002) and its generalization (Choubey, 1999). Hilbert

R-Tree which is a R-Tree based on Hilbert-Ordering was proposed in (Kamel, 1994).

Several proposed cost models (Kamel, 1993; Pagel, 1993; Theodoridis, 1996; Theodoridis, 2000) for R-Trees use the number of disk accesses as the performance measurement. These models are based on the observation that the number of node accesses to a R-Tree is proportional to the summation of the extended node sizes of all the nodes in the R-Tree, assuming the data sets are uniformly distributed. Suppose the MBR (Minimum Bounding Rectangle) of a two-dimensional R-Tree node is (N_x, N_y) and the query window size is (Q_x, Q_y), then the extended node size is defined as $(N_x+Q_x)*(N_y+Q_y)$. The buffer effect was further considered in (Theodoridis, 2000) where if an R-Tree node is already in the buffer then no upload from disks is needed. More recently, (An, 2003) proposed to use a density file data structure for more accurate spatial range query cost estimations. They claimed that their method did not make any assumptions about the data set. In our hypergraph representation of spatial range queries on point data sets (Chapter 4), we compute the weights of all possible spatial range query result sets, thus the applicability of our method is also data set independent. Although storing and manipulating hypergraphs have larger overhead than the density file structure, the overhead is well justified since the response time and energy consumption for air access is much more expensive than disk I/Os considering a high number of users.

The Compact Path Encoding structure was first proposed in (Agrawal, 1989) and later extended to a hierarchical scheme for path query in transportation networks

by (Jing, 1998). (Shekhar, 1997a) presented a connectivity-clustered access method for disk-based networks and network computations. It proposed using edge access frequencies to partition a network into smaller ones for efficient data accesses. (Zhao, 2001) proposed a different method by indexing segmented graph data rather than partitioning the graph data for indexing. Materialization tradeoffs (storing pre-computed paths versus on-demand computation) in hierarchical shortest path algorithms using graph partition was addressed in (Shekhar, 1997b). Finding the shortest path in large networks based on hierarchical graphs was proposed in (Chan, 2001). All these works on graph data query processing assume main memory or disk as the data access medium while our interest is on graph data access on the air medium.

2.3 Page Ordering and Graph Layout

The ordering problem of data items has been addressed in disk-resident database systems although in a different context than broadcast sequencing. To store two or more dimensional spatial data in disk-resident databases and process spatial queries efficiently, it is important to take advantage of the characteristics of disk management where a disk page is used as a basic unit which might contain multiple data items. The basic idea is to store semantically related data items in the same pages so that only a minimum number of I/Os is necessary even though multiple data items might be requested in a query. The order of data items will determine whether two data items can be put into the same page, and thus, it has a great impact on query

processing performance. However, the order of data items within a page is insignificant in terms of the number of I/Os. This is quite different from broadcast channel based data access where the position of each data item matters as formalized in Chapter 3. Almost all tree-based spatial indexing methods exploit spatial adjacency in forming node split/merge policies. Traversing a spatial index tree or a hierarchical clustering tree gives an order of spatial data items. SFCs generate orders of multi-dimensional data items. More details are discussed in Chapter 5.

(Bachmann, 1994) discussed the problem of assigning spatial data items to buckets (pages) to minimize access cost based on their region (range) query cost model. By converting the problem to a Minimum Weighted Matching Problem in a graph, an $O(n^3)$ solution for the case where the bucket capacity equals two (i.e. each bucket can have two data items at most) was proposed. However, they showed that for arbitrary bucket capacity the problem is NP-complete and proposed using Simulate Annealing (SA) for optimization.

(Cho, 2000) argued that ordering based on SFCs does not take into account the uneven distribution of spatial data and the types of spatial queries. They assumed spatial data items had been grouped into pages and transformed the page ordering problem into a Traveling Salesman Problem (TSP) of a weighted graph, the weight of which is defined by the performance measurement of its cost model. They also proposed using the SA method to solve this NP-complete problem. However, it is our belief that using the SA might be too costly to be practically useful in online

applications since SA based approaches are generally very slow to achieve good results ([HREF 4]).

Join query processing involves two or multiple data sets. There are two types of problems. The first one is to determine the optimal access sequence that uses the minimum buffer without any page being fetched more than once. The second is to determine the optimal access sequence that minimizes the number of pages re-accessed given a fixed buffer size. By forming a join connectivity graph between the pages, several heuristics have been proposed (Pramanik 1985; Fotouhi , 1989; Chan 1997; Lim, 1999).

Spatial join is usually very expensive, and thus, deserves more computation resources for a better join plan. Spatial join has been extensively studied in different scenarios since it is possible that none, only one or both participating data sets have spatial index. For static data sets, a page connectivity graph can be pre-computed. Given a buffer with size *B*, (Shekhar, 2002) transferred the problem of reducing redundant I/Os to minimizing the interrelations between pages that are *B* away in the ordering sequence. They proposed using graph partition techniques for spatial clustering and provided several heuristics for better graph partition. (Xiao 2000) proposed an approximation algorithm for solving the Maximum Overlapping problem which is essentially the same as the Longest Path problem. Its result is guaranteed to be at least half of the value (in terms of size of the total overlapping between pages) of the optimal order.

Page ordering in disk-resident systems and broadcast systems, although strongly related to each other, are also quite different. The purpose of the former is to find an ordering that maximizes the overlapping spatial relationships between pages (clusters), while the latter is to minimize the total weighted intervals between the beginning and ending data items in all possible query result sets. While both problems are NP-complete, the latter is practically more complex. Intuitively, all connected edges contribute their costs to the total costs in our problem while only the neighboring edges in the optimized ordering contribute costs to the total costs in their problem. Furthermore, the purposes of page ordering in a disk-resident system and data item ordering in a broadcast system are different. The former aims at making use of data items that are already in memory as much as possible to reduce the number of I/Os while the latter aims at reducing data access time directly.

In many of the above studies, spatial relationships are represented as graphs. From graph algorithms' perspective, sequencing graph nodes can be treated as a graph layout problem. A survey on graph layout problems was presented in (Daiz, 2002). A comprehensive analysis of heuristics for both symmetric TSP (Johnson, 2002a) and asymmetric TSP (Johnson, 2002b) were provided. Spreading metric algorithms were proposed to solve a special type of graph layout problems known as the Linear Arrangement (Ordering) problem (Kuo, 1997; Rao, 1998; Even, 2000). A window-based vertex orderings with applications to circuit clustering was presented in (Alpert, 1996) where unordered vertices are iteratively added to the ordering based on their attractions to the previously ordered vertices. Various choices of attraction

can capture Depth First Search (DFS), Breadth First Search (BFS) and other well-studied graph traversals. (Bar-Yehuda, 2001) presented a polynomial time algorithm for computing an optimal orientation (ordering) of a balanced decomposition tree for the graph linear arrangement problem. Although the theoretical approximation ratios were not improved, the experiments showed good results. A multi-scale scheme for the linear arrangement problem was presented in (Koren, 2002). Different from (Bar-Yehuda, 2001) which imposes global constraints on the ordering through a Binary Decomposition Tree (BDT), it imposes many local constraints restricting small sets of vertices throughout the entire multi-scale hierarchy. The optimization methods presented in this dissertation (Chapter 6) are closely related to the work of (Bar-Yehuda, 2001).

2.4 Other Related Work

Surveys on graph partition and spatial data clustering were presented in (Alpert, 1995; Schloege, 2000) and (Han, 2001), respectively. Network clustering methods have been proposed, such as graph partition based (Shekhar, 1997a), spatial proximity based (Huang, 1996) and network traversal based (Woo, 2000).

(Seydim 2001) presented a preliminary work on location dependent queries processing. (Ren 2000) proposed using semantic caching to manage location dependent data in mobile computing. Simulation results showed that their Furthest Away Replacement (FAR) cache replacement policy has better performance than conventional page caching. (Zheng, 2001) specifically addressed location dependent

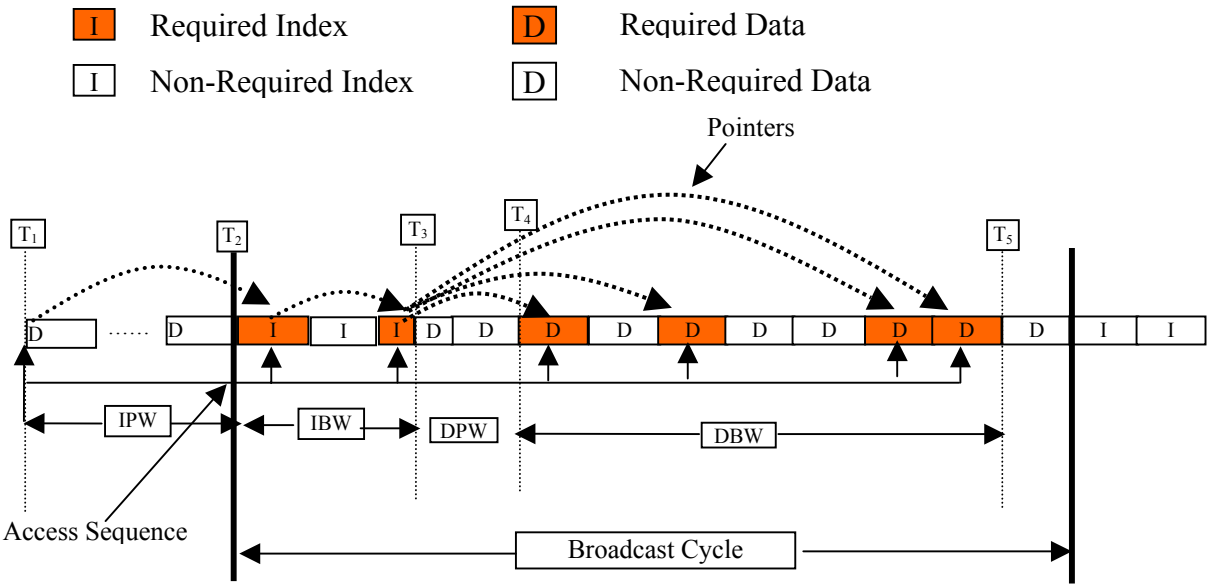
queries in a multi-cell wireless environment. They used Voronoi diagrams to construct an index and a semantic cache for improving data reusability. They also proposed several scheduling methods for handoff clients. Both (Ren 2000) and (Zheng 2001) assumed a point-to-point data communication architecture and the client can provide its exact location when it submits location dependent queries. Although a client can explicitly and continuously submit its location dependent queries and retrieve results, (Lam, 2001) addressed another kind of location dependent continuous queries. In this case, once a client has submitted a location dependent query, the system will automatically update the client's location and retrieve the updated information.

In this study we only address the problem of static geographical data broadcasting, i.e., geographical data items that do not change over time, such as locations of ATM machines and gas stations. It might be interesting to investigate on broadcasting moving objects. The possible applications include monitoring natural hazard events in remote areas and locating taxi in urban areas. Several data models (Guting, 2000; Forlizzi, 2000), storage (Chon, 2001), indexing (Kollios, 1999; Agarwal, 2000; Pfoser, 2000; Saltenis, 2000) and query processing methods (Sistla, 1997; Wolfson, 1999; Vazirgiannis, 2001) have been proposed for moving objects in disk-resident databases. However to the best of our knowledge, none of them has addressed the problem of broadcast moving data objects over air.

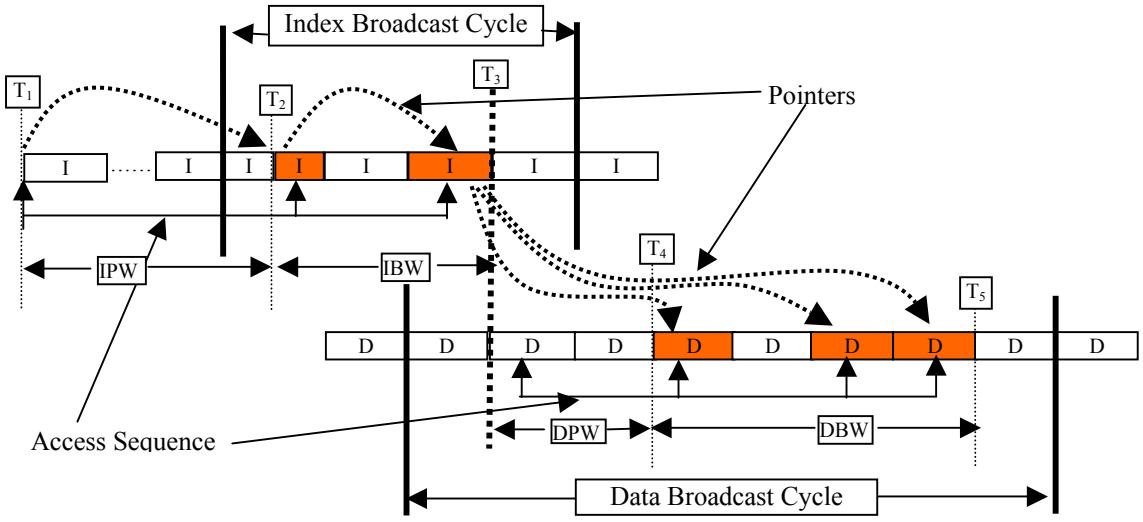
Chapter 3

Geographical Data Broadcast Cost Models

As discussed in Section 2.1, AT is further divided into two components, namely Probe Wait and Bcast Wait. 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 Probe Wait defined in (Imielinski, 1997), i.e., the time duration of getting to the nearest index segment. IBW is the time duration from the time when the first index segment is reached to the time when the last index segment is reached. 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 item is downloaded. The summation of IBW, DPW and DBW is equivalent to the Bcast Wait defined in (Imielinski, 1997). These four components are illustrated in Fig. 3-1 for two scenarios. We assume each index segment contains a certain number of pointers each of which points to a data segment, and each data segment contains only one data item. We use the intervals between the beginning and ending positions of data items as the measurements of the four components of access time as discussed in Chapter 2.



(a) Index and Data Are Multiplexed Into One Single Broadcast Channel



(b) Index and Data Use Two Separate Broadcast Channels

T_1: Time to begin accessing the broadcast channel
T_2: Time to reach the first requested index segment
T_3: Time when all requested index segments are downloaded
T_4: Time to reach the first requested data segment
T_5: Time when all requested data segments are downloaded

Fig. 3-1. The Four Components of Access Time

The definitions allow measuring the performance of index sequencing and data sequencing separately. Furthermore, they allow measuring the performance when index and data are broadcast using separate channels where the definitions in (Imielinski, 1997) only allow measuring the performance when index and data are broadcast using a single channel. In this study, we aim at reducing access time by providing a smarter arrangement of data items in the broadcast channel.

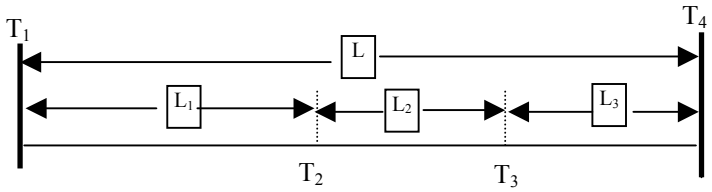
We believe reducing DBW is more important due to the following reasons. First, a client is able to remain in sleep mode during Probe Waits (IPW or DPW). However, it must switch modes during Bcast Waits (IBW or DBW). Generally we can assume that the shorter the DBW is, the fewer mode switches occur. The reason is that data segments that need to be downloaded are more likely to be in a consecutive order for a smaller DBW, and thus, the number of mode switches among them can be reduced. Compared with IBW, DBW is much larger since data is usually much larger than its index, and thus, it is critical to minimize DBW. Second, only one pointer (the position of the nearest index segment) is recorded during IPW and at most k pointers are recorded during IBW+DPW where k is the number of data items in a query result set. However, there could be up to k data items that are recorded during DBW. Since usually the size of a data item is much larger than that of a pointer, it is desirable to reduce DBW as much as possible to reduce energy consumption for storing data items that have already been downloaded during the data access process.

We are also interested in the total access time to data, i.e., AT_{Data}=DBW+DPW. In this dissertation we consider two popular scenarios that involve both DBW and DPW. The first is that we assume the data and the index are multiplexed into one single broadcast channel. Here both the data and index segments are broadcast only once in a cycle and the index segments are placed ahead of the data segments. Without loss of generality we assume the index is placed at the beginning of a broadcast cycle since the broadcast sequence is cyclic. In this case, a client will have to wait for the beginning of a broadcast cycle (Fig. 3-1a) before accessing data items. The second scenario is that the data and index segments are broadcast in two separate channels and both the data and index segments are broadcast only once in their respective cycles. In this case, a client might begin to access the data channel at any position after it retrieves the pointers to the data segments from the index channel (Fig. 3-1b). We call the first scenario as "Multiplexing Scheme" or MUL, and the second scenario as "Separate Channels Scheme" or SEP.

Throughout this dissertation, we use "access time cost" to refer to the access time needed to complete a complex query over a broadcast sequence. For the rest of this chapter, we first propose the cost models to compute the DBW and the AT_{Data} under the two scenarios for a single complex query. We then present the cost models to compute the total DBW and the access time to data in the two scenarios for all queries over a data set. We handle spatial range queries for point data and network path queries for graph data separately although they both follow a similar framework.

3.1 Cost Models for Processing a Single Complex Query

Suppose the length of the broadcast cycle of a data channel is L. Let L_2 denote the access time of a single complex query result, i.e., DBW. Let L_1 and L_3 denote the time before L_2 and after L_2 in the broadcast sequence as shown in Fig. 3-2. Note that $L=L_1+L_2+L_3$.



T_1: Time to begin a broadcast cycle
T_2: Time to access the first required data item
T_3: Time to access the last required data item
T_4: Time to end a broadcast cycle

Fig. 3-2. Illustration of L_1, L_2 and L_3

We next compute the average cost for a single complex query under the Multiplexing scheme and Separate Channels scheme assuming a client begins to access the data channel randomly.

- Multiplexing Scheme: the access time cost to the data channel can be calculated as:

$$AT_{Data}^{Mul} = L_1 + L_2$$

- Using Separate Channels: there are three cases that a client might begin to access the data channel. We compute their total access time costs seperately and then compute the average access time cost.

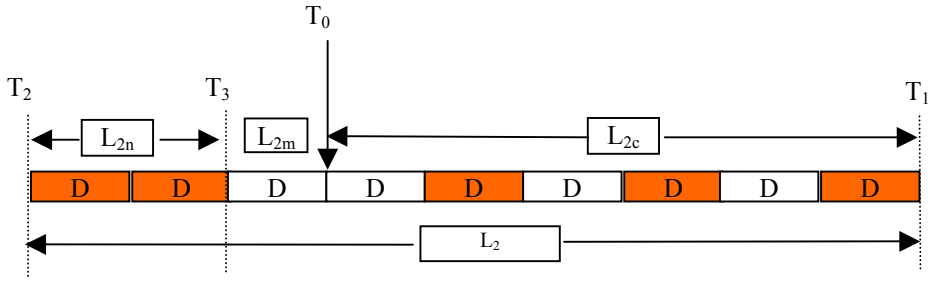
Case1: a client begins to access the data channel during L_1. Suppose its initial access poistion is i then it has to wait for an amount of time equivalent to (L_1-i) before downloading data during L_2, thus the total costs over all possible i is:

$$Cost1 = \sum_{i=0}^{L_1-1}(L_1 - i + L_2) = \frac{L_1(L_1+1)}{2} + L_1 * L_2$$

Case 2: a client begins to access the data channel during L_2. Regardless of its initial access position, it has to wait for the whole broadcast cycle to retrieve all the data items. Thus the total cost is:

$$Cost2 = L * L_2$$

Note that there is a slight overestimation here as shown in Fig. 3-3 (the shaded data items are the ones in the query result set). We need a part of L_2 in the current broadcast cycle (L_{2c}) and a part of L_2 in the next broadcast cycle (L_{2n}). However their total might be less than L_2 since the rest of L_2, i.e., L_{2m}= L_2 - L_{2c} - L_{2n}, is not required. Suppose there are n data items in the query result set and they are evenly distributed among L_2, then the overestimation on average is the half of the interval between two required data items among L_2, i.e., $L_2/(2n)$. We omit this overestimation to make our result more concise and easy to use as can be seen in computing the average shortly.



T_0: Time to begin access the broadcast channel
T_1: The end of the current broadcast cycle
T_2: The beginning of the next broadcast cycle (The same as T_1)
T_3: Time to access the last required item in the next broadcast cycle

Fig. 3-3. Illustration of the Overestimation in Case 2

Case 3: a client begins to access the data channel in L_3. It has to wait for the rest of the time in L_3 in the current broadcast cycle and L_1+L_2 in the next broadcast cycle. Thus the total cost is:

$$Cost3 = \sum_{i=0}^{L_3-1}(L_3 - i + L_1 + L_2) = \sum_{i=0}^{L_3-1}(L - i) = L_3 * L - \frac{L_3(L_3-1)}{2}$$

Since there are totally L intial access positions, the average is :

$$\begin{aligned}
AT_{Data}^{Sep} &= \frac{1}{L}(Cost1 + Cost2 + Cost3) \\
&= \frac{1}{L}[\frac{L_1(L_1+1)}{2} + L_1 * L_2 + L * L_2 + L_3 * L - \frac{L_3(L_3-1)}{2}] \\
&= \frac{1}{L}[\frac{(L_1+L_3)(L_1-L_3+1)}{2} + L_1 * L_2 + L * (L_2 + L_3)] \\
&= \frac{1}{L}[\frac{(L-L_2)(L_1-L_3+1)}{2} + L_1 * L_2 + L * (L - L_1)] \\
&= \frac{1}{L}[L^2 - L_1 * (L - L_2) + \frac{(L-L_2)(L_1-L_3+1)}{2}] \\
&= \frac{1}{L}[L^2 - \frac{(L-L_2)(2 * L_1 - L_1 + L_3 - 1)}{2}] \\
&= \frac{1}{L}[L^2 - \frac{(L-L_2)(L_1+L_3-1)}{2}] \\
&= \frac{1}{L}[L^2 - \frac{(L-L_2)(L-L_2-1)}{2}]
\end{aligned}$$

From the result we can see that the average access time to the data channel in the Separate Channels scheme is determined only by L and L_2. Usually the number of data items in a query result is far fewer than the total number of data items in a whole broadcast cycle, thus it is reasonable to assume L-L_2>>1. Under this assumption, the formula can be simplified as $AT_{Data}^{Sep} = \frac{L}{2} + L_2 - \frac{{L_2}^2}{2L}$

To further investigate the relationship between the average access time to the data channel and L_2, we can rewrite the formula as follows:

$$AT_{Data}^{Sep} = \frac{1}{L}[L^2 - \frac{(L-L_2)^2 - (L-L_2)}{2}]$$

$$= \frac{1}{L}[L^2 - \frac{(L-L_2)^2 - 2*(\frac{L-L_2}{2}) + (\frac{1}{2})^2 - \frac{1}{4}}{2}]$$

$$= \frac{1}{L}[L^2 + \frac{1}{8} - \frac{(L-L_2-\frac{1}{2})^2}{2}]$$

Since $L_2<L$, the average cost decreases monotonically as L_2 decreases.

Let function $g(L_2)$ be $g(L_2) = \frac{L}{2} + L_2 - \frac{L_2^2}{2L}$. We will use this in the following analysis.

3.2 Spatial Range Query for Point Data

In this section we first compute all possible query result sets and their weights by exploring spatial semantics of a point data set and then we develop the cost models for DBW and the two scenarios of AT_{Data} by summarizing the weighted access time of individual query results we have developed in the last section.

Let DS=$[x_1,x_2)\times [y_1,y_2)$ be the data space that defines all the geographical data points. Let the range query window size be (q_x,q_y). We define the Extended Region R_u of point P_u as the rectangle of size (q_x,q_y) centered at P_u. As shown in Fig. 3-4, the distribution of the centers of the query window regions of size (q_x,q_y) that contain the data item P_u is the extended region of R_u. Furthermore, from Fig. 3-5 we can see that the distribution of the centers of the query window of size (q_x,q_y) that contain both the data items P_u and P_v is the intersection of their extended regions R_u

and R_v. This relationship can be extended to higher orders, up to the intersected region among all n extended regions.

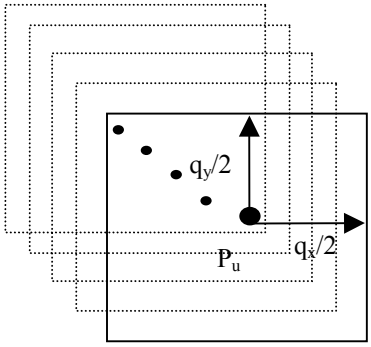


Fig. 3-4. The Possible Distribution of Centers of Query Regions That Contain P_u

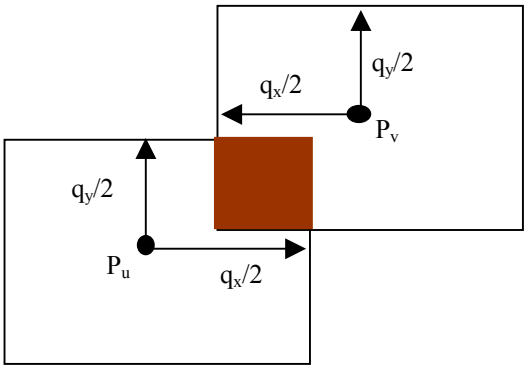


Fig. 3-5. The Possible Distribution of Centers of Query Regions That Contain Both P_u and P_v (Shaded Area)

We assume that all the locations inside the study region are equally likely to be the users' locations at the time they issue a spatial range query, i.e., the centers of query windows. The access frequencies of a subset of data points resulted from the spatial range location-dependent queries is proportional to the area of the distributions

of the centers of the query windows that contain the subset of data points. Thus, the access frequency of such data points in the subset is proportional to the intersection area of their extended regions *(area*). Assume the number of spatial range queries requested in the studied area is a fixed number (*M*), let *c*= $\frac{M}{(x_2 - x_1)*(y_2 - y_1)}$, then the access frequency of the query result set (*freq*) is *freq=c*area*. For the sake of simplicity we omit the constant factor *c* and only use *area* as the access frequency for a query result set. Note that the access frequency of a subset *S* is no less than the access frequency of another subset *S*' if $S \subseteq S'$ since the intersection area of the extended regions of the points in *S*' is a subset of the intersected area of the extended regions of the points in *S*.

Let A_i be the area of R_i, $A_{i,j}$ be the intersection area of R_i and R_j, …, $A_{1,2...n}$ be the intersection area of R_1, $R_2 \ldots R_n$. Let $\widetilde{A}_i$ be the part of A_i that solely contains point P_i, $\widetilde{A}_{i,j}$ be the part of $A_{i,j}$ that solely contains points P_i and P_j, … $\widetilde{A}_{1,2...n}$ be the part of the intersection area of R_1, $R_2 \ldots R_n$ that contains all *n* points. It is easy to see that we have the following relations:

$$\widetilde{A}_i = A_i - \bigcup_{j=1, i \neq j}^{n} A_{i,j}$$

$$\widetilde{A}_{i,j} = A_{i,j} - \bigcup_{k=1, i \neq j \neq k}^{n} A_{i,j,k}$$

...

$$\widetilde{A}_{1,2,...n} = A_{1,2..n}$$

Let function $\pi(u)$ map point u to its position in the broadcast sequence. According to our previous definitions and assumptions, the DBW for a single query result set that contains k data items $n_1, n_2 \ldots n_k$, which is the definition of L_2 (c.f. Fig. 3-2), is $\max(\pi(n_1),\pi(n_2),...\pi(n_k)) - \min(\pi(n_1),\pi(n_2),...\pi(n_k))$. Correspondingly, L_1 is $\min\{\pi(n_1),\pi(n_2),...\pi(n_k)\}$.

The total DBW cost for a query window (q_x,q_y) is the summation of the weighted DBW for all possible query result sets. For a query result set that contains only two points i and j, the interval between them in the broadcast sequence is $|\pi(i)-\pi(j)|$. Its weight is $\widetilde{A}_{i,j}$ and its weighted DBW is $\widetilde{A}_{ij}^{(q_x,q_y)} * |\pi(i)-\pi(j)|$. Note that i and j can be any two points the extended regions of which intersect with each other. Similarly the weighted DBW for a query result set that contain three points, i, j and k, is $\widetilde{A}_{i,j,k}^{(q_x,q_y)} * (\max(\pi(i),\pi(j),\pi(k)) - \min(\pi(i),\pi(j),\pi(k)))$ and so on. Thus the total DBW cost for all possible query result sets with query window of size (q_x,q_y) can be written as follows:

$$
\begin{aligned}
& Cost^{(q_x,q_y)} \\
&= \sum_{1\le i<j\le n} \widetilde{A}_{ij}^{(q_x,q_y)} * |\pi(i)-\pi(j)| \\
&+ \sum_{1\le i<j\le k\le n} \widetilde{A}_{i,j,k}^{(q_x,q_y)} * (\max(\pi(i),\pi(j),\pi(k)) - \min(\pi(i),\pi(j),\pi(k))) \\
&+ \ldots \\
&+ \widetilde{A}_{1,2,\ldots n}^{(q_x,q_y)} * (\max(\pi(1),\pi(2)...\pi(n)) - \min(\pi(1),\pi(2)...\pi(n)))
\end{aligned}
$$

The final total cost of DBW is the summation of DBW $Cost^{(q_x,q_y)}$ over all possible query windows Q, i.e.,

$$
\begin{aligned}
Cost^{DBW} &= \sum_{(q_x,q_y)\in Q} Cost^{(q_x,q_y)} \\
&= \sum_{(q_x,q_y)\in Q} \sum_{1\le i<j\le n} \widetilde{A}_{i,j}^{(q_x,q_y)} * |\pi(i)-\pi(j)| \\
&+ \sum_{(q_x,q_y)\in Q} \sum_{1\le i<j\le k\le n} \widetilde{A}_{i,j,k}^{(q_x,q_y)} * (\max(\pi(i),\pi(j),\pi(k)) - \min(\pi(i),\pi(j),\pi(k))) \\
&+ \ldots \\
&+ \sum_{(q_x,q_y)\in Q} \widetilde{A}_{1,2,\ldots n}^{(q_x,q_y)} * (\max(\pi(1),\pi(2)\ldots\pi(n)) - \min(\pi(1),\pi(2)\ldots\pi(n)))
\end{aligned}
$$

Let

$$
\begin{aligned}
w_i &= \sum_{(q_x,q_y)\in Q} \widetilde{A}_i^{(q_x,q_y)} \\
w_{i,j} &= \sum_{(q_x,q_y)\in Q} \widetilde{A}_{i,j}^{(q_x,q_y)} \\
w_{i,j,k} &= \sum_{(q_x,q_y)\in Q} \widetilde{A}_{i,j,k}^{(q_x,q_y)} \\
&\ldots \\
w_{1,2,\ldots n} &= \sum_{(q_x,q_y)\in Q} \widetilde{A}_{1,2,\ldots n}^{(q_x,q_y)}
\end{aligned}
$$

Then

$$
\begin{aligned}
DBW = & \\
& \sum_{1\le i<j\le n} w_{i,j} * |\pi(i)-\pi(j)| \\
& + \sum_{1\le i<j\le k\le n} w_{i,j,k} * (\max(\pi(i),\pi(j),\pi(k)) - \min(\pi(i),\pi(j),\pi(k))) \\
& + \ldots \\
& + w_{1,2,\ldots n} * (\max(\pi(1),\pi(2)\ldots\pi(n)) - \min(\pi(1),\pi(2)\ldots\pi(n)))
\end{aligned}
$$

Similarly, the costs of access time to data under the Multiplexing scheme, AT_{Data}^{Mul}, and Separate Channels scheme, AT_{Data}^{Sep} are as follows:

$$
\begin{aligned}
& AT_{Data}^{Mul} = \\
& = \sum_{1 \le i \le n} w_i * \pi(i) \\
& + \sum_{1 \le i < j \le n} w_{i,j} * \max(\pi(i), \pi(j)) \\
& + \sum_{1 \le i < j \le k \le n} w_{i,j,k} * \max(\pi(i), \pi(j), \pi(k)) \\
& + \ldots \\
& + w_{1,2,\ldots n} * \max(\pi(1), \pi(2) \ldots \pi(n))
\end{aligned}
$$

$$
\begin{aligned}
& AT_{Data}^{Sep} = \\
& \sum_{1 \le i < j \le n} w_{i,j} * g(|\, \pi(i) - \pi(j) \,|) \\
& + \sum_{1 \le i < j \le k \le} w_{i,j,k} * g(\max(\pi(i), \pi(j), \pi(k)) - \min(\pi(i), \pi(j), \pi(k))) \\
& + \ldots \\
& + w_{1,2,\ldots n} * g(\max(\pi(1), \pi(2) \ldots \pi(n))) - \min(\pi(1), \pi(2) \ldots \pi(n)))
\end{aligned}
$$

Note that we omit the access times of queries that only have a single data item (which is $L/2$) in the SEP scheme since they are constant and do not contribute to the determination of optimal ordering.

3.3 Network Path Query for Graph Data

Let V denote the vertex set of the network. Let S_{ij} denote a path sequence with access frequency $f(i,j)$ for source vertex i and destination vertex j. Assume the order of the k vertexes in the path are S^0_{ij}, S^1_{ij}, ...S^k_{ij}. The total DBW cost for the queries of all pairs of the shortest paths between any two vertexes over the broadcast sequence can be computed as follows:

$$DBW = \sum_{i \in V, j \in V} f(i,j) * (\max(\overset{k}{\underset{m=0}{\forall}} \pi(S^m{}_{ij})) - \min(\overset{k}{\underset{m=0}{\forall}} \pi(S^m{}_{ij})))$$

This is essentially the same as the cost for spatial range queries. $f(i,j)$ is equivalent to $w_{i,j}$, $w_{i,j,k}$, ...$w_{1,2\ldots,n}$ depending on the number of vertexes along the path between vertex i and j. If we group $f(i,j)$ by k=$|S_{ij}|$ and denote this as $f(i,j)^k$, then $f(i,j)^2 \equiv w_{i,j}$, $f(i,j)^3 \equiv w_{i,j,k}$, ... $f(i,j)^n \equiv w_{1,2,..n.}$

Similarly, the costs of access time to the data channel under the Multiplexing and Separate Channels schemes for network path query of graph data are as follows correspondingly:

$$AT_{Data}{}^{Mul} = \sum_{i \in V, j \in V} f(i,j) * \max(\overset{k}{\underset{m=0}{\forall}} \pi(S^m{}_{ij}))$$

$$AT_{Data}{}^{Sep} = \sum_{i \in V, j \in V} f(i,j) * g(\max(\overset{k}{\underset{m=0}{\forall}} \pi(S^m{}_{ij})) - \min(\overset{k}{\underset{m=0}{\forall}} \pi(S^m{}_{ij})))$$

Now our problem is how to minimize the DBW and the access time to data under the Multiplexing and Separate channels schemes. To solve this problem, we first present a unified hypergraph representation of the spatial range query for point data and network path query for graph data in Chapter 4. Based on this representation

we relate the optimization problem with the well-known graph layout problems. We then present the optimization methods for the access time under the three cost models which are presented in Chapter 6.

3.4 Discussions on Related Work

The cost model presented in (Chung, 2001) is the work most related to our cost model under the Multiplexing scheme. The cost model is restated as follows using our definitions for the purpose of consistency. Let $t_j=d_j+\delta_j$ where d_j is the time to access the j^{th} required item and δ_j is the time between the j^{th} and $(j+1)^{th}$ required data items in a broadcast channel. Let $F(y)$ be the access time of query q begin to access the broadcast channel at time (position) y in the broadcast sequence. They considered two scenarios when accessing the j^{th} data item in the broadcast sequence. When a user begins to access the broadcast channel during $0\sim|d_j|$, the user has to wait for the whole broadcast cycle to retrieve d_j, thus $F(y)=L$. During $|d_j| \sim t_j$, a user begins to access the channel at position y during δ_j and it takes $L-y$ units time at most to retrieve the j^{th} data item, thus $F(y)=L-y+|d_j|$. The average cost for processing the query can be written as follows as derived in (Chung, 2001):

$$
\begin{aligned}
Cost &= \int_0^L F(y)\,dy = \frac{1}{L}\sum_{j=1}^{n}\int_0^{t_j} F(y)\,dy \\
&= \frac{1}{L}\sum_{j=1}^{n}\left[\int_0^{|d_j|} F(y)\,dy + \int_{|d_j|}^{t_j} F(y)\,dy\right] \\
&= \frac{1}{L}\left[\int_0^{|d_j|} L\,dy + \int_{|d_j|}^{t_j} (L - y + |d_j|)\,dy\right] \\
&= L - \frac{1}{2L}\sum_{i=1}^{n} (t_j - |d_j|) \\
&= L - \frac{1}{2L}\sum_{i=1}^{n} \delta_j{}^2
\end{aligned}
$$

The authors also defined a new measure called Query Distance (QD) to approximate the cost for a complex query q under ordering π as follows: $QD(q,\pi)=L-\delta_k$, where δ_k is the maximum of all δ_js. If $\delta_k=L-L_2$ we can see that QD is L_2. Thus their proposal using QD to approximate the average access time is similar to ours using L_2 to approximate $g(L_2)$ as explained in details in Section 6.5.

The authors also claimed that if $QD(q, \pi_1) \geq QD(q, \pi_2)$, then $\text{cost}(\pi_1) \geq \text{cost}(\pi_2)$ as the rationale for the approximation. Unfortunately their proof on the induction of "if $\delta_k(\pi_1) \leq \delta_k(\pi_2)$, then $\sum_{i=1}^{n} \delta_i(\pi_1)^2 \leq \sum_{i=1}^{n} \delta_i(\pi_2)^2$ " is incorrect. A counter example is as follows. Suppose we have only three data items. Their δ_js under π_1 are 2, 4, 4 and 3, 3, 4 under π_2. Thus $\delta_k(\pi_1)$ and $\delta_k(\pi_2)$ are both 4. Although we have $\delta_k(\pi_1) \leq \delta_k(\pi 2)$, $\sum_{i=1}^{n} \delta_i(\pi_1)^2 = 2*2+4*4=20 \leq \sum_{i=1}^{n} \delta_i(\pi_2)^2 = 3*3+3*3=18$ does not hold in the second induction step.

Since they failed to prove the correctness of using QD (L_2) to approximate the cost ($g(L_2)$), it is natural for readers to question the validity of our cost model and approximation proposal (using L_2 to approximate $g(L_2)$). We next show that under certain circumstances we can reduce their complex cost model to ours and prove the validity of our approximation proposal.

In (Chung, 2001), when $\delta_k=L-L_2$ and $\delta_j=0$ (for all $j=1\sim n$ and $j \neq k$), this cost model is essentially the same as ours. Although this condition does not hold in most

cases, when δ_k is large, δ_k^2 dominates the term $\sum_{i=1}^{n} \delta_j^2$ due to the quadratic relationship and we can use δ_k^2 to approximate $\sum_{i=1}^{n} \delta_j^2$. We believe that the condition can be satisfied in the orderings based on reasonably good heuristics that utilize spatial relationship. For orderings based on these heuristics, data items in the same queries are likely to be close to each other in the orderings which makes $\delta_k = L - L_2$ much larger than other δ_i.

In summary, the work (Chuang 2001) first provided an accurate yet complex cost model. They proposed to use a simpler parameter (QD) to approximate the cost computed by the model. Unfortunately, their proof of the monotonic relationship between QD and the cost model is not correct. On the other hand, we make some approximations (by omitting L_{2m} as discussed in Section 3.1) at the beginning of deriving our cost model. The derived cost model is simpler and easier to compute. In addition, we are able to show the monotonic relationship between L_2 and $g(L_2)$ which provides a theoretical foundation for approximation.

Chapter 4

Hypergraph Representation of Spatial Semantics

In Section 3.2 and Section 3.3 of Chapter 3 we treat each query result set individually without considering the relationships between the elements in multiple query result sets. From the graph theory perspective we can represent all the elements in the query result sets as nodes and the query result sets as the hyperedges. A hyperedge is an extension of a regular edge and consists one or more nodes. This representation allows the application of many well-studied hypergraph/graph algorithms to our applications directly or after some modifications. In this chapter we first introduce the hypergraph representation for both spatial range queries and network path queries. We then present an efficient method to compute the weights of the hypergraph for spatial range queries in point data sets. We propose an approximation method to covert a hypergraph into a regular graph that allows the use of regular graph algorithms, such as traversal and partition algorithms, in generating heuristic orderings (Chapter 5) and developing optimization methods (Chapter 6).

4.1 The Hypergraph Representation

In our hypergraph representation, the node set is all the points in a point data set or all vertexes in a graph data set. The points of a spatial range query result set $\{n_1,n_2..n_k\}$ or the vertexes along a path between vertex i and vertex j, i.e., $\{S^0_{ij}, S^1_{ij}, \ldots S^k_{ij}\}$ form a subset of the node set V. Each of such subsets makes a hyperedge

(Fig. 4-1 and Fig. 4-2). Note that the circled numbers represent the hypergraph nodes and the numbers inside rectangles denote the weights of the hyperedges.

In Fig. 4-1, the query window size is (10,10), thus all the four points have extended areas (R_1, R_2, R_3 and R_4) of size 100, i.e. A_1=A_2=A_3=A_4=100. The intersection of R_1 and R_2 is R_{12} (shaded) and the area of which is A_{12}=36. Similarly we have A_{13}=16, A_{23}=56, A_{24}=7, A_{34}=12, A_{123}=14, A_{234}=4. By using the Inclusion-Exclusion Theorem in set operations, we can compute $\widetilde{A}_1$ as

$$\widetilde{A}_1 = A_1 - (A_{12} + A_{13} - A_{123}) = 100 - 36 - 16 + 14 = 62$$

Similarly we can have $\widetilde{A}_2$=19, $\widetilde{A}_3$=34, $\widetilde{A}_4$=85, $\widetilde{A}_{12}$=22, $\widetilde{A}_{13}$=2, $\widetilde{A}_{23}$=38, $\widetilde{A}_{24}$=3, $\widetilde{A}_{34}$=8, $\widetilde{A}_{123}$=14, $\widetilde{A}_{234}$=4.

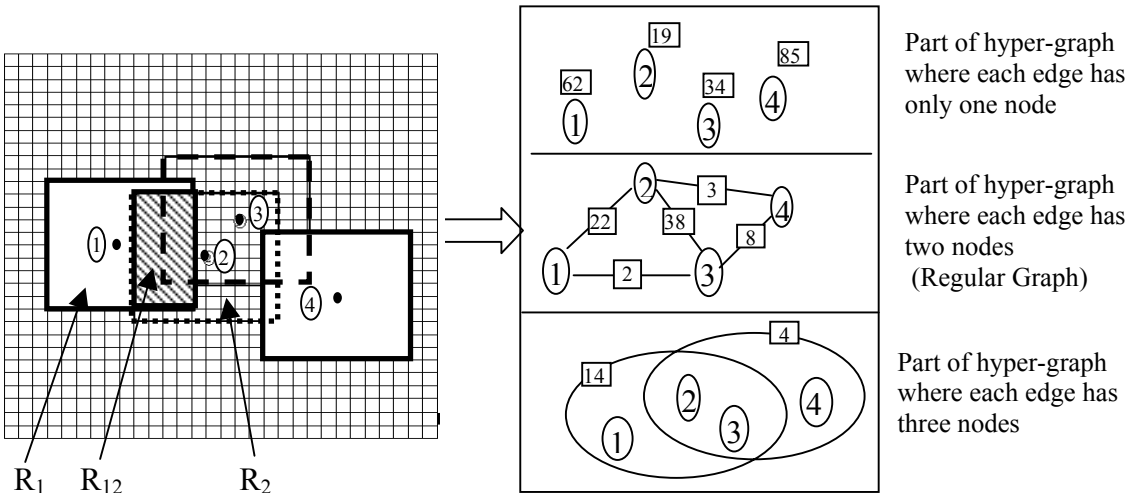


Fig. 4-1. Hypergraph Representation of Spatial Relationships of Point Data

In Fig. 4-2, the first component of the hypergraph representation is the four vertexes. They can be treated as the degenerated cases for path query results where the source vertex *i* is the same as the destination vertex *j*, i.e., $||S_{ij}||$=1. The second component includes paths (1,2), (2,3) and (2,4) where $||S_{ij}||$=2, the third component includes paths (1,2,3), (2,3,4) and (1,2,4) where $||S_{ij}||$=3.

For spatial range queries, we use w_i, $w_{i,j}$, $w_{i,j,k}$…$w_{1,2,..n}$ as the weights for the hyperedges that have 1,2,…n nodes. For network path queries, we use $f(i,j)$ as the weights for the hyperedges. Although the weights of the hypergraph can be computed based on the geometry of the points in a point data set as shown in the next section, the weights of the hypergraph of graph data, which are the access frequencies of the paths in the graph data, can only be measured or estimated by domain experts. For example, the access frequency of a highway between two cities is determined by the mutual attraction factors and the transportation cost between them that is decided not only by the distance but also by road conditions and other complex interrelated factors.

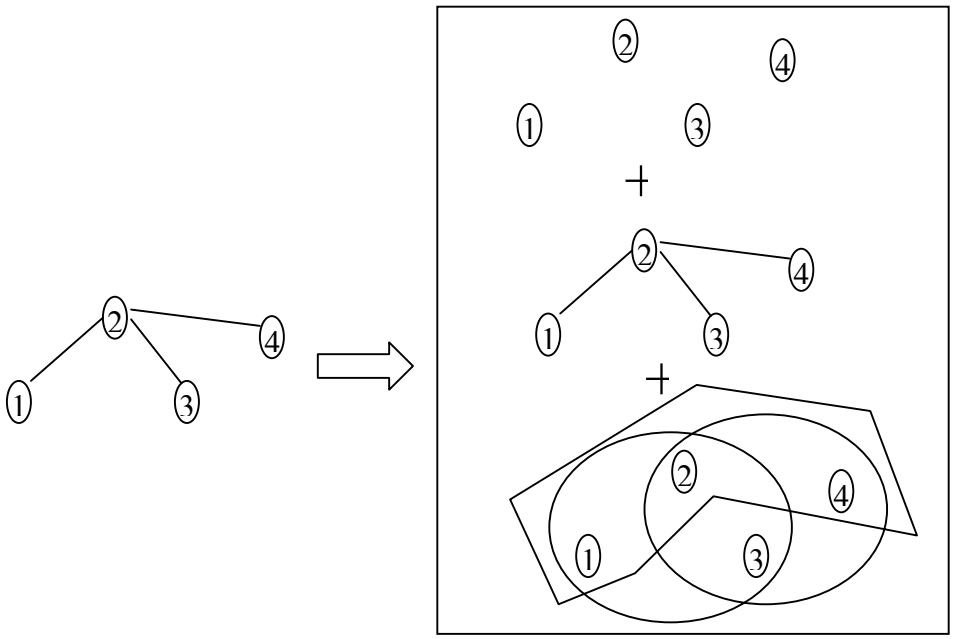


Fig. 4-2. Hypergraph Representation of Spatial Relationship of Graph Data

4.2 Computing Hypergraph Weights for Point Data

Although the method based on the Inclusion-Exclusion theorem used in the last section to compute the weight of the hypergraph edges is straightforward, the implementation based on it is not very efficient. We will refer to this method as the Inclusion-Exclusion method. The reasons are as follows. Suppose the number of data points in a data set is n. First of all we need to compute $A_i, A_{i,j} \ldots A_{1,2\ldots n}$ for all possible $0<i<n$, $0<i<j<n$, …etc. This can be done by computing the intersections among all R_is to get $A_{i,j}$, computing the intersections among all $R_{i,j}$ to get $A_{i,j,k}$, etc. This process might repeat up to n rounds and the number of regions to be intersected in each round increases monotonically. For each round, the computation complexity can be reduced from $O(N^2)$ of an intuitive method, which exhaustively examines the intersection between two regions, to $O(N*\log N)$ by using the well known Line Sweeping algorithm (Cormen, 2001) where N is the number of regions to be intersected in each round. Thus the total number of intersections performed is in the order of $\sum_{i=1}^{n} N_i * \log N_i$ where N_i is the number of regions to be intersected in each round i. N_1 is the number of regions to be intersected in the initial data set, i.e., $N_1=n$. Since $N_1<N_2\ldots<N_n$, this number is at least in the order of $(n^2 log(n))$. Second, maintaining the relationships among $A_i, A_{i,j} \ldots A_{1,2\ldots n}$ in order to compute $\widetilde{A}_i$, $\widetilde{A}_{i,j}$, … $\widetilde{A}_{1,2\ldots n}$ is either very time consuming or very space consuming. Furthermore, the implementation of the Line Sweeping algorithm is not trivial.

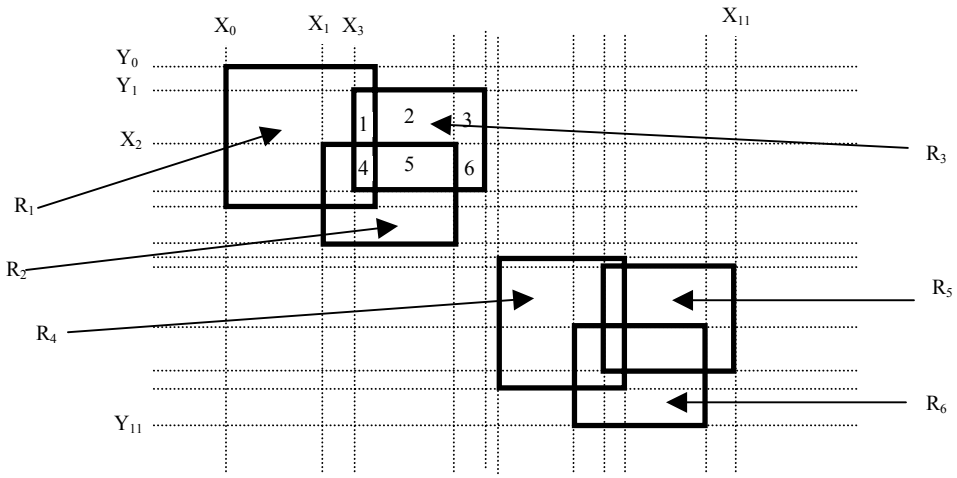


Fig. 4-3. Computing the Smallest Intersection Regions

We next describe a simple intuitive method with a complexity of O(n^3). The idea behind the method is that we first compute all the possible smallest intersection regions (e.g., regions 1-6 in Fig. 4-3) and then assemble them in their corresponding result set (Cixiang Zhan, Environmental Research Institute - ESRI, 2001, Personal Communication). We call this method Intersect-Assemble method. The process is shown in Fig. 4-4. It first sorts the coordinates of all points along the *x* and *y* directions respectively. For each of the two neighboring coordinates along the *x* and *y* direction, x_i and x_{i+1} and y_i and y_{i+1}, a smallest rectangle can be constructed using these coordinates. For each of such rectangles, the algorithm examines which original regions contain it and all the labels of these original rectangles form a final result set. If multiple smallest rectangles are contained in a result set, their area will be summed up and set as the area of the result set.

Assume there are six regions (R_1 through R_6) to be intersected as shown in Fig 4-3. They have 12 distinct coordinates along the *x* and *y* directions, thus there are (12-

1)*(12-1) smallest rectangles. We focus on the three regions on the top-left since they intersect with one another while do not intersect with the other three regions. There are six smallest rectangles in region R_3 as numbered 1 through 6. Rectangle 1 is contained in both region R_1 and R_3, thus its label is L_{13}, similarly rectangle 4 is labeled as L_{123} and rectangle 5 is labeled as L_{23}. Rectangles 2, 3 and 6 are all labeled as L_3 and their areas are summed up. Thus we have

$$\widetilde{A}_3 = \text{area(2)+area(3)+area(6)},\ \widetilde{A}_{23} = \text{area(5)},\ \widetilde{A}_{13} = \text{area(1)},\ \widetilde{A}_{123} = \text{area(4)}$$

```
Input: An array of n regions Rect
Output: A hash table H, each entry of which stores the label (hash key) and the area value.

Set H to empty
 Extract x and y coordinates of the points in the n regions into two arrays X and Y with size
      of 2*n.
Sort these two arrays, in ascending order.
For each i from 0 to 2*n-1
     For j  from 0 to 2*n-1
       Build a rectangle (tempRect) with the following four coordinates (X[i],Y[j],
       X[i+1],y[j+1])
       Set the label set (L) of tempRect to empty.
           For k=0 to n-1
                If tempRect is within Rect[k] then
                     Add k to L
                End if
           End for k
        If L is not empty
           If L is already in H
              H(L)=H(L)+area(tempRect)
           Else
              H(L)= area(tempRect)
            End if
        End if
    End For j
  End For i
End
```

Fig. 4-4. The Intersect-Assemble Method for Generating Hyperedge Weights

It takes four comparisons to determine whether tempRect is within an original rectangle. Assuming it takes Q_0 time on average to perform a lookup in a hash table, since in the worst case there is always an update for each lookup which is also assumed to take Q_0 *time*, the cost of processing a single smallest rectangle is at most $4*n+ Q_0+ Q_0=4*n+2*Q_0$. Since there are $(2n-1)*(2n-1)$ of such smallest rectangles (i.e., *i* and *j* loops), the complexity of the algorithm is $(2n-1)*(2n-1)*(4n+2*Q_0)$. Since usually it takes sub-linear complexity to look up a data item in a hash table by using a reasonable hash function, i.e., $O(Q_0)<O(n)$. Thus the above algorithm has approximately $O(n*n*n)=O(n^3)$ complexity. Although the theoretical complexity of the Intersect-Assemble method is higher than that of Inclusion-Exclusion method, it is still competitive due to the simple implementation when *n* is small. However, the computation cost is prohibitive when *n* is big and we need a more efficient method.

We observe that the number of intersection rectangles associated with each intersection line, i.e., a unique coordinates along either the *x* or *y* direction as shown in Fig. 4-3, is very likely to be much smaller than *n*. A region often only intersects with a limited number of other regions since the size of a region is limited. According to a hypergraph representation, this also means the number of nodes in a hyperedge is bounded by a constant. In fact, this is one of the assumptions in our complexity analysis of one of the proposed optimization methods as detailed in Section 6.6 of Chapter 6. As an example, in Fig. 4-3, the number of regions to be intersected is six while the maximum number of regions that intersect with one another is only 3.

Based on this observation, we propose a new method in computing the weights of the hypergraph for a point data set. Since the method explores R-Tree spatial index (Guttman, 1984), we call it the R-Tree based method.

For each of the extended regions of the points in a point data set, the method first retrieves all the extended regions that intersect with it. It then applies the Intersection-Assemble method on these regions. For each of the entries in the resultant hash table, the method first checks whether the label of the entry contains the label of the extended region under consideration. If true, the method further checks whether the entry has already existed in the output hash table. It will add the area value of the entry to the output hash table if the entry does exist, otherwise it will add the entry to the output hash table. The process is shown in Fig. 4-5.

```
Input: An array of n regions Rect
Output: A hash table H, each entry of which stores the label (hash key) and the area value

1. Construct an R-Tree for Rect
2. For each leaf node n_i in the R-Tree (i.e., an extend region)
        2.1 Retrieve all the extended regions that intersect n_i from the R-Tree and store it in array
              Rect'
        2.2 Apply the Intersect-Assemble method for Rect' (c.f. Fig.4-4) and store the result in a
              hash table H'
        2.3 For entry (L,A) pair in H' where L is the hash key and A is the value of the entry
              Test whether the entry can be found in H and set the corresponding flag array
                element F(L)
            End for
        2.4 For entry (L,A) pair in H'
              If L contains the label of n_i and F(L) then
                  If L is already in H then
                      H(L)=H(L)+A
                  Else
                      H(L)=A
                  End if
              End If
          End For
End For
```

Fig. 4-5. The R-Tree Based Method for Generating Hyperedge Weights

The efficiency of the R-Tree based method is achieved by only performing the expensive $O(n^3)$ Intersection-Assemble method for a subset of the point data set. Although an extended region of a point might be involved multiple times when calling the Intersection-Assemble method, the overall computation complexity can be reduced as analyzed in the following. Although the strict complexity analysis of R-Tree and its variant R*-tree (Beckmann, 1990) is not available, they are experimentally shown to be low-cost spatial indexing methods which is super linear but sub-quadratic. We assume R-Tree construction complexity is $O(n*log(n))$ and the search complexity in an R-Tree is $O(log(n))$ (which is the lower bounds of sorting and searching in tree data structures), where n is the number of points in a point data set. We also assumes the number of extended regions intersecting with the extended region of a node is bounded by a constant as discussed above, the cost of the Intersection-Assemble method on them is also independent of n. We use C_{IA} to denote such a cost. Thus the total cost of the R-Tree based method in the best scenario is in the order of $n*log(n)+n*(log(n)+C_{IA})$, i.e. $O(n*log(n)+n*C_{IA})$. Theoretically when n is big, the $n*log(n)$ term will dominate and reduce the complexity to $O(n*log(n))$. However, for practical n values (e.g., 100-10000), the C_{IA} constant is likely to be much larger than $log(n)$. Thus we are expecting this algorithm to be linear with respect to n with a large hidden factor.

4.3 Relationship with MinLA

Using the hypergraph representation, our problem of minimizing the total query cost is related to the graph Minimum Linear Arrangement problem (MinLA) as explained below.

The goal in MinLA is to find an ordering that minimizes the weighted sum of the edge lengths. The edge length is defined as the difference of the positions of the beginning node and the ending node of an edge in an ordering. Let $w(u,v)$ be the weight of the edge (u,v) and $\pi(u)$ be the position mapping function the same as defined in Chapter 3. The weighted sum of the edge lengths with respect to an ordering is defined as follows (Daíz, 2002).

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

As we can see that the definition of DBW is the same as that of $la(G)$ except that we need to replace a regular graph edge with a hypergraph edge and set u and v to be the nodes that have the maximum/minimum positions in the ordering. However, $AT_{\text{Data}}^{\text{Mul}}$ depends only on the maximum position of nodes the in a hyperedge and $AT_{\text{Data}}^{\text{Sep}}$ has a quadratic relation with respect to the difference of the maximum/minimum positions of the nodes in a hyperedge. Both of them are non-linear with respect to the difference (or the "edge length" defined in MinLA) that makes the existing MinLA methods (Bar-Yehuda, 2001; Koren, 2002) not suitable to solve our problems. Nevertheless they can be the basis for further improvement.

Actually we propose to use DBW to approximate $AT_{\text{Data}}^{\text{Sep}}$ and then use an efficient MinLA algorithm to optimize it which is discussed in detail in Chapter 6.

4.4 Efficient Hypergraph Data Structures

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

First, the nodes of the hyperedges are stored sequentially in an array (called a node array) and the positions that mark the endings of the hyperedges in the array are stored in another array (called an edge index array), which serve as indexes to the nodes in the hyperedges. The weights of the hyperedges are stored in a third array (weight array). An inverse hypergraph is also built for the hypergraph. The inverse hypergraph has an array to store the hyperedge IDs that contain the nodes in the original graph sequentially (called an edge array). Similar to the index array used in a hypergraph, the positions that mark the endings of hyperedges containing the nodes are stored in a second array (called a node index array) of the inverse hypergraph. We will show how these arrays can be used to efficiently manipulate a hypergraph through an example shortly.

To build the hypergraph data arrays, a hypergraph data file is needed to be scanned twice. The hypergraph data file is assumed to be stored in the order of hyperedges and the numbers of nodes and hyperedges are given. Each hyperedge is

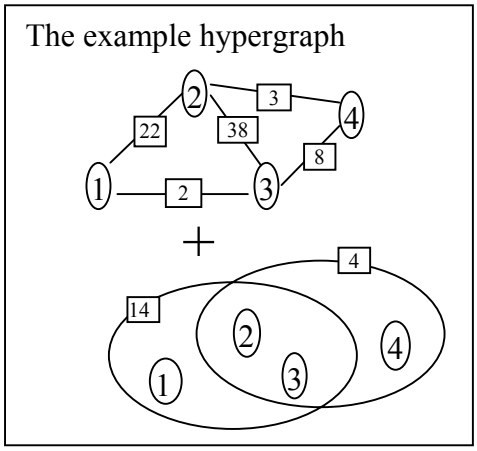
stored as a record in the data file with the following format: its number of nodes in the hyperedge, followed by the IDs of the nodes, then its weight. During the first scan, the edge index array and the weight array are filled. After the first scan, the total number of nodes in all hyperedges is computed and the node array of the hypergraph is then allocated. During the second scan, the node array is actually filled while scanning.

Once the hypergraph data arrays are constructed, the inverse hypergraph data arrays can be constructed in a more efficient manner since their constructions require only memory access to the hypergraph data arrays. The node array of a hypergraph is first scanned to compute the number of hyperedges associated with each node. These numbers are accumulated and filled in the node index array of the corresponding inverse hypergraph. The node array of the hypergraph is then scanned the second time to find the edge number of each node that is contained in the edge and which is then put into the proper position of the edge array of the inverse hypergraph.

From the construction processes, we can see that both the time complexity and space complexity of the node array, the edge index array and weight array in a hypergraph are linear with respect to the number of total nodes in all hyperedges, the number of hyperedges and the number of hyperedges, respectively. Similarly, both the time complexity and space complexity of the edge array and the node index array are linear with respect to the number of total nodes in all hyperedges and the number of nodes in a hypergraph, respectively.

By using the arrays of a hypergraph and the corresponding inverse hypergraph, we can perform the following operations efficiently. First, we can retrieve all the nodes in a hyperedge of ID *e* by first retrieving the ending position of the previous edge (*e*-1) and the ending position of *e* from the edge index array and then retrieving all the nodes from the node array of the hypergraph. Second, we can retrieve all the hyperedges that contain a node *v* by first retrieving the ending position of the previous node (*v*-1) and the ending position of *v* from the node index array and then retrieving all the IDs of the hyperedges between the two positions from the edge array of the corresponding inverse hypergraph. Note that we can retrieve the weight of a hyperedge by accessing the weight array of the hypergraph by its ID *e* directly.

We next illustrate these hypergraph data structures using the hypergraph shown in Fig. 4-1. For the sake of simplicity, we remove the edges that have only one node. The node array, edge index array and the weight array of the hypergraph, and the edge array and the node index array of the inverse hypergraph are shown in Fig. 4-6. From the node index array of the inverse hypergraph, we know that there are 8-3=5 hyperedges passing through node 2. By seeking the 4^{th} through the 8^{th} elements in the edge array of the inverse hypergraph we know that they are hyperedges 2, 3, 5, 6, and 7. By accessing the weight array of the hypergraph, we know that their weights are 38, 22, 3, 14 and 4, respectively. Furthermore, by looking up at the edge index array of the hypergraph, we know that there are 13-10 =3 nodes in the hyperedge 6 and we know that they are the 11^{th} to the 13^{th} elements in the node array of the hypergraph, which are nodes 1, 2 and 3 respectively.



Hyperedge	Node List	Weight
1	1,3	2
2	2,3	38
3	1,2	22
4	3,4	8
5	2,4	3
6	1,2,3	14
7	2,3,4	4

Hypergraph:

Node Array:
[1,3,2,3,1,2,3,4,2,4,1,2,3,2,3,4]
Edge Array:
[0,2,4,6,8,10,13,16]
Weight Array:
[2,38,22,8,3,14,4]

Inverse Hypergraph:
Edge Array:
[1,3,6,2,3,5,6,7,1,2,4,6,7,4,5,7]
Node Array:
[0,3,8,13,16]

Fig. 4-6. Illustration of the Hypergraph Data Structures

4.5 Converting A Hypergraph to A Regular Graph

As we have discussed in Section 4.1, the spatial relationships between points in a point data set are implicit. The purpose of computing the weights of the hyperedges in our hypergraph representation for a point data set is to make such spatial relationships explicit. For graph data, the spatial relationships are explicitly expressed as the access frequencies of paths and can be observed or estimated by domain experts. Many graph algorithms related to ordering or partitioning have been well studied, however, unfortunately, there are no corresponding hypergraph algorithms. In order to apply these algorithms, one way is to convert a hypergraph to a regular graph through approximations.

The approximation is simple for point data since we can build a graph by adding an edge between two nodes (points) if their extended regions intersect with each other and using the intersection area as the weight of the graph. For graph data we propose to build a graph by summarizing the access frequencies of the paths that pass by an edge in the original graph. The weight of edge (u,v) in the converted graph is defined as follows:

$$w(u,v) = \sum (\underset{(u,v)\in S_{ij}}{\forall} f(i,j))$$

where S_{ij} is a path that contains edge (u,v) and $f(i,j)$ is the access frequency of path S_{ij}.

We call the resulting regular graph an Edge Access Frequency Graph (EAFG). EAFG has the same topology as the original graph (not the derived hypergraph) since neither the node set and the edge set is changed. The only difference is the weights associated with them. In the example shown in Fig. 4-7, there are two paths passing by edge (3,7), namely path $S_{4,6}$ of [4,3,7,8,6] and path $S_{1,5}$ of [1,3,7,5]. The edge access frequency of edge (3,7) is computed as $w(3,7) = f_{4,6}+f_{1,5}$.

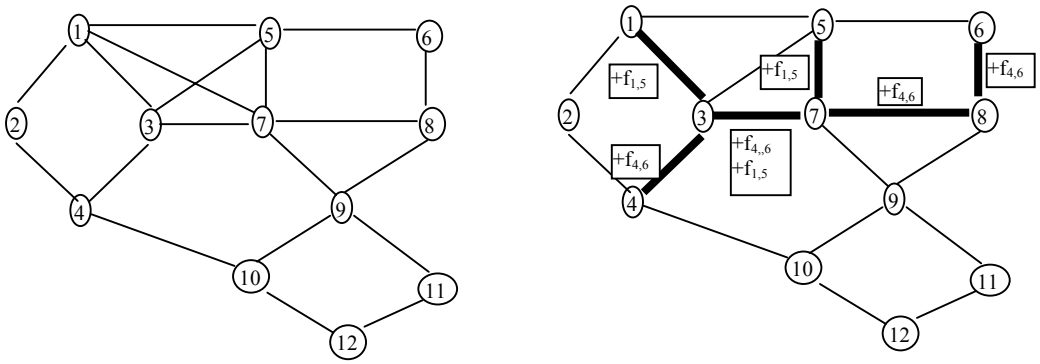


Fig. 4-7. Illustration of EAFG Derivation

Since there are $n!$ orderings for n points (for point data) or vertices (for graph data) which are exponential with respect to n, it is impractical to enumerate all of them and find a globally optimal ordering. Thus low-cost approximation algorithms are desirable. We first present a family of ordering heuristics for both point data and graph data in Chapter 5. They can be used either as the low-cost ordering techniques when speed is a primary concern, or as the initial orderings for further improvement using the optimization methods we are going to present in Chapter 6.

4.6 Discussions On Related Work

There are some existing work proposed to represent access patterns as data access graphs and then use graph-theoretical approaches to generate broadcast sequences. Compared to representing complex query result sets as hypergraphs directly, these representations are essentially approximate in nature, similar to what we proposed in Section 4.5 of this chapter for spatial data.

(Si, 1999) presented a Semantic Ordering Model (SOM) for relational/object-oriented database broadcast using entity type (field/attribute) as the basic broadcast unit (data item) and represented the access patterns of a broadcast database by a directed graph as shown in Fig. 4-8. Each node v_i is associated with a cost of accessing an entity type (s_i, which reflects the total size of all entities belonging to the entity type). Each node v_i is further associated with a probability p_i denoting the probability of being accessed as the first entity in a query. P_i can be estimated as n_i/n where n_i is the number of queries that accesses v_i as the first entity type and $n=\Sigma n_i$.

Each edge e_{ij} is associated with a weight α_{ij} indicating the likehood that v_j will be accessed by a query that v_i has been accessed by the same query.

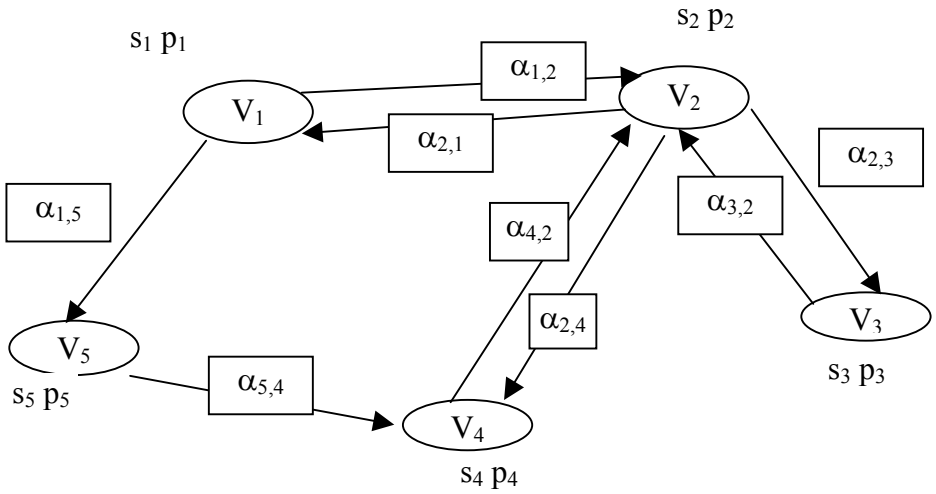


Fig. 4-8. The SOM Model and its Graph Representation (Si, 1999)

The SOM model and the directed graph representation are suitable for the scenario where the precedence relationship between v_i and v_j can be easily determined, such as the referential integrity constraints in relational databases and the parent-children relationship in object-oriented databases. They are not applicable for the scenario where a set of entity types is involved in a query processing but has no precedence order between the entity types in the set. It is also worth to note that the SOM model and the graph representation are designed for using an entity type (attribute) as the minimum broadcast unit, i.e., vertical partition of database. Due to the bandwidth limitation, usually only hot data items and frequent attributes are chosen to broadcast. If almost all attributes are required by clients which is very likely in practice, it will take almost a whole broadcast cycle to retrieve only a single

data item in the vertical partitioning broadcast scheme. Also since location is the most selective attributes in spatial queries (where often most attributes are needed) and usually only a small portion of all the data items is in a spatial query result set, we believe tuple (record) selection rather than entity type selection is more practical in geographical data broadcast. Thus the SOM model and its graph representation are not suitable for geographical data broadcast.

The graph representation in (Lee, 2002) was also based on directed graph. For each query pattern, they classified the related attributes into three groups, the select attribute (SA), the join attributes(JA) and the project attributes (PA). They assumed the order of the three groups to be SA→JA→PA. However, the attributes inside each group are unordered. An initial graph can be built as proposed in (Si, 1999). The unordered pairs in an attribute group in a query pattern are scanned through the rest query patterns to determine their precedence relationship by using the SA→JA→PA orders. For the attributes that still do not have a precedence relationship with any other attributes, all the attributes in SA have directed edges with the attributes in JA, and similarly, all the attributes in JA have directed edges with all the attributes in PA. During the process, if there are two directed edges between node u and node v with access frequency f_{uv} and f_{vu} then the two directed edges will be replaced by one directed edge with access frequencies f_{uv}-f_{vu}. Although (Lee, 2002) provided several additional methods in determining the precedence relationship between two attributes according to SQL query patterns, it has the same problems as (Si, 1999). In the simplest SQL query patterns where SA and JA are empty and only PA exists, it will

be impossible to determine the precedence relationship between the attributes in PA. Although it is beneficial to put attributes that are often queried together near each other, unfortunately, it is impossible to do so based on the graph representation of query patterns proposed in (Lee, 2002a).

The method presented in (Lee, 2003) also represented query patterns as a graph. They assumed a data item (which can be a tuple/record or an object) is the basic broadcast unit and the data items in a query result set are unordered. Thus their problem is essentially the same as ours. They constructed a graph before sequencing as well. For each query and for any two data items in the query, they will put an undirected edge between the two data items with the weight being the access frequency of the query. The final graph is generated by combining identical edges and setting the summation of their weights as the final weights for the combined edges. The resulting graph in (Lee, 2003) is a combination of m complete graph where m is the number of queries and is very likely to be dense, which makes it hard to handle.

For spatial range query on point data, we can prove that the graph generated by the method of (Lee, 2003) is exactly the same as the approximation graph generated by the method proposed for point data in Section 4.5 in this chapter. In order to do so, it is sufficient to prove the weight of an edge between two arbitrary nodes in the graph is the same in the two methods. The weight of the edge between any two nodes (without lose of generality, we assume they are node 1 and node 2) is $A_{i,j}$ in our method. The possible query result set that contains data items 1 and 2 are $\{1,2\}$, $\{1,2,3\}$,$\{1,2,4\}$,…$\{1,2,n\}$,$\{1,2,3,4\}$,…$\{1,2,…n\}$. Their weights, according to

the spatial semantics presented in Section 3.2 of Chapter 3 are $\widetilde{A}_{1,2}$, $\widetilde{A}_{1,2,3}$, $\widetilde{A}_{1,2,4}$ … $\widetilde{A}_{1,2,n}$ $\widetilde{A}_{1,2,3,4}$ … $\widetilde{A}_{1,2\ldots n}$. The weight of edge (1,2) based on the method proposed in (Lee, 2003) is the summation of these weights. By using the Inclusion-Exclusion theorem, the summarized weight is $A_{1,2,}$ which is the same as our result. The method of (Lee, 2003), although applicable for handling generic complex queries, suffers from the exponential number of possible queries with respect to the number of data items when applied to spatial range queries. Furthermore, even if the number of queries is bounded by a constant *M*, their graph construction method has the complexity of $O(\sum_{i=1}^{M} m_i^2)$ where m_i is the number of data items in a query. Our method is much simpler by exploring spatial semantics. The worst case complexity of our method is O(*n*log(n)*) using the Line Sweeping algorithm, where *n* is the number of nodes in the graph, or the number of points in the data set. Although for all *i*, m_i is less than *n*, $O(\sum_{i=1}^{M} m_i^2)$ is likely to be much more expensive than O(*n*log(n)*).

Chapter 5

Ordering Heuristics

In this chapter, we discuss several heuristics to generate orderings of point data. They are based on the state-of-the-art techniques in spatial data handling. Although we try our best to cover all known heuristics that are related to our work, obviously, they are far from complete. Nevertheless, we manage to classify them into a coherent framework in hope that new heuristics can find their places in the classification structure and be plugged into the architecture for further optimization and evaluation as shown in Chapter 6 and Chapter 7.

5.1 Overview

The ordering heuristics can be generally classified into two categories. The first category is geometry-based and the second category is graph-based. The first category can be further divided into hierarchical and non-hierarchical sub-categories. The hierarchical sub-category includes orderings generated by traversal of hierarchical spatial clustering trees and traversal of spatial indexing trees, such as Quad-Tree and the family of R-Trees (Gaede, 1998). The second sub-category of geometry-based heuristics includes all kinds of Space Filling Curves (SFCs), such as Z-ordering and Hilbert (Gaede, 1998). The graph-based heuristics can also be classified into the hierarchical and non-hierarchical sub-categories. The hierarchical graph-based heuristics are based on the traversal of recursive graph partition trees

(Schloegel, 2000). The non-hierarchal graph-based heuristics include classical graph-traversals, such as Breadth-First Search (BFS), Depth-First Search (DFS) and ordering by node degree (Gondhalekar, 1997), node weight and edge weight ([HREF 5]). We discuss heuristics based on node degree, node weight and edge weight and their complexities in Section 5.5 since they are less explored. The BFS/DFS heuristics are purely based on the graph topology while heuristics based on spanning tree (Cormen, 2001) are combinations of graph topology and edge weights. We discuss spanning tree based heuristics in Section 5.6 since it is the Maximum Spanning Tree rather than the Minimum Spanning Tree that is proposed for broadcast ordering (Liberatore, 2002), and there are several interesting points that need further discussions. The classification is illustrated in Fig 5-1.

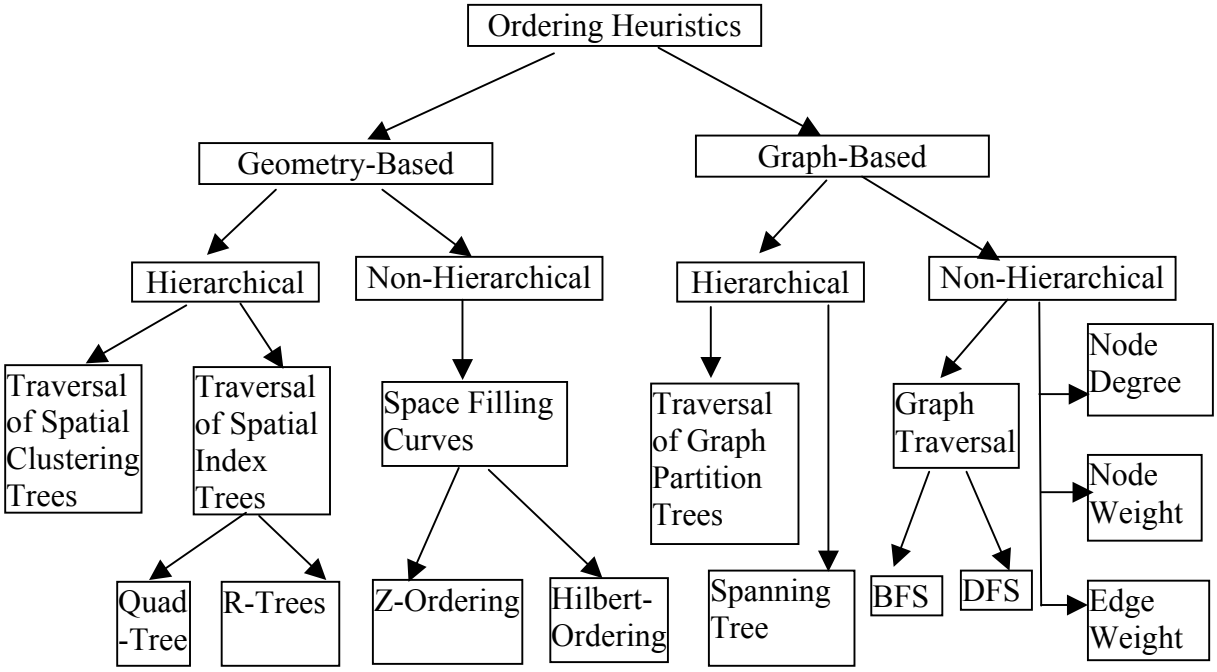


Fig. 5-1. The Classification Structure of Ordering Heuristics

Except for constructing graph partition trees, the heuristics we present here have very low computation overheads. Constructing SFC codes for a point is independent of the number of points in a data set while sorting the codes and generating a SFC ordering takes $O(n*log(n))$ time using the quick sort algorithm (Cormen, 2001). The time complexity for constructing spatial indexing trees varies, but popular spatial indexing tree methods are sub-quadratic in order to be practically useful. Hierarchical spatial clustering algorithms that have time complexity from $O(n)$ to $O(n^2)$ have been proposed (Han, 2001) where n is the number of points in the data set. The BFS has $O(V+E)$ complexity and the DFS has $\Theta(V+E)$ complexity, the MST (or its variants) of Kruskal's algorithm and Prim's algorithm has complexities of $O(E*log(V))$ and $O(E*log(E))$, respectively (Cormen, 2001), where V is the number of nodes and E is the number of edges in a graph. Furthermore, many of the data structures that are needed by the heuristics already exist in spatial databases for other purposes (e.g. indexing), thus the extra cost, if there is any, to generate an ordering is generally only $O(n)$ for traversing different types of trees (Cormen, 2001). The low cost of these heuristics make it suitable to use them to generate broadcast sequences if speed is the primary concern or use them as the initial orderings for further optimization if the query processing cost is the primary concern.

We explain the following ordering heuristics in detail in the subsequent sections due to their popularity in practice, namely R-Tree traversal ordering (Section 5.2), Hilbert SFC ordering (Section 5.3), graph-partition tree traversal ordering (Section 5.4), ordering based on degree/weights (Section 5.5) and spanning tree

ordering (Section 5.6). In addition, we discuss the more recent ordering methods in Section 5.7. These ordering heuristics are used for comparisons in Chapter 7.

5.2 R-Tree Traversal Ordering

The R-Trees are extensions of B-Trees to K-dimension (Guttman, 1984) and originally designed for disk-resident spatial data indexing. Putting spatially adjacent data items into the same node in an index tree, the search space is reduced quickly as the level of the R-Tree increases. Since the R-Tree is balanced, the search speed is logarithmic with respect to the number of data items it is indexing.

In this study, we are concerned more on ordering quality rather than search speed. In Fig. 5-2, points 1 and 2 have more chances to be queried together, thus putting points 1 and 2 close to each other in the broadcast sequence, instead of putting them far away from each other, will be very likely to reduce the total access time to the data broadcast channel. Similarly we can argue for data points 3 and 4. An ordering can be generated by the traversal of the branches in their R-Tree index.

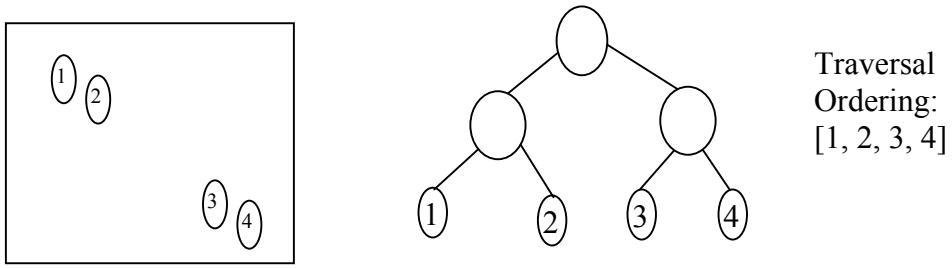


Fig. 5-2. A Simple Point Data Set, Its R-Tree and Traversal Ordering

However, the order of the sibling branches, which determines the access time of a query result set that has points stranding over multiple branches, can not be optimally determined by in-order traversal (left-to-right) as shown in Fig. 5-3.

Suppose the R-Tree branching factor is 3 and fill factor is 0.5 then there are at least two branches within a node. Considering the case where we insert four data items in the order of 1, 2, 3 and 4, then we get the R-tree as shown in Fig. 5.3. The R-tree traversal ordering will be [1,4,2,3]. If our query region consists of points 1 and 2 then the total length of access time will be 3 while it could be as small as 2. If our query region consists of point 3 and 4 then we might need the next broadcast cycle to get point 3. On the other hand, the order of [1,2,3,4] is optimal for both of the spatial range queries. Our optimization methods proposed in Chapter 6 first decompose an R-Tree into a binary tree and then switch the left sub-trees and the right sub-trees of the binary tree recursively to find the best ordering in 2^{n-1} possible orderings.

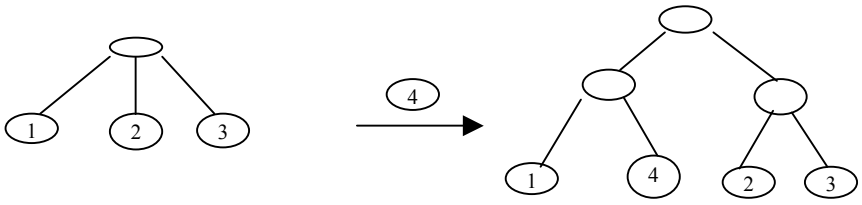


Fig. 5-3. Illustration of Non-Optimal R-tree Traversal Ordering

5.3 Hilbert SFC Ordering

The SFCs first partition the whole space of a data set with a grid. Each of the grid cells is labelled with a unique number that defines its position in the total order. The points in the given data set are then sorted to generate a sequence. We choose Hilbert ordering as one of the ordering heuristics to be evaluated in Chapter 7 primarily because of its theoretical capability in preserving the proximity of two

dimensional points in one dimensional sequence (Jagadish, 1990) and its practical popularity as well.

Although generating Hilbert SFC for arbitrary dimensional data is not trivial, it is relatively easy to do so in two-dimensional data. An algorithm to generate a two-dimensional Hilbert SFC can be found in (Shekhar, 2003). Fig 5-4 shows an example of how to construct a Hilbert SFC recursively where grid resolution refers to the number of bits used to represent a point coordinates (in both the *x* and *y* directions). Examples of the Hilbert SFCs of grid resolution 2 and of resolution 4 are shown in the top and bottom parts of Fig. 5-4, respectively. We next illustrate how to generate a resolution 4 SFC from a resolution 2 SFC.

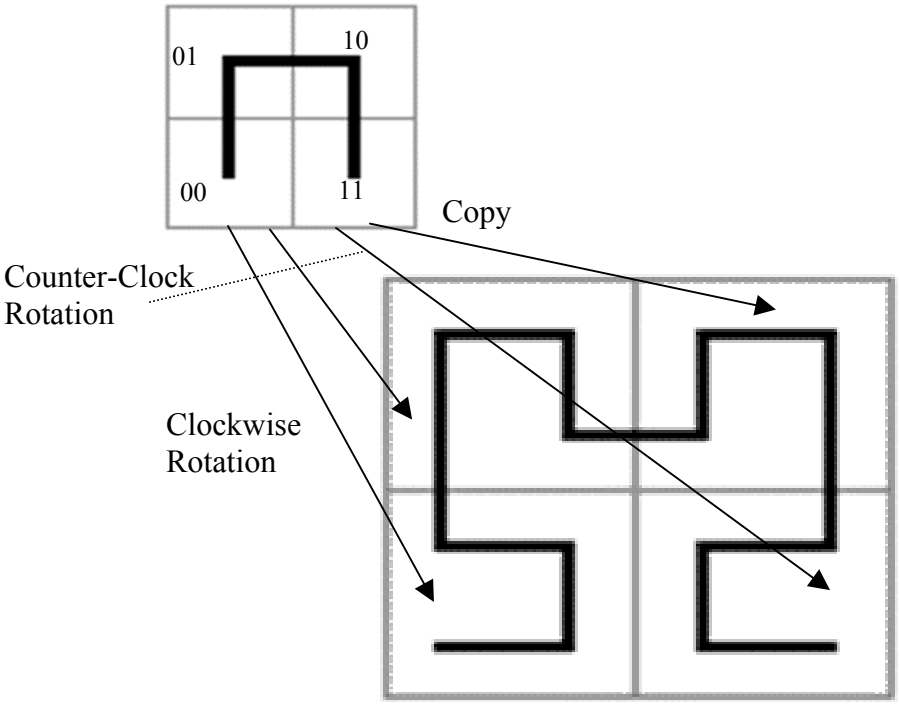


Fig. 5-4. Illustration of Recursively Generating Hilbert SFC

In Fig. 5-4, the grid resolution 2 Hilbert SFC is treated as a unit. A translation is performed for a copy of the unit and these two units are put at the top of the grid resolution 4 Hilbert SFC under generation. A copy of the two units are then made. A 90 degree clockwise rotation is performed for the left unit and a 90 degree counter-clockwise rotation is then performed for the right unit. These two units are put at the bottom to complete the resolution 4 Hilbert SFC.

Unlike generating R-Trees, generating Hilbert code for a point is independent of other points in the data set. The finer the resolution, the more number of bits will be needed to represent the coordinates and the more time is needed to generate a Hilbert code. Since using 20 bits to represent a coordinate has a resolution of about 3 meters even the range of the data set is the whole global Earth which is sufficient for broadcast geographical information in most cases, we can treat the computation cost to generate the Hilbert code for a point as a small constant. Thus generating Hilbert codes for all the points in a data set is linear with respect to the number of points. The complexity in the sorting step is in the order of $O(n*log(n))$ using the quick sort algorithm. Thus the complexity of the complete method is in the order of $O(n*log(n))$.

One problem we found regarding Hilbert-ordering (and SFC orderings in general) is that, although data items adjacent to each other in the generated ordering is also adjacent to each other in the original space (to a certain extent), the other way around is not true. Two adjacent points might fall far apart in the SFC orderings. In Fig.5-5, suppose that our data points have an ordering of [0,1,7,8,9,14] according to

the Hilbert SFC ordering. If a query window contains points 1 and 14, then the access time will be almost the whole broadcast cycle. On the other hand, if we order the data objects by traversing the dynamically generated R-Tree (Fig. 5-6) then the latency could be only 2 for the same query window. We will evaluate their performances of Hilbert SFC ordering using both real and synthetic data sets in Chapter 7.

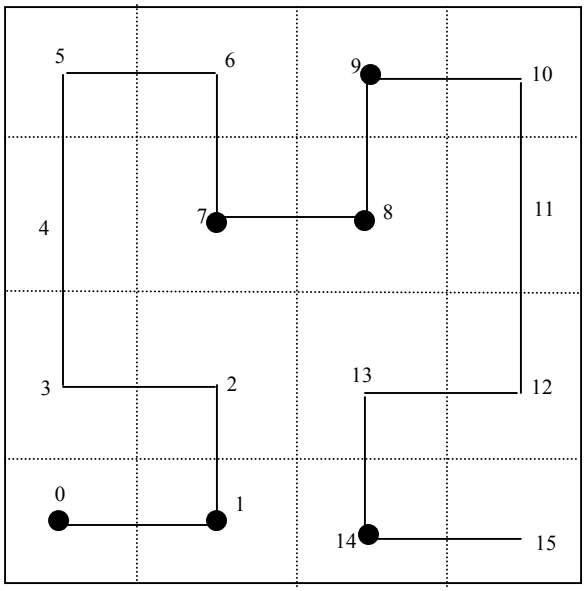


Fig. 5-5. Illustration of a Non-Optimal Hilbert SFC Ordering for a Query Window

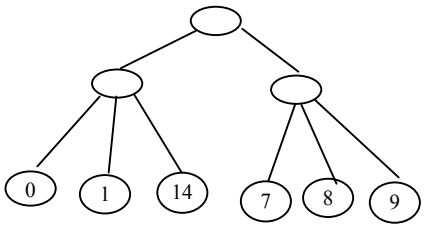


Fig. 5-6. R-Tree Generated by Inserting Points Dynamically for the Data Set in Fig. 5-5 (Hilbert-Codes Are Used As Leaf Node Labels)

5.4 Graph Partition Tree Traversal Ordering

We can make an analogy between using R-Trees for geometric data indexing, where geometric space is hierarchically partitioned, and hierarchical graph partition.

Traversal of a hierarchical graph partition tree can also generate an ordering as shown in Fig. 5-7. In the figure, the graph is partitioned into two sub-graphs divided by thick dotted line. In the partition tree, we represent the whole graph with root node T_0 and the two sub-graphs with its two child nodes, T_1 and T_2, respectively. We further partition the right sub-graph into two sub-graphs divided by a thin dotted line. Since the numbers of nodes in these sub-graphs are below a predefined threshold (four in the example), they are not further partitioned. We represent these two sub-graphs with leaf nodes 1 and 2 respectively and put them as the child nodes of T_1. Similarly the left sub-graph is further partitioned into two sub-graphs divided by another thin dotted line. We present them with leaf nodes 3 and 4 respectively and put them as the child nodes of T_2.

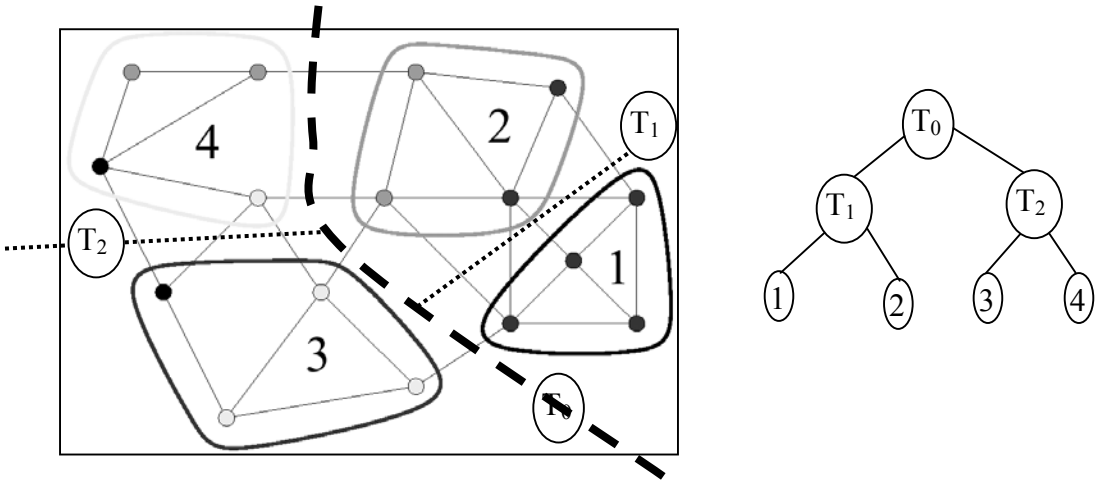


Fig. 5-7. Illustration of Graph Partition Tree

In graphs, the relationships between two nodes are explicitly defined by the weights of the edges between them. To take such explicit relationships into consideration, graph partition is a natural choice to retain the main features of the

graph while reducing the complexity of the relationships, which is the basis for many graph problems (see the surveys in (Alpert, 1995; Schloege, 2000) for details).

Although the graph data in this study is assumed to be geometric, nodes in the graph data that are spatially closest to each other do not always have the strongest relationship. For example, two big cities linked by a highway have a stronger geographical relationship in road networks than two small towns even though the distance between them is much longer than that between the two small towns. In addition, for some transformed graphs, such as EAFG discussed in Section 4.4 of Chapter 4, geometric information is generally irrelevant to their semantics. Generally speaking, traversal of a graph partition tree can be a good ordering heuristic since it keeps nodes with strong relationships close to each other, provided that efficiency is not a problem. In this study we use the recursive graph partition technique implemented in METIS and HMETIS ([HREF 6]), which is freely available over the Internet.

5.5 Ordering based on Degree/Weight

We use the graph shown in Fig. 5-8 to demonstrate the ordering heuristics discussed in this section and in the next section. The graph has 12 nodes and 20 edges.

(Gondhalekar, 1997) provided two heuristics for optimizing broadcast sequence under the scenario where a user begins accessing the broadcast channel from the beginning of a broadcast cycle and there are only two data items in a query. The MAX heuristic orders of the data items by their descending out degree. For

undirected graph, the out degree of a node is defined as the nodes that are directly connected with the node. For the graph in Fig. 5-8, the ordering will be: [2, 6, 0,4, 8, 1,3,7,9, 5, 10, 11]. If there is a tie during the order, the tie will be broken by node ID (or node number), i.e., the node with smaller ID value will be placed first.

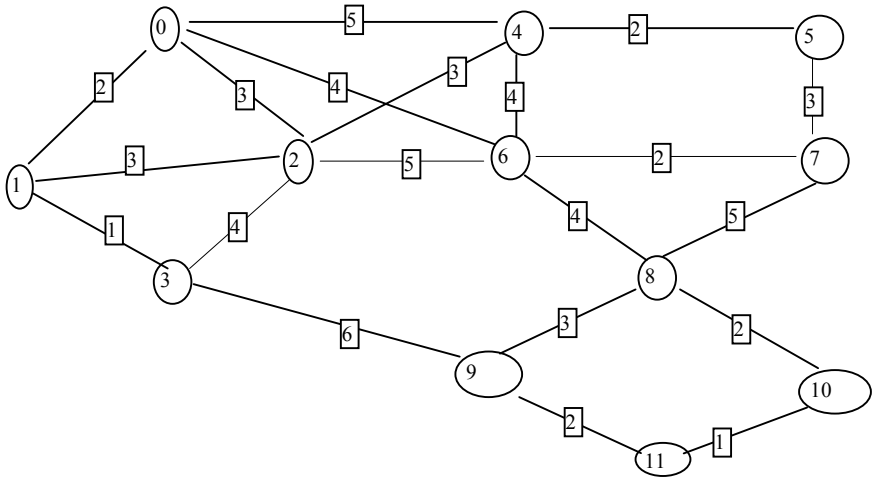


Fig. 5-8. An Illustrative Graph for Degree/Weight Based Orderings and Spanning Tree Ordering

The MAX-LD heuristic consists of two steps. The first step is to obtain an initial ordering by sorting the vertices by descending degree, i.e., using the MAX heuristic. In the second step, the following operation is repeated for i=1,…,n-1: if the left degree of vertex i+1 exceeds that of vertex i, the positions of the two vertices are interchanged. The left-degree of vertex v is defined as the number of edges that have v as the ending node in an ordering sequence, i.e., $ld(v)=|\{(u,v):(u,v)\in E \cap (\pi(u)<\pi(v)\}|$. This is based on the observation that $\sum_{(u,v)\in E} \max(\pi(u),\pi(v)) = \sum_{v\in V} \pi(v) * ld(v)$: the left hand side calculates the cost by considering each edge and finds the larger position of its two nodes (or the position of the right endpoint), while the right side

hand counts the number of edges which are ended at each vertex and multiplied by the position of that vertex. The MAX heuristic takes O(m) to calculate the vertex degrees and O(n*log(n)) time to sort, where m is the number of edges and n is the number of nodes. Thus the MAX heuristic takes O(m+n*$log(n)$) time. The MAX-LD heuristic takes additional O(m+n) time to perform the left-degree check compared with MAX, thus its total complexity is still O(m+n*log(n)) (Gondhalekar, 1997). For the graph in Fig. 5-8, the ordering will be (the tie is broken by node ID again): [2,0,4,6, 1,3,7,8, 9, 5, 11,10]. Compared with the MAX heuristic result of [2, 6, 0, 4, 8, 1, 3, 7, 9, 5, 10, 11], node 8 is moved from the 5th to the 8th position. This is reasonable since node 8, although has a total larger out-degree, only has the out-degree of one to the nodes [2, 6, 0, 4] that have already been scheduled for broadcast.

One of the possible problems with this method is that, for two nodes v_1 and v_2, although when $ld(v_1)>ld(v_2)$ it is beneficial to exchange v_1 and v_2 under the sequence of …v_1v_2…, for the exchanged sequence …v_2v_1…, it is still possible to have $ld(v_2)>ld(v_1)$. It is unclear how to handle this case in the heuristic. In the above example, the problem happens when switching the node pairs (6,0), (6,4), (8,7) and (10,11). One solution might be to compute $\pi(v_1)*ld(v_1)+\pi(v_2)*ld(v_2)$ under both the sequences and choose the one that has the smaller value.

An extension of the MAX heuristic is to use the summation of the weights of edges that contain a node instead of the degree of the node (where the weight can be treated as a unit), i.e., the order of a node is determined by the summation of the weights of edges that contain it. We call this heuristic NODE-WEIGHT. In Fig.5-8,

the ordering based on the NODE-WEIGHT is [6,2,0,4,8,3,9,7,1,5,10,11]. Since the NODE-WEIGHT heuristic is also per-node based, similar to the MAX-LD heuristic, we can also develop the NODE-LEFT-WEIGHT heuristic. In Fig.5-8, the ordering based on the NODE-LEFT-WEIGHT is [6,0,4,2,8,9,3,7,1,5,11,10]. Since adding the weights up has the same complexity of counting degrees, NODE-WEIGHT and NODE-LEFT-WEIGHT also have the complexity of O(m+n*log(n))

Similar to the NODE-WEIGHT heuristic we propose the EDGE-WEIGHT heuristic. We first sort the edges according to their deceasing weights. The nodes in the edge that has the largest weight are placed at the beginning of the broadcast sequence. We then check the edge that has second largest weight and place its nodes that haven't been placed onto the broadcast sequence. This process is repeated until all the nodes in the graph are placed. In this heuristic, with m as the number of edges, it takes O(m*log(m)) time to sort weights of the edges, O(m) time to place the edges onto the broadcast channel, provided that time to check whether a node has already been in the placed node list is constant. Thus the total complexity of the heuristic is O(m*log(m)). In Fig.5-8, the ordering based on the EDGE-WEIGHT heuristic is [3, 9, 0, 4, 2, 6, 7, 8, 1, 5, 10, 11].

Note that all the heuristics in this section are only applicable to the multiplexing scheme (MUL). Some of them, such as NODE-WEIGHT and EDGE-WEIGHT, can be easily extended to hypergraphs too. Conceptually the weight-based heuristics are better than the degree-based heuristics since they take the weights into consideration.

5.6 Spanning Tree Ordering

(Liberatore, 2002) proposed to use the Maximum Spanning Tree (MST) heuristic for broadcast sequencing. The underlining philosophy is similar to the construction of index trees: placing nodes that have stronger relationships (larger weights) as close to each other as possible. Clearly a Maximum Spanning Tree can be generated using the algorithms for the Minimum Spanning Tree problem: we only need to replace the weight $w_{i,j}$ between node i and node j with W- $w_{i,j}$ where W is a constant that is larger than the maximum of $w_{i,j}$ for all $(i,j) \in E$. There are two popular algorithms to generate a minimum/maximum spanning tree: the Prim's algorithm and the Kruskal's algorithm (Cormen, 2001).

(Liberatore, 2002) adopted the Kruskal's method. Initially each node is treated as a singleton. At the beginning the algorithm places the nodes i and j next to each other if the edge (i,j) has the maximum weight. As the algorithm processes, if edge (i,j) has the largest weight among the remaining edges, it combines the ordering that contains node i and the ordering that contains node j. For the graph in Fig. 5-8, the process is as follows. The edge that has the largest weight is (3,9) with a weight of 6, the next three largest weight edges are (0,4), (2,6), (8,7) with all weights of 5. These four edges are not connected up to now. The largest weight edge among the remaining edges is (3,2) with a weight of 4 and thus edges (3,9) and (2,6) are combined since they contain the source and the target nodes of edge (3,2), respectively. Since edge (3,9) has larger weight than edge (2,6), the combined sequence will be (3,9,2,6). Similarly for the next largest weight edge (3,8), the

sequence (3,9,2,6) is combined with the edge (8,7) to form a sequence of (3,9,2,6,8,7). This process continues until all the nodes are sequenced and the final sequence is [3, 9, 2, 6, 8, 7, 0, 4, 5, 1, 11, 10].

Although the maximum spanning tree does not need to be explicitly generated for ordering, the process of combining previous orderings is binary and hierarchical. Thus a Binary Decomposition Tree (BDT) can be generated from the process and used further for optimization as shown in Chapter 6. In fact, we can see that there are two possible ways to combine two existing orderings: one can be put ahead of the other and vice versa. However, it is very likely that the generated BDT might be very unbalanced since balance is not considered in the MST algorithm, which will degenerate the complexity of the algorithm from quadratic to exponential. In the above example, the BDT is shown in Fig. 5-9(left). We rotate the tree to generate a balanced BDT as shown in Fig. 5-9(right) by first performing four zigs (a zig is a right rotation of a binary tree) on the whole tree followed by performing a zag (a zag is a left rotation of a binary tree) on the sub-tree rooted at (3,5).

The Prim's Minimum Spanning Tree algorithm is essentially identical to Dijkstra's algorithm for shortest paths (Weiss, 1997). At any point of the algorithm, there is a set of vertices that have already been in the tree. For each step, the algorithm finds a new vertex to add to the tree by choosing the edge that has the smallest (largest for Maximum Spanning Tree) weight among all edges of *(u,v)* where *u* is in the tree and *v* is not. The ordering of adding new vertices to the tree will be used as the broadcast sequence. Since every node can be the source, there could be as

many as *n* such orderings. We evaluate the MST orderings using both the Prim's algorithm and the Kruskal's algorithm in Chapter 7.

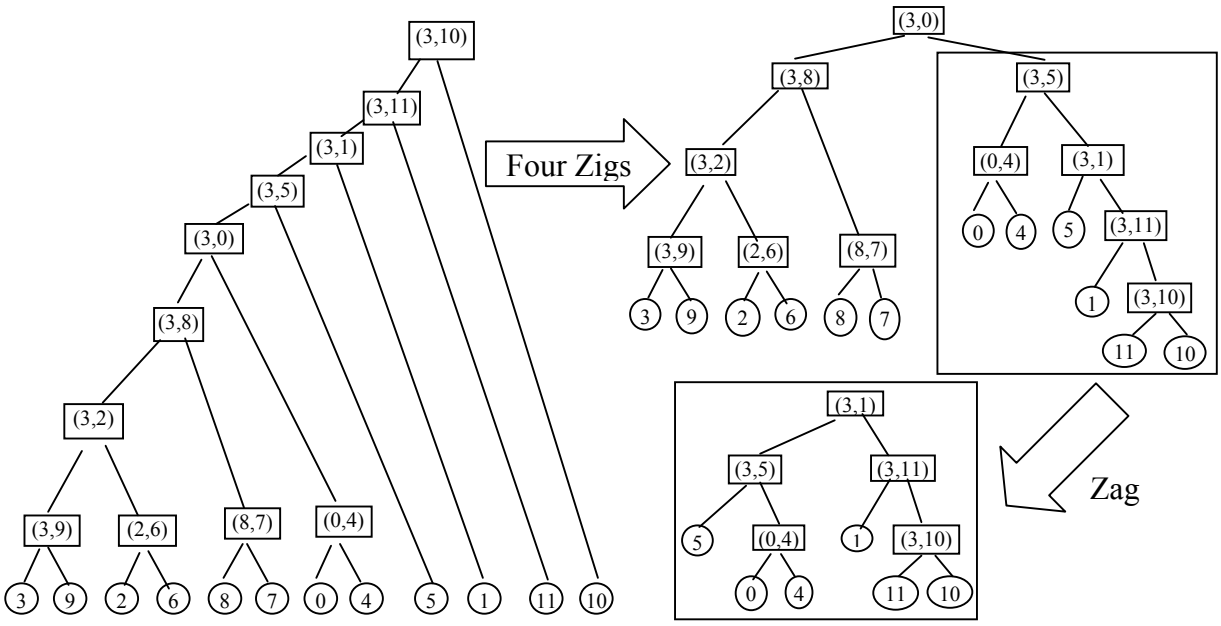


Fig. 5-9. The Binary Tree Generated From MST Ordering (Left) and its Balanced Tree After Rotations (Right) Using the Graph in Fig. 5-8

5.7 Discussions of Other Related Work

The QEM algorithm presented in (Chuang 2001) is essentially the extension of the hypergraph version of the EDGE-WEIGHT heuristic for the Separate Channel scheme. It begins with the hyperedge that has the largest weight and tries to append the nodes, which are on the hyperedge that has the next-largest weight but are not in previously placed node list yet, to both sides of the list and compare their resulting Query Distances (QD, c.f. Section 3.4 in Chapter 3). The one with smaller QD will be kept for further expansion and the one with larger QD will be discarded. A left/right append is defined as appending new nodes to the left/right of an existing sequence.

The left append will be used if there is a tie. The order of the unordered nodes in the previous list (nodes under ordering) will be split according to the newly appended nodes. An example shown in (Chuang 2001) is as follows. Suppose the first hyperedge contains unordered nodes [2,3,4,6] and the second hyperedge contains unordered nodes [3,4,5,7], then the left-append will be [5,7][3,4][2,6] and the right-append will be [2,6][3,4][5,7]. The unordered nodes of [2,3,4,6] are split into the two unordered set [3,4] and [2,6] since [2,6] is the intersection of the first node set and the second node set. This process continues until all the hyperedges are processed. This algorithm is greedy since the order of previously processed nodes cannot be changed.

This method was further extended in (Lee, 2003) by moving (reordering) nodes that have already been ordered to achieve less total QD. However, although the moving might benefit current expansion, it might increase the total QD for later expansions. Thus they proposed to use a weight threshold and any hyperedge whose weight is below this threshold will not be checked for moving. In addition, they proposed to check whether the summation of the frequencies of the remaining queries (i.e., the weights of the remaining hyperedges) that benefit from the moving is larger than the summation of the frequencies of the remaining queries which may be lost by applying the moving.

One problem with the QEM algorithm is its greedy expanding nature. In addition, it only considers the largest weight hyperedge for expansion without considering the hyperedges, which although have smaller weights individually, might have greater influences when combined. The modification in (Lee 2003) imposes

significant computation demands. The complexity of the QEM, and its extensions as well, is O(m*n) (Chung, 2001) where m is the number of queries (hyperedges) and n is the number of data items (nodes). When m is greater than n, which is the case in the hypergraphs based on spatial semantics of point data, the complexity is more than O(n^2).

(Chehadeh, 1999) proposed two heuristics: Approximate Linear Ordering for unit weight directed graph and Partial Linear Ordering for weighted directed graph. The Approximate Linear Ordering heuristic proposed to traverse the DAG using the principles of smaller out-degree first, DFS traversal and placing nodes immediately after their parents are placed. The first principle is somehow contradictory to the MAX heuristic as we discussed in Section 5.5. One explanation might be that the Approximate Linear Ordering is designed for directed graph with a special node that has in-degree of zero which is often serves as the first data item in a broadcast sequence. Using the smaller out-degree first principle will allow the queries that have a smaller number of data items to span less, i.e., have less access time. In conjunction with the DFS principle, the queries that have a larger number of data items will also span less. The reason is that these data items have less possibility to be interleaved by data items that have smaller out-degrees since they have already been placed onto the broadcast channel. The MAX heuristic, on the other hand, is designed for queries involved exactly two data items. Using the MAX heuristic to place node u, since it does not need to worry about placing nodes other than the immediate neighbors of u,

the heuristic is well justified since the cost of $\sum \max(\pi(u), \pi(i))$ will be minimized if we put node *i* as close to *u* as possible where *i* is the immediate node of *u*.

The Partial Linear Ordering heuristic takes a weighted directed graph as its input and produces a linear sequence. It iteratively combines nodes until all the nodes are combined and the sequence of combinations denotes an ordering. The order of the previous ordering and the node that is currently being combined is determined by the weighted distance $d_{i,j}$ which can be computed as follows:

$$d_{i,j} = \sum_{allEdges \quad i->j} \frac{w_{i,j}}{length(multi\_node_i) - order(single\_node_i) + order(single\_node_j)}$$

where *multi_node* is the previously combine sequence (denoted as node *i*) and length($multi\_node_i$) is the number of nodes within the *multi_node i*. The order(*single_node*) is the position of the *single_node* within the *multi_node*. If $d_{i,j}$ is larger than $d_{j,i}$ then the order will be node *i* followed by *j* and vice versa.

(Lee, 2003) extends this heuristic to undirected graph. The formula to compute $d_{u,v}$ is revised as follows:

$$d_{u,v} = \sum_{\substack{i \in u \\ j \in v}} \max\left(\frac{w_{i,j}}{length(u) - order(i) + order(j)}, \frac{w_{i,j}}{L - [length(u) - order(i) + order(j)]} \right)$$

where *L* is the broadcast cycle length, *i* and *j* are regular nodes and *u* and *v* are multi_nodes. In (Lee, 2003), the combining process always combines nodes *i* and *j* where edge (*i,j*) has the largest weight. However, at the very beginning of the algorithm where both node *i* and node *j* are single nodes, it is not clear how to determine the order of *i* and *j*. It is also worth to note that in the formula to calculate

$d_{u,v}$, u is always the multi-node while v is always the single node. Thus it always takes n-1 steps to finish the algorithm.

5.8 Further Discussions

Although it is intuitive to use geometry-based heuristics for point data and use graph-based heuristics for graph data, there are some other options. In Chapter 4, we proposed a hypergraph representation of spatial semantics for spatial range queries on point data. The representation allows us to use graph-based heuristics if they can be extended to hypergraphs, such as the NODE-WEIGHT and EDGE-WEIGHT as discussed in Section 5.5. Another option is to construct a Delaunay Triangulation (Aurenhammer, 1991) network as shown in Fig. 5-10 to convert a point data set into a graph data set and then we can use graph-based heuristics. The construction process generally has a complexity of O(n*log(n)) (Aurenhammer, 1991). On the other hand, since the graph data sets in this study are two-dimensional geometric graphs, we can use the geometry of their vertices as points and use geometry-based heuristics. We evaluate these heuristics based on our cost models by experiments using a real data set in Section 7.4 of Chapter 7.

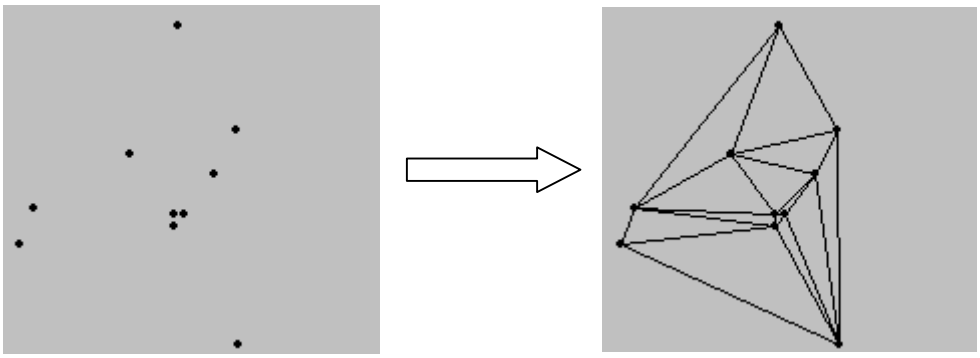


Fig. 5-10. Illustration of Constructing Delaunay Triangulation Network From a Point Data Set

In summary, among the heuristics we have discussed, we suggest to explore geometric-based heuristics, such as Hilbert SFC ordering and R-Tree traversal ordering if computation time is the primary concern. On the other hand, if ordering quality is the primary concern, ordering based on graph or hypergraph partition tree might be a good candidate. We should also take pre-existing spatial data structures, such as SFCs, R-Trees and Delaunay Triangulation Networks, into consideration for efficiency purposes.

Chapter 6

Optimization Methods

As discussed in Section 4.3 of Chapter 4, our problem of minimizing the total query cost is related to the graph Minimum Linear Arrangement problem (MinLA). The graph MinLA problem is a well-studied problem and several efficient approximation methods have been proposed (Bar-Yehuda, 2001; Koren, 2002).

By extending an edge of (u,v) to a hyperedge $\{n_1,n_2..n_k\}$ and defining the "edge length" of a hyperedge length as $L_2 = \max\{\pi(n_1),\pi(n_2),...\pi(n_k)\} - \min\{\pi(n_1),\pi(n_2),...\pi(n_k)\}$, the problem of minimizing DBW is essentially the same as the hypergraph MinLA problem. We want to use the existing efficient MinLA approximation methods to solve our geographical data broadcast sequencing problem. In this chapter we propose to use the low-polynomial cost approximation method presented in (Bar-Yehuda, 2001) to solve the DBW minimization problem. We then propose to use L_2 to approximate $g(L_2)$ (as defined in Section 3.1 of Chapter 3) to solve the AT_{Data}^{Sep} optimization problem. An novel approach is developed to optimize $AT_{Data}^{Mul.}$

For the rest of this chapter, we first briefly introduce the algorithm of (Bar-Yehuda, 2001) and we then prove the correctness of using this algorithm for hypergraphs. We show the importance of generating the Binary Decomposition Tree (BDT) (Bar-Yehuda, 2001) and propose to use R-Tree as the basis for generating a

BDT. The methods for optimizing DBW, AT_{Data}^{Sep} and AT_{Data}^{Mul} are presented in Section 6.4 through Section 6.6. For each of the three methods, DBW optimization, AT_{Data}^{Sep} optimization, AT_{Data}^{Sep} optimization, we illustrate its process through a simple example using the data set shown in Fig. 4-1 of Chapter 4.

6.1 The Approximation Algorithm

(Bar-Yehuda, 2001) proposed a divide and conquer method to approximately solve the graph MinLA problem in low-cost polynomial time. The space complexity of the proposed implementation is $O(2^{\text{depth}(T)})$ where T is the BDT (Fig. 6-1) of the graph. If T is balanced then space complexity is $O(n)$ where n is the number of nodes in a graph. For time complexity, if the out-degrees of the nodes in the graph are bounded by a constant, it is linear in $\sum_{t \in T} 2^{depth(t)}$ (Bar-Yehuda, 2001). This is quadratic if T is perfectly balanced and $O(n^{2.2})$ if T is 1/3-balanced (Bar-Yehuda, 2001).

The approximation algorithm proposed in the paper imposes a global ordering constraint on a hypergraph by using a BDT. A BDT T (Fig. 6-1) is a binary tree that has all the nodes in a hypergraph as its leaf nodes. For each tree $t \in T$ that has two sub-trees t_1 and t_2, there are two options in placing the nodes under it onto a broadcast channel, i.e., either the nodes of t_1 are placed ahead of the nodes of 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 the BDT. The orientation tree determines an ordering sequence of all the nodes in a graph.

Since t has two orientations and the orientations of its two sub-trees, t_1 and t_2, are independent of each other, it can be proved that there are 2^{n-1} orderings for a full and balanced BDT as shown in Fig. 6-2.

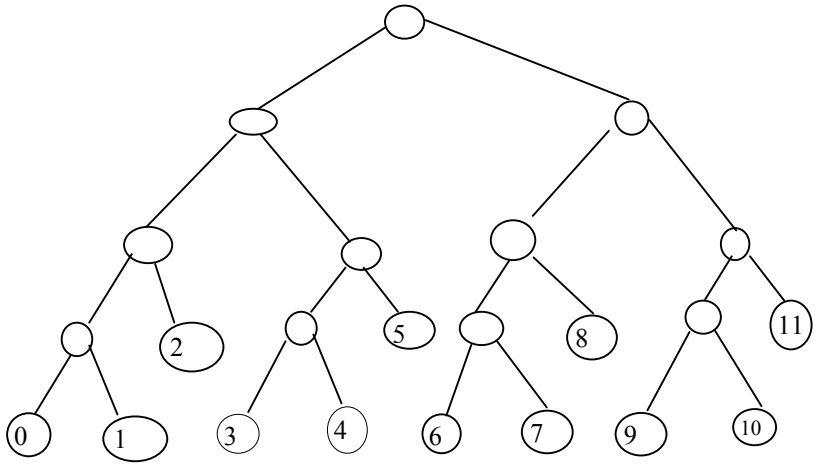


Fig. 6-1. Illustration of a Binary Decomposition Tree

Let n be the number of nodes in a graph whose BDT T is full and balanced, i.e., $n=2^k$. Let S(n) be the number of possible orderings of T We have $S(1)=1$

$$
\begin{aligned}
S(n) &= 2*S(n/2)*S(n/2) = 2*[S(n/2)]^2 \\
&= 2*[2*S(n/2^2)]^2 = 2*2^2*S(n/2^3) \\
&= \ldots\ldots \\
&= 2^{1+2+2^2+2^3+\ldots+2^{k-1}} S(1) \\
&= 2^{\frac{2^k-1}{2-1}} = 2^{2^k-1} = 2^{n-1}
\end{aligned}
$$

Fig. 6-2. Proof of the Number of Possible Orderings of a BDT

We next describe the approximation algorithm briefly. The Cost $_{L,V(t),R,\pi}$ with regards to a BDT sub-tree under ordering π is defined as

$$\text{Cost}\,[_{\text{L, V(t),R}}, \pi] = \sum_{(u,v)\in E} \begin{Bmatrix} w(u,v)*|\pi(u)-\pi(v)| & u \in V(t) \cap v \in V(t) \\ w(u,v)*\pi(u) & u \in V(t) \cap v \in L \\ w(u,v)*|V(t)|-\pi(u) & u \in V(t) \cap v \in R \\ 0 & otherwise \end{Bmatrix}$$

where $V(t)$ is the node set of t, L and R are the node sets that are to the left of $V(t)$ and to the right of $V(t)$, respectively. When t is the whole BDT, $L=R=\varnothing$, $\text{Cost}_{L,V(t),R,\pi}$ is exactly the $la(G)$. The costs under the two orientations can be efficiently computed as briefly described in the following.

Let *t be* a BDT node corresponding to the ordered partition which consists of L, $V(t)$ and R. Let t_1 and t_2 be the sub-trees of t. The left child of t is called *left* and the right child of t is called *right* under both orientations of t. Suppose that each child of the BDT is assigned a cost for both the 0-orientation (i.e., cost(*left*(0) and cost(*right*(0)) and 1-orientation (i.e., cost(*left*(1) and cost(*right*(1)). The cost of t under the two orientations are computed as follows:

$$\begin{aligned} \text{cost}_0 &= \text{cost}(left(0)) + \text{cost}(right(0)) + |V(t_2)|.\text{cost}(V(t_1),R) + |V(t_1)|.\text{cost}(L,V(t_2)) \\ \text{cost}_1 &= \text{cost}(left(1)) + \text{cost}(right(1)) + |V(t_1)|.\text{cost}(V(t_2),R) + |V(t_2)|.\text{cost}(L,V(t_1)) \end{aligned} \quad (1)$$

where cost(L,$V(t_1)$) and cost(L,$V(t_2)$) are called left outer cuts (or left_cut for short), cost($V(t_1)$,R) and cost($V(t_2)$,R) are called right outer cuts (or right_cut for short) and $|V(t)|$ denotes the number of leaf nodes of t. Both the left outer cuts and the right outer cuts can be computed recursively as follows.

Let $\hat{t}$ denote the orientation of the root node of t, left_cut($\hat{t}$) and right_cut($\hat{t}$) be the left outer cut (i.e., cost(L,$V(t_1)$) or cost(L,$V(t_2)$)) and the right outer cut (i.e.,

cost($V(t_1)$,R) or cost($V(t_2)$,R)) of $\hat{t}$, respectively. Let in_cut be the total cost of edges whose beginning node and ending node have t as the Least Common Ancestor (LCA). When $\hat{t}$ is a leaf node, the values of the outer cuts are computed by considering the edges incident to t. When $\hat{t}$ is an intermediate node the following formulas hold:

cost(left_cut($\hat{t}$))=cost(left_cut(left($\hat{t}$)))+cost(left_cut(right($\hat{t}$)))-cost(in_cut(t))

cost(right_cut($\hat{t}$))=cost(right_cut(left($\hat{t}$)))+cost(right_cut(right($\hat{t}$)))-cost(in_cut(t))

Also formula (1) can be rewritten as:

$$\begin{aligned} cost_0 &= cost(left(0)) + cost(right(0)) + |V(t_2)|.cost(right\_cut(\hat{t}_1)) + |V(t_1)|.cost(left\_cut(\hat{t}_2)) \\ cost_1 &= cost(left(1)) + cost(right(1)) + |V(t_1)|.cost(right\_cut(\hat{t}_2)) + |V(t_2)|.cost(left\_cut(\hat{t}_1)) \end{aligned} \quad (2)$$

As discussed earlier, the cost of t is the lower of the two costs, $cost_0$ and $cost_1$, and the orientation that has the lower cost will be set as the winner. The orientation that has less cost (access time in our case) will be adopted for t.

For illustration convenience, we use "+" to denote the 1-orientation and "-" to denote the 0-orientation hereafter. When the orientation of the root of tree T is determined, the ordering of all the data items can be determined based on the orientations of the nodes on the path from the root to the leaf nodes (data items). For example, in Fig. 6-3, the orientations along the path from the root to node 6 are "+ + - +" and the position of node 6 in the ordering is 4 (starting at 0).

Note that in_cut(t) is independent of the orientation of t and can be pre-computed after the BDT is built. Both left_cut ($\hat{t}$) and right_cut ($\hat{t}$) can be computed from t's two children under the same orientation by one addition and one subtraction.

Thus this algorithm is very efficient. We refer the readers to (Bar-Yehuda, 2001) for a detailed complexity proof.

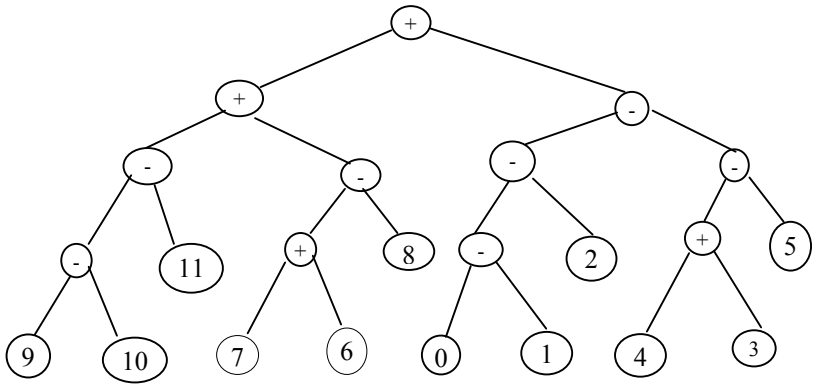


Fig. 6-3. An Orientation Tree Corresponding to the BDT in Fig. 6-1

6.2. Proof of Correctness for Hypergraph Case

The method given in (Bar-Yehuda, 2001) is only applicable to regular graphs. We next prove the method is also applicable to hypergraphs. For a hyperedge e of $\{n_1,n_2..n_k\}$, let u be the first node and v be the last node according to an ordering π. To prove the value computed by formula (2) equals the hypergraph version of $la(G)$ and, hence, DBW, it is sufficient to prove that for any hyperedge e, the cost computed by formula (2) equals $w(e)^* | \pi(u) - \pi(v) |$.

Let the least common ancestor of all nodes of e in the decomposition tree be t_0, then all the nodes involved in computing the costs regarding e are within t_0. Thus we do not need examining the nodes that are outside of t_0 in the proof. In Fig. 6-4., let the right-most node in t_0's left child ($\hat{t}_1$) be x and the left-most node in t_0's right child ($\hat{t}_2$) be y. Clearly $\pi(y) - \pi(x)=1$. Suppose the nodes on the path from t_1 to u are L_1,

…L_{k-2}, L_{k-1}, L_k, and the sizes of the right sub-trees of the trees having L_1, …L_{k-2}, L_{k-1}, L_k as the root nodes are p_1, …p_{k-2}, p_{k-1}, p_k, respectively, we have $p_1+p_2+\ldots+p_k = \pi(x)-\pi(u)$ since they are the number of the nodes between node u and node x. Note that L_1 is the root node of t_1.

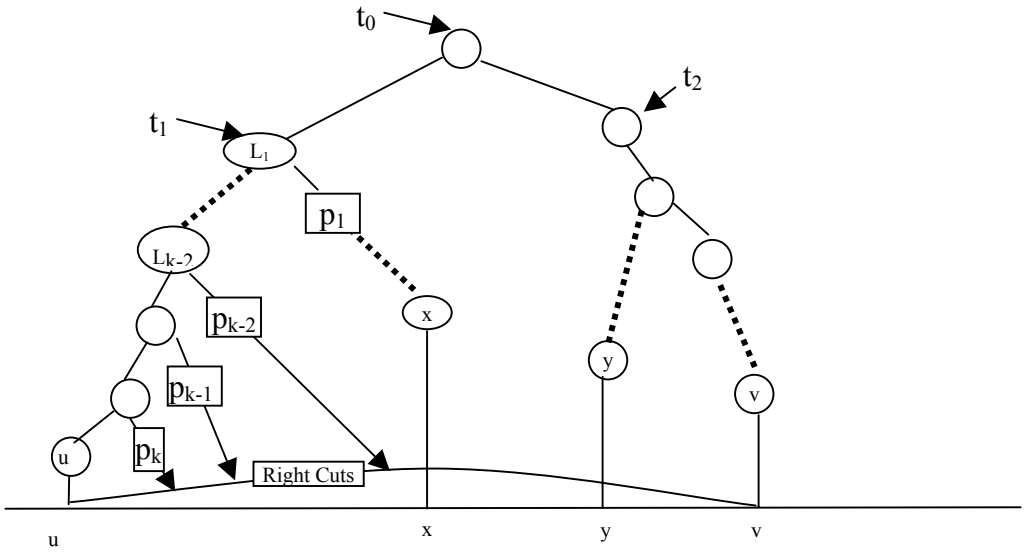


Fig. 6-4. The BDT Structure in an Ordering Sequence for a Hyperedge

Now let us expand formula (2) completely and examine which terms involve $w(e)$. According to the method used for deriving formula (2), $w(e)$ appears in the following cases: appear once as the tree cost when the sub-tree is a leaf node and labeled as v, appear in the right outer cuts from $\hat{t}_1$ (L) to $\hat{t}_2$ (R) and appear as the left outer cuts from $\hat{t}_2$ (R) to $\hat{t}_1$ (L). We examine the left outer cuts first.

Observe that only sub-trees that contain node u can contribute costs in terms of $w(e)$ to the left outer cuts of node v. In the generalized form of formula (2)

$$\text{cost}(\hat{t}) = \text{cost}(\hat{t}_1) + \text{cost}(\hat{t}_2) + |V(t_2)|.\text{cost}(\text{right\_cut}(\hat{t}_1)) + |V(t_1)|.\text{cost}(\text{left\_cut}(\hat{t}_2))$$

only cost($\hat{t}_1$) and cost(left_cut(v)) can contribute to the left outer cuts of node v. Cost($\hat{t}_1$) can contribute to the left outer cuts of node v because when it is computed recursively another level down, the left outer cuts of node v with regard to e will appear. Since we are only concerning costs with regard to e, left_cut($\hat{t}_2$) with regard to e is $w(e)$. Since $|V(t_1)| = L_1$ at root level of t_1, $|V(t_1)|.\text{cost}(\text{left\_cut}(\hat{t}_2)) = L_1 * w(e)$. Continue the recursion process till leaf node u is reached and the total left outer cuts with regard to e is as follows:

$$|L_k| * w(e) + |L_{k-1}| * w(e) + \ldots + |L_1| * w(e)$$

$$= p_k * w(e) + p_{k-1} * w(e) + \ldots + p_1 * w(e)$$

$$= (p_1 + p_2 + \ldots + p_k) * w(e)$$

$$= [\pi(x) - \pi(u)] * w(e)$$

Similarly we can prove that the right outer cut of $\hat{t}_1$ is $[\pi(v) - \pi(y)] * w(e)$. Recall that $w(e)$ will appear once as the tree (leaf node) cost, thus the total cost with respect to $w(e)$ is:

$$[\pi(x) - \pi(u)] * w(e) + [\pi(v) - \pi(y)] * w(e) + w(e)$$

$$= [\pi(v) - \pi(u)] * w(e) + [\pi(x) + 1 - \pi(y)] * w(e)$$

$$= [\pi(v) - \pi(u)] * w(e)$$

Since for any hyperedge e, the cost computed using formula (2) equals $w(e) * |\pi(u) - \pi(v)|$, we can claim that the total cost computed by formula (2) for a

hypergraph having the set of the hyperedges E is exactly the definition of $la(G)$ for a hypergraph, i.e.,

$$la(HG) = \sum_{e \in E} \max\{\pi(n_1), \pi(n_2), ...\pi(n_k)\} - \min\{\pi(n_1), \pi(n_2), ...\pi(n_k)\}$$

where $e=\{n_1, n_2, ...n_k\}$, which is the same as our cost model. Thus we can apply the algorithm proposed in (Bar-Yehuda, 2001) for optimizing broadcast sequencing.

6.3. Generating BDT

A BDT can be generated from an arbitrary ordering sequence. However, the possible number of orderings using a BDT is reduced from $n!$ to 2^{n-1} which means that some of the orderings are not possible under certain decompositions, thus the global optimal ordering might be missed. A good decomposition will lead to good ordering which is shown through the following simple example.

In Fig 6-5, for the example access graph, where the numbers inside rectangles are the weights of the corresponding edges, the four possible orderings of decomposition #1 is {{0,1,2}, {0,2,1}, {1,2,0}, {2,1,0}} and the corresponding total access time costs are {13,11,11,13}. The four possible orderings of decomposition #2 is {{1,2,0}, {1,0,2}, {2,0,1}, {0,2,1}} and the corresponding costs are {11,12,12,11}. The four possible orderings for decomposition #3 is {{2,1,0}, {2,0,1}, 1,0,2}, {0,1,2}} and the corresponding costs are {13,12,12,13}. Although it is possible for decomposition #1 and decomposition #2 to obtain the globally optimal solution, it is simply not possible for decomposition #3 due to its bad binary decomposition. Thus generating the initial BDT is very important to obtain good results.

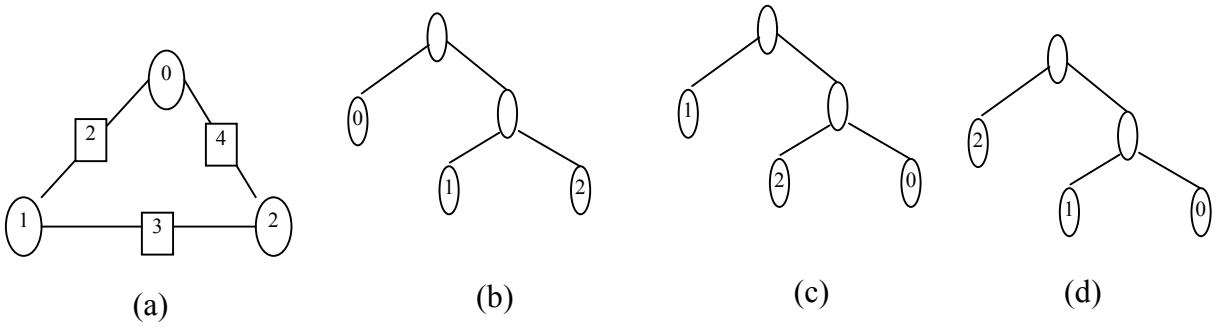
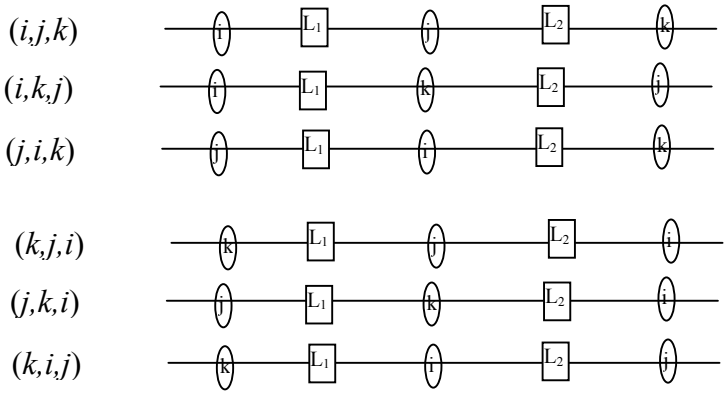


Fig. 6-5. (a) The Access Graph (b) Decomposition #1
(c) Decomposition #2 (d) Decomposition #3

In the example, the minimum cost (one of the orderings in decomposition #1 and #2) is obtained when node 0 and node 2 are placed next to each other. Thus to reduce the total access time, an intuitive idea would be to cluster data items that have larger edge weight into the same sub-tree. Actually we can prove the following general case.

For three data items *i, j* and *k*, without loss of generality, we assume their edge weights $w_{i,j} > w_{j,k} > w_{i,k}$, L_1 is the interval between the first and the second item and L_2 is the interval between the second and the third item, then the order of (*i,j,k*) or (*k,j,i*) has the smallest cost among all possible six orderings.

Proof:



$\text{Cost}(i,j,k) = w_{i,j} * L_1 + w_{j,k} * L_2 + w_{i,k} * (L_1 + L_2)$

$\text{Cost}(i,k,j) = w_{i,k} * L_1 + w_{k,j} * L_2 + w_{i,j} * (L_1 + L_2)$

$\text{Cost}(j,i,k) = w_{j,i} * L_1 + w_{i,k} * L_2 + w_{j,k} * (L_1 + L_2)$

Then

$\text{Cost}(i,j,k) - \text{Cost}(i,k,j) = - w_{i,j} * L_2 + w_{i,k} * L_2 = (w_{i,k} - w_{i,j}) * L_2 < 0$

$\text{Cost}(i,j,k) - \text{Cost}(j,i,k) = - w_{j,k} * L_1 + w_{i,k} * L_1 = (w_{i,k} - w_{j,k}) * L_1 < 0$

Thus the order of (i,j,k) has the smallest cost among (i,j,k), (i,k,j) and (j,i,k). Similarly,

$\text{Cost}(k,j,i) = w_{k,j} * L_1 + w_{j,i} * L_2 + w_{k,i} * (L_1 + L_2)$

$\text{Cost}(j,k,i) = w_{j,k} * L_1 + w_{k,i} * L_2 + w_{j,i} * (L_1 + L_2)$

$\text{Cost}(k,i,j) = w_{k,i} * L_1 + w_{i,j} * L_2 + w_{k,j} * (L_1 + L_2)$

Then

$\text{Cost}(k,j,i) - \text{Cost}(j,k,i) = - w_{j,i} * L_1 + w_{k,i} * L_1 = (w_{k,i} - w_{j,i}) * L_1 < 0$

$\text{Cost}(k,j,i) - \text{Cost}(k,i,j) = - w_{k,j} * L_2 + w_{k,i} * L_2 = (w_{k,i} - w_{k,j}) * L_1 < 0$

Thus the order of (k,j,i) has the smallest cost among (k,j,i), (j,k,i) and (k,i,j). On the other hand,

$\text{Cost}(i,j,k) - \text{Cost}(k,j,i) = w_{i,j} * (L_1 - L_2) - w_{j,k} * (L_1 - L_2) = (w_{i,j} - w_{j,k}) * (L_1 - L_2)$

The relationship between $\text{Cost}(i,j,k)$ and $\text{Cost}(k,j,i)$ also depends on the relationship between L_1 and L_2. Nevertheless we can draw the conclusion that the order of (i,j,k) or (k,j,i) has the smallest cost among all six possible orderings. In either case, data items i and j are placed next to each other.

Spatial index trees, such as the R-Tree (Guttman, 1984), R+-Tree (Sellis, 1987) and R*-Tree (Beckmann, 1990), are designed to put data items spatially close to each other into the same branch while put data items spatially far away from each other into different branches. Based on our hypergraph representation, the extended regions of the points represented by the nodes of a hyperedge generally have a larger portion of overlap in the case where these nodes are from the same branch than in the case where these nodes are from different branches. In other words, the weight of a hyperedge whose nodes are from the same branch is generally larger than the weight of a hyperedge whose nodes are from different branches in spatial range queries. Thus tree-based spatial index methods are good candidates to generate a BDT for point data. We use R-Tree to generate a BDT in this study due to its popularity in spatial databases for geographical data. We replace an *m*-branches R-tree node with a small binary tree and connect all such small binary trees to build the BDT. An illustration is given in Fig. 6-6.

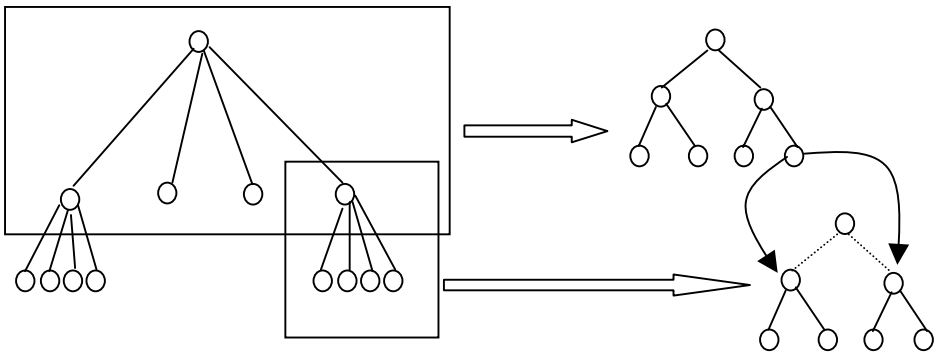


Fig. 6-6. Replacing an R-Tree Node by a BDT Sub-Tree

```
Algorithm RTreeToBDT
Input:
        r_root: the root of an R-Tree
        m: the number of the children of r_root
Output:
        b_root: the root of the built BDT

        Put the children of r_root in array seq
        b_root=GenBDT(seq,0,m)

Algorithm GenBDT
Input:
        seq: An array of the children node of an R-Tree
        first: The beginning position to build a BDT in seq
        last: The ending position to build a BDT in seq
Output:
        root: The pointer to the root of the BDT being built

        Allocate memory for root
        If first equals last
                If(seq[first]) is a non-leaf R-Tree node
                        Mark root as the intermediate node
                        Let new_root be the pointer of a BDT node
                        Let num be the number of children of seq[first]
                        new_root =RTreeToBDT(seq[first],num)
                        Set the first child of root to the first child of new_root
                        Set the second child of root to the second child of new_root
                else
                        Set the ID of root to the ID of seq[first]
                        Set the two children of root to NULL
        Else
                Mark root as the intermediate node
                middle=(first+last)/2
                Set the first child of root to the result of GenBDT(seq, first, middle)
                Set the second child of root to the result of GenBDT(seq, middle+1,last)
        Return(root)
```

Fig. 6-7. The Process of Generating a BDT From an R-Tree

The process of generating an BDT from an R-Tree is presented in Fig. 6-7. We begin with the root of the R-Tree and divide the immediate nodes of the root into two groups recursively to build a small binary tree. The root of the small binary tree will be the root of the BDT. This process is performed recursively until the leaf nodes of the R-Tree are reached. Since the algorithm runs in a divide and conquer manner

and each R-Tree node is processed exactly once, we claim that the complexity of the algorithm is linear with respect to the number of nodes in the R-Tree. The proof is similar to the proof of linearity of the tree traversal problem as shown in (Cormen, 2001).

6.4 Optimizing DBW

Due to the similarity between DBW and $la(G)$ as we discussed in Section 4.3 of Chapter 4, we can use the MinLA method proposed in (Bar-Yehuda, 2001). Although the paper only handles the graph MinLA problem, its implementation can handle the hypergraph MinLA problem, however, with two restrictions. The first restriction is that it requires that there are at least two nodes in a hyperedge while hyperedges (query result sets) in our representation might only include one node (either a point in a point data set or a vertex in a graph data set). The second restriction is that the implementation assumes that all hyperedges have unit weight. Since the DBW for accessing a single data item is always 0, the first restriction is not a problem. We also modify the implementation to allow hyperedges to have different weights.

In the implementation, for each of the node in the BDT, there is a pointer to its parent and two pointers to its two children. An orientation flag is also associated with each of the node in the BDT. The parent pointer of the root of BDT is empty and the two children pointers of a leaf node of BDT are also empty. We next briefly introduce the Least Common Ancestor Tree (LCA-Tree) data structure used in the

implementation to efficiently determine whether a node is the beginning/ending node of a hyperedge.

The LCA-Tree is constructed during preprocessing. An auxiliary array is needed in the construction. The pointers in the array map hypergraph nodes to the corresponding leaf nodes in the BDT of the hyper graph. For each of the nodes in a hyperedge, the corresponding BDT node is first retrieved and the path from the node all the way to the root of the BDT is travelled. The ID of the hyperedge is assigned as the flag of all the intermediate nodes on the paths. Next, starting from the root of the BDT, the implementation first tries to find a node of the BDT whose both children's flags have the value of ID of the hyperedge. In case only one child whose flag has the value of ID of the hyperedge, the implementation follows the child until a node whose both children's flags have the value of ID of the hyperedge is reached. This process is performed recursively until the leaf nodes of the BDT is reached. The implementation adds a node in the LCA tree in both cases, i.e., either a BDT node whose both children's flags are the assigned ID or it is a leaf BDT node. Fig. 6-8 shows the process, where the ID represents the edge number of the hyperedge of {1,4,5,6} and the dashed lines shows the correspondences between the nodes in the BDT and the LCA tree.

For a hyperedge with k nodes, it takes at most $log(n)$ for each of them to reach the root of the BDT where n is the number of the nodes in the hypergraph and the BDT. From the root of the BDT, it takes at most $log(n)$ to reach each of the nodes at the leaves of the BDT. Thus the time complexity of constructing a LCA tree is

O(*k***log*(*n*)). Since we assume *k* is bounded by a constant, thus the total time complexity for constructing LCA trees for the *m* hyperedges is O(*m***log*(*n*)). The space complexity of a LCA tree is in the order of O(*k*) , thus the space complexity for constructing the LCA trees is O(*m*), provided that *k* is bounded by a constant.

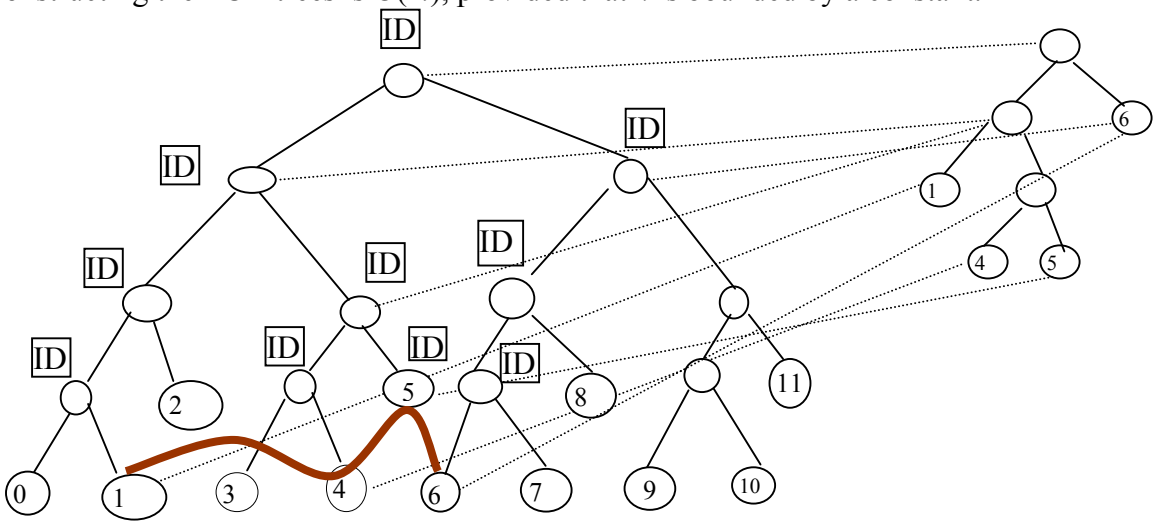


Fig. 6-8. Determining the Beginning/Ending Node of a Hyperedge

By using the LCA tree, whether a leaf node of the BDT is the beginning/ending node of a hyperedge can be determined efficiently. First start with the root of the LCA tree of the node and then follow the left/right child (depending on the orientation of the sub-tree rooted at the node) until we reach a leaf node of the LCA tree. Since the nodes of a LCA tree is a subset of the nodes of the BDT tree, the ID of the LCA leaf node and the ID of the BDT leaf node can be compared and decision can be made. The cost of the traversal from the root to a leaf node of a LCA tree is in the order of *log*(*k*) where *k* is the number of nodes in the corresponding hyperedge. Since we assume the maximum number of the nodes in a hyperedge of the

hypergraph is bounded by a constant, thus the determination can be made in small constant time.

We use the example data set shown in Fig. 4-1 to illustrate the optimization method for broadcast sequencing. The data set has 4 nodes and 11 hyperedges. The hyperedges and their weights are listed in Table 6-1. This simple data set will also be used to illustrate the other two proposed optimization methods subsequently.

Table 6-1. The Hyperedges and Their Weights for the Data in Fig. 4-1

Hyperedge Nodes	Weight
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

We first remove the 4 hyperedges each of which has only one single node as discussed above. We then build an R-Tree with a branch factor of 3. The resulting R-tree has two leaf nodes in each of its two sub-trees as shown in Fig. 6-9 and we use it as the BDT. Traversal of the R-Tree gives an ordering of [1,2,4,3] and we use it as our initial ordering. Among the 7 hyperedges, edge {1,2} rooted at T_{11} with in_cut of 22, edge {3,4} rooted at T_{12} with in_cut of 8 and the rest rooted at T_0 with their total in_cuts being the summation of the following values: 2 for edge {1,3}, 38 for edge

{2,3}, 3 for edge {2,4}, 14 for edge {1,2,3}and 4 for edge {2,3,4}, Thus the total inner_cut is 61.

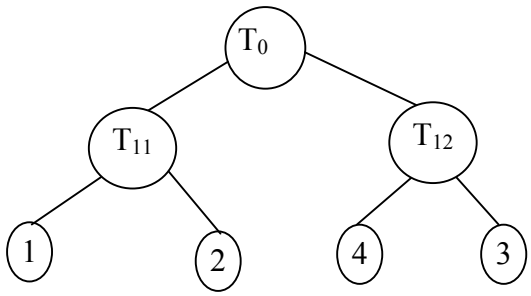
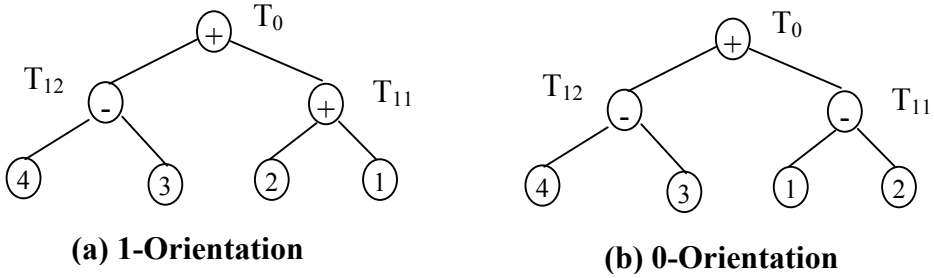


Fig. 6-9. The BDT of the Example for Illustrating DBW Optimization



(a) 1-Orientation **(b) 0-Orientation**

Fig. 6-10. The Orientation Trees of Two Possible Orientations of T_{11}

For the orientation tree in Fig. 6-10(a), the ordering is [4, 3, 2, 1]. Node 2 is the ending node of edges {2,3}, {2,4} and {2,3,4}, thus the left_cut of node 2 is 38+3+4=45. Node 2 is also the beginning node of edge {1,2} and thus its right_cut is 22. Similarly, the left_cut of node 1 is 2+14+22=38 and the right_cut of node 1 is 0. Since nodes 1 and 2 are leaf nodes, their costs are the same as their left_cuts which are 45 and 38, respectively. Thus the left_cut and the right_cut of their parents, T_{11}, are 45+38-22=61 and 22+0-22=0, respectively. The total cost of T_{11} under the current 1-orientation can be computed as 45+38+(22-22)*1+(38-22)*1=99. If the orientation of T_{11} is switched to 0-orientation, we can get the left_cut of node 1 as 2, the right_cut

of node 1 as 22, the left_cut of node 2 as 81 and the right_cut of node 2 as 0, thus the left_cut and the right_cut of T_{11} under the current 0-orientation are 2+81-22=61 and 22+0-22=0, respectively. The total cost of T_{11} is 2+81+(22-22)*1+(81-22)*1=142. Since the 1-orientation of T_{11} has the smaller cost (99) than the 0-orientation (142) of T_{11}, we set the 0-orientation to T_{11}. Similarly, we set the 0-orientation to T_{12} since its 0-orientation cost (15) is smaller than its 1-orientation cost (66) as shown in Fig. 6-10 (b). The left_cut and the right_cut under the 0-orientation of T_{12} are 0 and 61, respectively. Thus the total cost of T_0 is 99+15+(61-61)*2+(61-61)*2=114. This is the global optimal value of all possible 4!=24 orderings. The cost of the optimized ordering [1,2,3,4] is 44.1% better than the initial ordering {1,2,4,3} whose cost is 165.

To examine how the BDT affects the best optimization we can achieve, we put nodes 2 and 4 in a branch and nodes 1 and 3 in another branch of the BDT to use. Although the cost of the initial ordering [2,4,1,3] has the worst cost (233) among the possible 24 orderings, the cost of the optimized ordering [4,2,3,1], which is 139, is 67.6% better. However, it is still 21.9% worse than the optimized ordering using R-tree traversal as the initial ordering. This simple example demonstrates the effectiveness of both the proposed optimization method and the proposed heuristics of generating BDT from R-Tree and generating initial ordering using R-Tree traversal.

6.5 Optimizing AT_{Data}^{Sep}

Recall the cost model of AT_{Data}^{Sep} as we have derived in Section 3.2 in Chapter 3.

$$
\begin{aligned}
& AT_{Data}^{\ Sep} = \\
& \sum_{1 \le i < j \le n} w_{i,j} * g(|\pi(i) - \pi(j)|) \\
& + \sum_{1 \le i < j \le k \le} w_{i,j,k} * g(\max(\pi(i), \pi(j), \pi(k)) - \min\{\pi(i), \pi(j), \pi(k))) \\
& + \ldots \\
& + w_{1,2,\ldots n} * g(\max(\pi(1), \pi(2) \ldots \pi(n))) - \min(\pi(1), \pi(2) \ldots \pi(n)))
\end{aligned}
$$

For the function $g(L_2) = \frac{L}{2} + L_2 - \frac{L_2^{\ 2}}{2L}$ as defined in Section 3.1 in Chapter 3, we observe that the cost model for a single query in terms of AT_{Data}^{Sep} (DPW+DBW), i.e., $g(L_2)$, increases monotonically as DBW, i.e., L_2, increases and vice versa. Thus L_2, which is the hypergraph version of "edge length", is a good linear approximation of $g(L_2)$. By doing so we are expecting that the optimized ordering where the optimization is based on the definition of $la(G)$, which is linear with respect to L_2, is also a good ordering according to $g(L_2)$.

To compare the goodness of the approximation, we enumerate all possible 4!=24 orderings and compute both the linear cost and quadratic cost as shown in Fig. 6-11. They have the same trends which supports our theoretical result. The access

time under the quadratic model is always larger than the linear model as expected since the former includes both DPW and DBW while the latter only includes DBW.

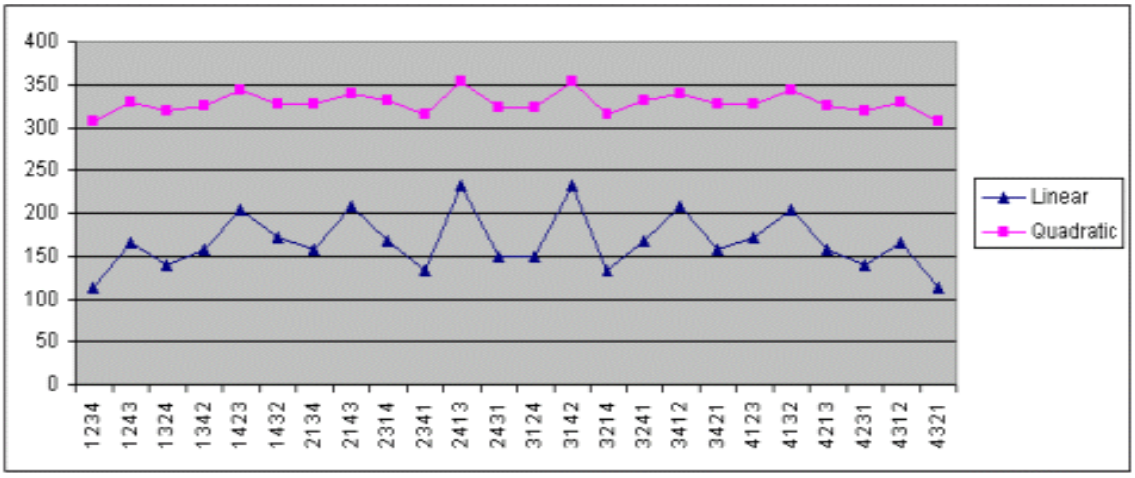


Fig. 6-11. Comparison of Access Time of Linear Versus Quadratic Models

6.6 Optimizing AT_{Data}^{Mul}

Recall the cost model of AT_{Data}^{Mul} as we have derived in Section 3.2 in Chapter3.

$$
\begin{aligned}
& AT_{Data}^{\ Mul} = \\
& = \sum_{1 \le i \le n} w_i * \pi(i) \\
& + \sum_{1 \le i < j \le n} w_{i,j} * \max(\pi(i), \pi(j)) \\
& + \sum_{1 \le i < j \le k \le n} w_{i,j,k} * \max(\pi(i), \pi(j), \pi(k)) \\
& + \ldots \\
& + w_{1,2,\ldots n} * \max(\pi(1), \pi(2) \ldots \pi(n))
\end{aligned}
$$

By relating $w(u,v)$ with w_i, $w_{i,j}...w_{1,2...n}$ and relating $|\pi(u)-\pi(v)|$ 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. Rather than computing $|\pi(u)-\pi(v)|$ for an edge directly, (Bar-Yehuda, 2001) computes it as a serial summations of the sub-tree sizes of the BDT to achieve its efficiency. Unfortunately, this is not possible to compute $\max\{\pi(n_1),\pi(n_2),...\pi(n_k)\}$ in a similar manor for AT_{Data}^{Mul} optimization due to its nonlinearity. In this study, we adopt the divide-and-conquer strategy and propose a new method for AT_{Data}^{Mul} optimization. Like (Bar-Yehuda, 2001), the method checks all possible 2^{n-1} orderings that can be derived from a BDT in $O(n^2)$ time complexity.

The process of the proposed method is shown in Fig. 6-12. The hypergraph data structures described in Section 4.4 of Chapter 4, the BDT enhancements and the LCA tree structure described in Section 6.4 of this chapter are also needed in the method. In addition, for each BDT node, we also compute the size of the sub-trees of the BDT rooted at the node.

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 tree *t* to the root of the BDT. Do the following recursively.
2 If *t* is an intermediate node of the BDT:
   a) Test the two orientations of its sub-trees t_1 and t_2 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 the 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. 6-12. The Process of Optimising AT_{Data}^{Mul}

Before illustrating the method with a simple example and performing complexity analysis, we next show how to compute the position of a node in an ordering efficiently which is crucial in the proposed method. As shown in Fig. 6-13, start at a leaf node (node 6 in our 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. 6-13, 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 value. Finally since T_3 is the left child of T_4, which is the root of the BDT, we skip again. Thus 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 where n is the number of nodes in the hypergraph and the BDT.

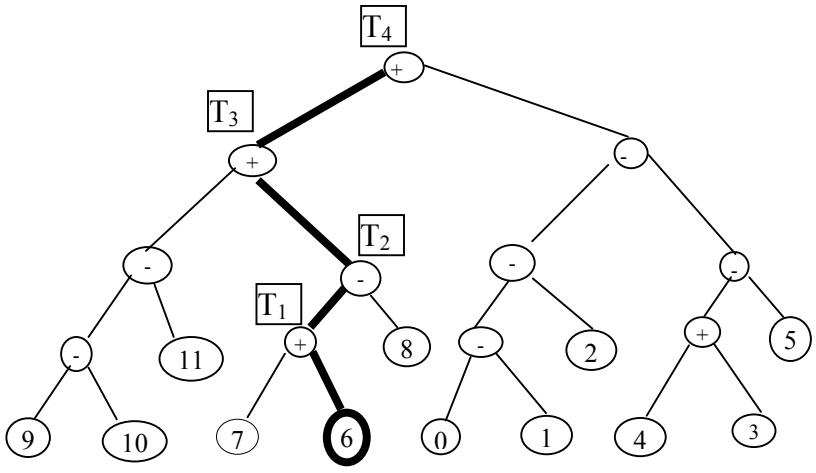


Fig. 6-13. Illustration of Computing Position of a BDT Node

We use the same example in Section 6.4 to illustrate the proposed AT_{Data}^{Mul} optimization method. The BDT we use is shown in Fig 6-14.

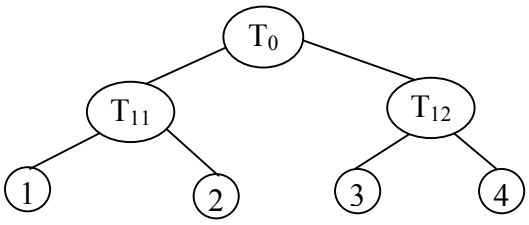


Fig. 6-14. The BDT of the Example for Illustrating AT_{Data}^{Mul} Optimization

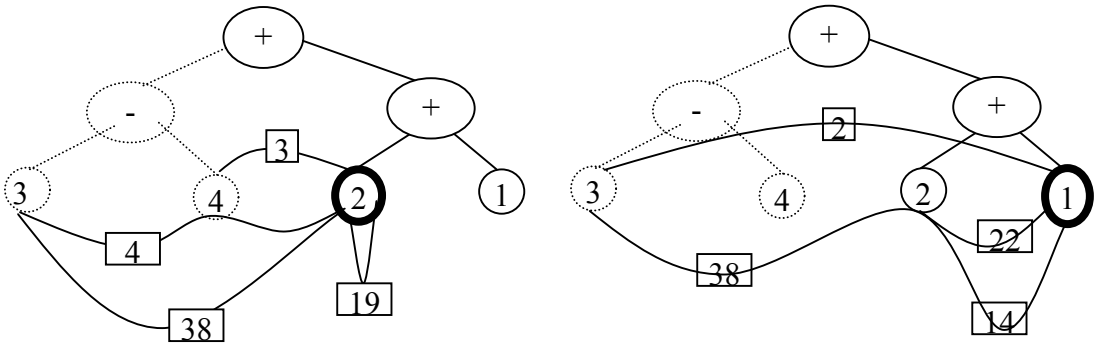


Fig. 6-15. Computing AT_{Data}^{Mul} for Nodes 1 and 2 Under 1-Orientation of T_{11}

We use $AT(x)$ to denote AT_{Data}^{Mul} of x, where x can be either a leaf node or a sub-tree of the BDT. If x is a sub-tree, we also denote its orientation by putting 0 or 1 in its top-right part. For T_{11} in the 1-orientation, as shown in Fig. 6-15, 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 choose the 0- orientation if the two orientations have the same cost. We next compute the cost of T_{12} under both orientations.

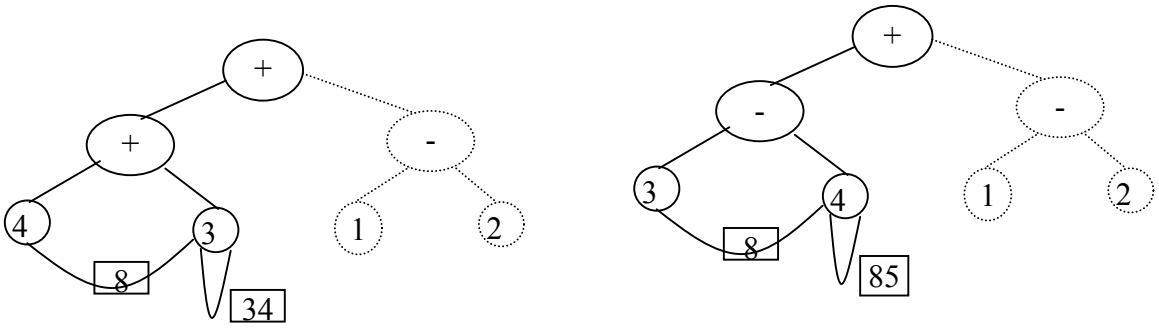


Fig. 6-16. Computing AT(T_{12}) Under 1- and 0-Orientations

For the T_{12} in the 1-orientation, as shown in Fig 6-16, 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 the 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, the 0-orientaiotn of T_{12} is the winner. Thus the total cost of T_0 under the 1-orientation is $AT(T_0^{1})= AT(T_{11}^{1})+ AT(T_{12}^{0}) = 428+42= 470$. Similarly we can compute the $AT(T_0)$ under the 0-orientation as 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].

Let the computation cost of sequencing an n-node hypergraph be $S(n)$. At each $t \in T$ having n nodes (i.e. $n=|t|$) we need to calculate the costs of its two children under two orientations which results 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 in Fig. 6-17.

$$
\begin{aligned}
S(n) &= 4 * S\left(\frac{n}{2}\right) + 3 \\
&= 4 * \left[4 * S\left(\frac{n}{2^2}\right) + 3 \right] + n + 3 \\
&= 4^2 * S\left(\frac{n}{2^2}\right) + 4 * 3 + 3 \\
&= \ldots \\
&= 4^k \left[S\left(\frac{n}{2^k}\right) + 3 \right] + \ldots + 4 * 3 + 3 \\
&= 4^k * S\left(\frac{n}{2^k}\right) + \ldots + 4^k * 3 + 4 * 3 + 3 \\
&= 4^k * S(1) + (4^k + 4^{k-1} + \ldots 1) * 3 \\
&= (2^k)^2 * S(1) + \frac{4^{k+1} - 1}{4 - 1} * 3 \\
&= n^2 * S(1) + 4 * n^2 - 1
\end{aligned}
$$

Fig. 6-17. Complexity Analysis of AT_{Data}^{Mul} Optimization Method

$S(1)$ has the following components. It takes $O(log(n))$ to compute the position of a node in Step 3.b. It takes constant time to retrieve all the hyperedges that contain the node by using the inverse hypergraph in Step 3.c. We assume the number of hyperedges that contain a node is bounded by a constant, so it takes constant time to determine whether a node is the ending node of a hyperedge, thus the total cost in Step 3.d is also bounded by a constant. Furthermore, in real applications, for a reasonably big number n (e.g., from 100 to 10000), $log(n)$ (7-14) is less than the multiplication of the average number of hyperedges that contain a node (e.g. 10-20) and the average depth of the LCA tree (e.g. 3-5) and we can treat $log(n)$ as a bounded constant for practical values of n. Thus the proposed method approximately has a complexity of $O(n^2)$.

Chapter 7

Experiments and Evaluations

7.1 Experiment Software Modules

We use several publicly available packages to make our experiments possible. They are the Boost Graph Library (BGL) at the Indiana University ([HREF 5]), the binary version H-Metis graph partition package from George Karypis at the University of Minnesota([HREF 6]), the Java version R-Tree package from Marios Hadjieleftheriou at the University of California, Riverside ([HREF 7]) and the C version Hilbert SFC package from Doug Moore at the Rice University ([HREF 8]). The software modules we developed are: Java version hypergraph generation package for spatial point data, the C version implementation of Floyd-Warshall algorithm for generating all pair shortest paths and its corresponding hypergraph, the C version EAFG generation module, the C version R-Tree to BDT conversion module and the C++ versions of MAX/MAX-LD heuristics for regular graphs, NODE-WEIGHT/EDGE-WEIGHT heuristics for regular graphs and NODE-WEIGHT/EDGE-WEIGHT heuristics for hypergraphs. The overall data flow of the experiments are shown in Fig. 7-1. The dashed line from "Graph Data Set" to "Point Data Set" means we use the geometries of the graph data set and treat them as a point data set. The modules are combined differently when applied to different data sets which will be described in detail in Section 7.2 and Sections 7.3 through 7.5 when experiments are performed on these data sets.

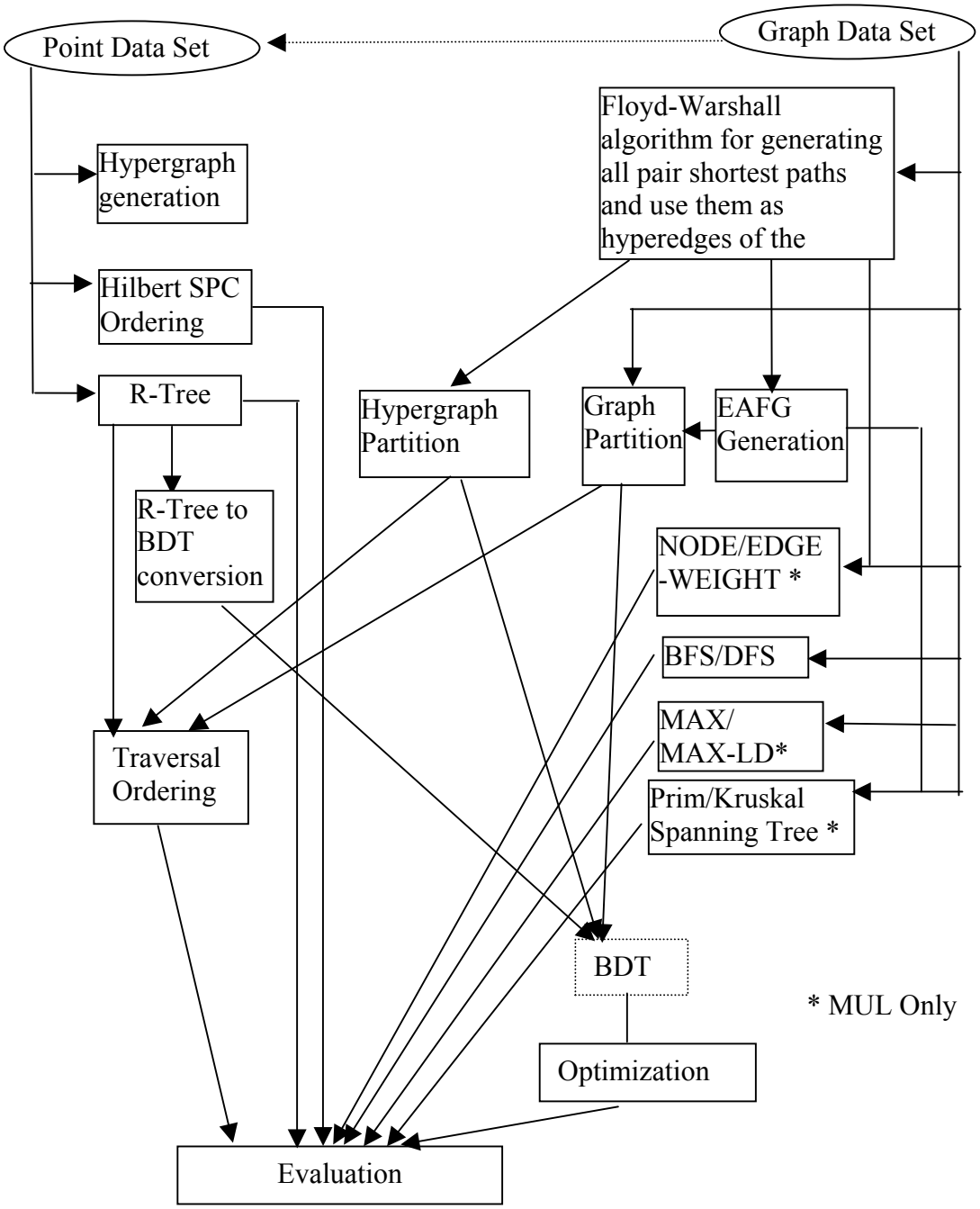


Fig. 7-1. Overall Data Flow of the Experiments

7.2 Data Sets and Performance Metrics

The majority of the data sets we use in this study are point data sets due to availability reasons. We first generate five random point data sets. The real data sets we use are the centers of the zip codes among the 50 states and the District of Columbia of the United States. The graph data set we use is the transportation network of the State of Texas manually input from an AAA map.

We start with the synthetic data sets and use a single query window. The small volumes of the data sets allow us listing all the experiment results for analysis under the three cost models individually. The 51 real data sets vary from a great scope of the covered areas, number of points, densities and distributions. Due to the huge volume of the results on the data sets under the five query window sizes and three cost models, the analysis is performed based on the average results while the detailed results are listed in the appendix. In addition to evaluating the ordering qualities, we also evaluate the computation time for running the optimization algorithms on the real data sets.

The experiments on the transportation network are designed to compare the geometric-based heuristics and graph-based heuristics for network path queries on graph data as discussed in Chapter 5. Besides the graph-based heuristics, the geometric-based heuristics are also used for network path queries. All the heuristic orderings and their optimized orderings of the graph data set for network path queries are also evaluated on the hypergraph representation of spatial range queries for comparison purposes.

For each of the point data sets, we first generate the hypergraph representations for the given query windows. To comply with the assumption in the optimization methods presented in Chapter 6 which states that the number of nodes in hyperedges is bounded by a constant, we discard the points (nodes) whose extended region contains more than a certain number of points. We also remove the points that only fall into their own extended regions, i.e., hyperedges that have only one node, for optimizations of DBW and AT_{Data}^{Sep} while keeping them as they are for optimizations of AT_{Data}^{Mul}.

For the graph data set, i.e., the transportation network of the State of Texas, due to lack of access frequency information, we use all pair shortest paths of the network as query sets and they have unit weights. Although we illustrate the optimization methods in Chapter 6 using a point data set, they can be applied to the optimization of graph data without any modification due to the reasons discussed in Section 3.3 of Chapter 3. Because of the monotonic relationship between DBW and AT_{Data}^{Mul} for a single complex query, and more importantly, due to the fact that DBW is more important than DPW as discussed in Chapter 3, we focus on DBW for the experiments on the graph data. We also perform experiments on graph data for the heuristics that are applicable to only MUL scheme. The results of these heuristic orderings are compared with the results of the optimized orderings under MUL scheme.

Our experiments are all done on a Dell Dimension 4100 personal computer with 866 MHZ Intel processor and 512M memory under the Windows 2000 professional operating system.

Our cost models presented in Chapter 3 measure the access time in terms of weight*time where weight is the access frequency of a query and the time is the data access time to broadcast channel to retrieve all the data items in the query result set. To make the experiment results more intuitive and comparable to each other, we define a new measurement called Normalized Data Access Time (NAT). Let AT_{Data} (DBW, AT_{Data}^{Sep} or AT_{Data}^{Mul}) be the access time to data according to the three cost models. Let W be the summation of the weights of the hyperedges of a hypergraph and L be the length of a broadcast cycle (i.e., number of data items to broadcast), the measurement is defined as $NAT = \frac{AT_{Data}}{W}$

The NAT value will be greater than 0 and less than L. Generally speaking, the larger the average number of nodes in the hyperedges of a hypergraph, the larger NAT. This is because a larger access time is needed to access all the data items even if they are next to each other in a broadcast sequence. We could have defined another measurement NAT/L, which will be between 0 and 1. However, it will be very small for hypergraphs that have a large number of nodes, i.e., larger broadcast cycle length. This is because we assume the average number of nodes per hyperedge is bounded by a constant. The definition of NAT is more suitable than NAT/L since it eliminates weighting factor while still having the capability to tell the ordering quality when used in conjunction with L.

7.3 Synthetic Data Set

We use five synthetic data sets in our experiments with sizes of 100,200, 300, 400 and 500 points respectively. They all have a data space of $[0,1) \times [0,1)$ and we use a query window size of 0.1 by 0.1. The points in the data sets are generated randomly within the data space and with the following restrictions:

First, the extended region of a point intersects with no more than N other Extended Regions. This is to ensure that the lengths of the hyperedges are bounded by the constant N to be complied with the requirement of the optimization methods. We choose N to be 10 in the experiments.

Second, the distances between a point and the points that fall into its extended region are no less than 1% of the radius of the query window (0.0005 in our experiments). This is to prevent from generating very tiny intersected regions to ensure that the weights of hyperedges is not too small to be meaningful for optimization.

Finally we remove the points whose extended region does not intersect with any other extended regions since they do not contribute to orderings. This might make the sizes of some data sets slightly less than their original size. For example, data sets 1, 3, 4 and 5 listed in Table 7-1 where their number of points are less than 100, 300, 400 and 500, respectively. Table 7-1 shows the parameters of the five data sets. It can be observed that as the number of points increases, both the number of hyperedges and the average nodes per hyperedge increase.

Table 7-1. Parameters of the Synthetic Data Sets

Data Set	# of Points	# of Hyperedges	Total # of Nodes in All Hyperedges	Average Nodes Per Hyperedge
1	96	253	667	2.64
2	200	1054	3393	3.22
3	294	1796	6358	3.54
4	382	2111	8854	4.19
5	452	2147	10802	5.03

7.3.1 Experiments Using The DBW Cost Model

We compare six orderings for each of the five data sets: the minimum of 1000 random orderings, the maximum of 1000 random orderings, the average of 1000 random orderings, the Hilbert SFC ordering, the R-Tree traversal ordering and the optimized R-Tree traversal ordering. The results are listed in Table. 7-2 through Table 7-4.

Table 7-2. Results of 1000 Random Orderings Under DBW Cost Model for Synthetic Data Sets

Data Set	Minimum AT (Rand_Min)	Maximum AT (Rand_Max)	Average AT (Rand_Ave)	Improvement $\frac{Rand\_\max - Rand\_Min}{Rand\_Ave}$
1	31.67	47.8	40.19	40.13%
2	81.02	102.24	92.86	22.85%
3	142.27	171.42	154.95	18.81%
4	200.15	239.73	222.11	17.82%
5	272.65	317.27	297.09	15.02%

Table 7-3. Comparisons of Hilbert and R-Tree Traversal Ordering With 1000 Random Orderings Average Under DBW Cost Model for Synthetic Data Sets

Data Set	Rand-Ave	Hilbert Ordering (HO)	R-Tree Ordering (RO)	Hilbert Ordering Improvement $\frac{Rand\_Ave}{HO}-1$	R-Tree Ordering Improvement $\frac{Rand\_Ave}{RO}-1$
1	40.19	41.47	27.06	-3.09%	48.52%
2	92.86	94.15	63.04	-1.37%	47.30%
3	154.95	152.31	94.87	1.73%	63.33%
4	222.11	211.08	135.93	5.23%	63.40%
5	297.09	294.84	178.86	0.76%	66.10%

Table 7-4. Comparison of Optimized Ordering, R-Tree Ordering and 1000 Random Orderings Average Under DBW Cost Model for Synthetic Data Sets

Data Set	Rand-Ave	R-Tree Ordering (RO)	Optimized R-Tree Ordering (OO)	R-Tree Improvement $\frac{Rand\_Ave}{RO}-1$	Opt-Improvement $\frac{RO}{OO}-1$	Overall Imrpovement $\frac{Rand\_Ave}{OO}-1$
1	40.19	27.06	22.7	48.52%	19.29%	77.05%
2	92.86	63.04	52.77	47.30%	19.45%	75.97%
3	154.95	94.87	63.07	63.33%	50.43%	145.68%
4	222.11	135.93	101.21	63.40%	34.30%	119.45%
5	297.09	178.86	121.9	66.10%	46.72%	143.72%

From the results we can see that the improvements of 1000 random orderings drop from 40% to 15% (Table 7-2). This is expected. The reason behind is that as the number of points goes up, the ratio (r) of the number of examined orderings in the algorithm to the number of all possible orderings (1000/n!) drops exponentially. Thus finding a good ordering by examining a fixed number of random orderings is not a feasible solution.

The result (Table 7.3) also shows that Hilbert SFC ordering may be better or worse than the 1000 random orderings average for the five synthetic data sets. On the contrary, the R-Tree traversal orderings improve the 1000 random orderings average significantly, from 47% to 66%. The optimized ordering further improves the R-tree traversal ordering, which varies from 19% to 47%. Consequently, the overall improvement (Table 7-4) of the optimized ordering over the 1000 random orderings average varies from 76% to 146%.

7.3.2 Experiments Using AT_{Data}^{Sep} Cost Model

Similar to the experiments on the synthetic data sets based on the DBW cost model, the corresponding results based on the AT_{Data}^{Sep} cost model are listed in Table 7-5 through Table 7-7. The overall improvement of the R-Tree traversal ordering heuristic and the optimization method varies from 17% to 30%. One of the noticeable patterns is that the improvement percentage is significantly less than that under DBW cost model. This can be explained as follows. Recall the cost model for a single complex query under the Separate Channel scheme:

$$AT_{Data}^{Sep} == \frac{1}{L}[L^2 + \frac{1}{8} - \frac{(L - L_2 - \frac{1}{2})^2}{2}]$$

This cost reaches its minimum (L/2) when L_2=0, i.e., it takes half of the cycle to reach the first data item. Similarly AT_{Data}^{Sep} reaches its maximum (L) when L_2=L, i.e., all the data items in the broadcast cycle are accessed. The possible value of AT_{Data}^{Sep} varies from L/2 to L, while it varies from 1 to L for DBW. According to our definition of improvement (AT_{Org}/AT_{Opt}-1), the upper bound of AT_{Data}^{Sep}

improvement is $\frac{L}{L/2}-1=1$ (100%), while the upper bound of DBW improvement is $L/1-1=L-1$. Note that these upper bounds are not reachable.

Table 7-5. Results of 1000 Random Orderings Under AT_{Data}^{Sep} Cost Model for Synthetic Data Sets

Data Set	Minimum AT (Rand_Min)	Maximum AT (Rand_Max)	Average AT (Rand_Ave)	Improvement $\frac{Rand\_\max - Rand\_Min}{Rand\_Ave}$
1	72.19	81.07	77.09	11.52%
2	158.12	170.73	165.28	7.63%
3	246.22	259.50	252.63	5.26%
4	328.01	345.44	338.24	5.15%
5	405.33	423.26	415.65	4.31%

Table 7-6. Comparisons of Hilbert and R-Tree Traversal Ordering Access Time with 1000 Random Orderings Average of Access Time Under AT_{Data}^{Sep} Cost Model for Synthetic Data Sets

Data Set	Rand-Ave	Hilbert Ordering (HO)	R-Tree Ordering (RO)	Hilbert Ordering Improvement $\frac{Rand\_Ave}{HO}-1$	R-Tree Ordering Improvement $\frac{Rand\_Ave}{RO}-1$
1	77.09	78.02	69.03	-1.19%	11.68%
2	165.28	165.46	145.76	-0.11%	13.39%
3	252.63	251.42	214.79	0.48%	17.62%
4	338.24	333.08	287.20	1.55%	17.77%
5	415.65	411.55	352.06	1.00%	18.06%

Table 7-7. Comparison of Access Time for Optimized Ordering, R-Tree Ordering and 1000 Random Orderings Average Under AT_{Data}^{Sep} Cost Model for Synthetic Data Sets

Data Set	Rand-Ave	R-Tree Ordering (RO)	Optimized R-Tree Ordering (OO)	R-Tree Improvement $\frac{Rand\_Ave}{RO}-1$	Opt-Improvement $\frac{RO}{OO}-1$	Overall Imrpovement $\frac{Rand\_Ave}{OO}-1$
1	77.09	69.03	65.64	11.68%	5.16%	17.44%
2	165.28	145.76	141.40	13.39%	3.08%	16.89%
3	252.63	214.79	197.47	17.62%	8.77%	27.93%
4	338.24	287.20	269.45	17.77%	6.59%	25.53%
5	415.65	352.06	319.79	18.06%	10.09%	29.98%

7.3.3 Experiments Using AT_{Data}^{Mul} Cost Model

The experiment results based on the AT_{Data}^{Mul} cost model are listed in Table 7-8 through Table 7-10. The results are similar to what we reported in the two previous sections. The R-Tree traversal heuristic and the optimization method together provide an overall improvement of 14% to 39% for the five synthetic data sets.

Although it is tempting to compare the access time of the orderings under the two cost models and then determine which one is better, we warn readers not to do so. The reason is the possible non-proportional split of access time to index and access time to data under the SEP and MUL schemes. In the MUL scheme, it takes half of the broadcast cycle length of the multiplexed channel to reach the beginning of the index and it takes additional access time to the index to reach the first data item. This is to say that reaching the beginning of the index and the beginning of the data are correlated. However, they are separate in the SEP scheme. The final result might be determined by the data set and the allocation of the bandwidth between the index channel and the data channel.

Table 7-8. Results of 1000 Random Orderings Under AT_{Data}^{Mul} Cost Model for Synthetic Data Sets

Data Set	Minimum AT (Rand_Min)	Maximum AT (Rand_Max)	Average AT (Rand_Ave)	Improvement $\frac{Rand\_\max - Rand\_Min}{Rand\_Ave}$
1	61.13	72.33	67.62	16.56%
2	138.00	152.63	145.89	10.03%
3	211.69	234.90	223.80	10.37%
4	280.71	315.15	301.38	11.43%
5	356.31	390.89	373.93	9.25%

Table 7-9. Comparisons of Hilbert and R-Tree Traversal Ordering Access Time With 1000 Random Orderings Average of Access Time Under AT_{Data}^{Mul} Cost Model for Synthetic Data Sets

Data Set	Rand-Ave	Hilbert Ordering (HO)	R-Tree Ordering (RO)	Hilbert Ordering Improvement $\frac{Rand\_Ave}{HO}-1$	R-Tree Ordering Improvement $\frac{Rand\_Ave}{RO}-1$
1	67.62	40.32	35.2	-2.68%	11.48%
2	145.89	199.4	182.42	1.24%	10.66%
3	223.80	328.93	283.52	0.99%	17.16%
4	301.38	300.12	304.4	9.58%	8.04%
5	373.93	243.6	218.69	11.06%	23.71%

Table 7-10. Comparison of Access Time for Optimized Ordering, R-Tree Ordering and 1000 Random Orderings Average Under AT_{Data}^{Mul} Cost Model for Synthetic Data Sets

Data Set	Rand-Ave	R-Tree Ordering (RO)	Optimized R-Tree Ordering (OO)	R-Tree Improvement $\frac{Rand\_Ave}{RO}-1$	Opt-Improvement $\frac{RO}{OO}-1$	Overall Improvement $\frac{Rand\_Ave}{OO}-1$
1	67.62	60.65	58.3	11.49%	4.03%	15.99%
2	145.89	131.84	128.15	10.66%	2.88%	13.84%
3	223.80	191.01	167.86	17.17%	13.79%	33.33%
4	301.38	278.96	248.11	8.04%	12.43%	21.47%
5	373.93	302.26	269.69	23.71%	12.08%	38.65%

7.4 The Zip-code Point Data Sets

The centers of the zip codes are in the form of latitude/longitude pairs. We choose the following query window sizes in our experiments for the zip code data sets: 0.05 degree by 0.05 degree, 0.1 degree by 0.1 degree, 0.5 degree by 0.5 degree, 1 degree by 1 degree and 5 degree by 5 degree. The smallest query window size is approximately 5 kilometers by 5 kilometers and the largest query window size is

approximately 500 kilometers by 500 kilometers. We believe these query window sizes are meaningful in real applications. We set the maximum numbers of nodes in a hyperedge to be 10, 20, 30, 40 and 50 for the five query window sizes, respectively. Note that for some data sets that have small areas but a large number of zip codes (points), such as the Washington D.C. data set and the Rode Island data set, it is impossible for a large query window to contain fewer than the threshold numbers of points centered at any point in those data sets. Consequently there is no hypergraph generated for the data set. In this case we simply discard the data set for the particular query window. Although it is possible for the data sets with a large area but small number of points to have hypergraphs that all hyperedges of which have only one node for small query windows, i.e., no two extended regions intersect with each other, this scenario does not happen in our experiments. This is primarily due to the clustered distribution of the data sets.

Due to the volume of the data sets and their experimental results, we list them in the appendix A. Table A-1 lists the parameters that characterizing the hypergraphs for each of the data sets in terms of the number of points for sequencing, the number of hyperedges (i.e., the number of possible distinct query result sets) and the average number of nodes per hyperedge. Table A-3 through Table A-17 list the experiment results for the data sets using the five query window sizes under the three cost models. The meanings of the column names in these 15 tables are listed in Table A-2. The computation time for the optimization methods (DBW/AT_{Data}^{Sep} and AT_{Data}^{Mul})

are listed in Table A-18. The average access times under the three cost models and five query windows for the zip code data sets are listed in Table 7-11.

Table 7-11. Summary of Results of Zip Code Data Sets

Cost Model	Window Size	Rand _Ave	HO	RO	OO	$\frac{Rand\_Ave}{RO}-1$	$\frac{RO}{OO}-1$	$\frac{Rand\_Ave}{OO}-1$
DBW	0.05	158.56	157.93	61.02	42.07	160.19%	51.81%	290.33%
	0.1	178.07	177.30	77.84	57.58	144.75%	44.76%	254.83%
	0.5	330.23	326.75	153.78	114.38	130.03%	42.62%	229.57%
	1	419.02	418.95	224.95	177.96	89.10%	30.03%	144.89%
	5	141.78	140.84	95.35	67.26	46.92%	43.32%	109.26%
SEP	0.05	282.15	282.14	218.02	206.12	27.66%	6.65%	36.05%
	0.1	309.92	309.48	245.46	233.22	26.52%	6.09%	34.20%
	0.5	548.01	546.07	438.52	414.86	25.50%	6.24%	33.31%
	1	637.31	637.32	526.39	499.88	20.52%	5.82%	27.49%
	5	175.59	175.37	155.09	141.87	11.47%	9.11%	21.56%
MUL	0.05	250.65	250.31	201.66	192.59	23.50%	6.07%	30.89%
	0.1	276.09	275.45	224.71	216.07	23.43%	5.46%	30.14%
	0.5	489.8	488.09	398.75	377.27	22.55%	7.43%	31.62%
	1	572.85	572.04	473.57	453.62	20.11%	7.35%	28.84%
	5	162.37	161.29	134.85	124.27	20.27%	14.49%	38.28%

From Table A-1 to A-18 and the summary table (Table 7-11) we can see the similar results as those in the five synthetic data sets. For the zip code data sets, on average, the optimized orderings are better than 1000 times random ordering average 1.09 to 2.90 times under DBW, 22% to 36% under SEP and 30% to 38% under MUL, respectively. The R-Tree heuristics contribute approximately 2/3 to the overall improvement and the optimization methods contribute approximately 1/3 to the overall improvements. The Hilbert orderings have similar performance as the 1000 times random ordering average. The results show that both the R-Tree traversal heuristic and the optimization methods are effective which makes the orderings based on them query efficient. We next analyze the computation times for the optimizations, i.e., the time it takes to run the optimization methods.

The computation time for the optimization method for DBW/AT_{Data}^{Sep} and AT_{Data}^{Mul} are listed in Table A-18 and shown in Fig. 7-2 and Fig. 7-3, respectively. Note that the trend lines for query window 0.05 by 0.05 and query window 0.1 by 0.1 are almost overlapped due to their small computation times. From the results we can see that the computation times are generally quadratic with respect to the number of nodes in the hypergraphs. They support the theoretical result of MinLA (on which the DBW optimization method is based) given in (Bar-Yehuda, 2001) and our AT_{Data}^{Mul} optimization method very well. The computation time for our AT_{Data}^{Mul} optimization method is about 2/3 of that of the MinLA algorithm.

The computation times for running DBW/AT_{Data}^{Sep} and AT_{Data}^{Mul} optimization reach their maximum for PA data set for 1.0 by 1.0 query window, which are 127 and 76 seconds respectively. The hypergraph representation of the PA data set has 1847 nodes, 17447 hyperedges and 8.69 nodes per hyperedge on average. The PA data set also has the largest computation time for both optimization methods under the 0.5 by 0.5 query window, which are 46 and 34 seconds respectively and its hypergraph representation has 2075 nodes, 6862 hyperedges and 5.62 nodes per hyperedge on average. For PA data set, although it has more nodes in the hypergraph representation under the 0.5 by 0.5 query window than the hypergraph representation under 1.0 by 1.0 query window, the number of hyperedges and the number of nodes per hyperedge under the 1.0 by 1.0 query window is significantly larger than the number of hyperedges and the number of nodes per hyperedge under the 0.5 by 0.5 query window respectively. This explains why it takes more time for both

optimization methods under the 1.0 by 1.0 query window than that under 0.5 by 0.5 query window. The similar thing happens to the SD data set in 5.0 by 5.0 query window. Although the SD data set under the query window has only the 6th largest number of nodes among all the data sets, it has the largest number of hyperedges and considerable large average number of nodes per hyperedge, thus it takes the longest computation times for the two optimization methods, which are 33 and 15 seconds, respectively. The results show the importance of the hidden const factor behind the big O notation.

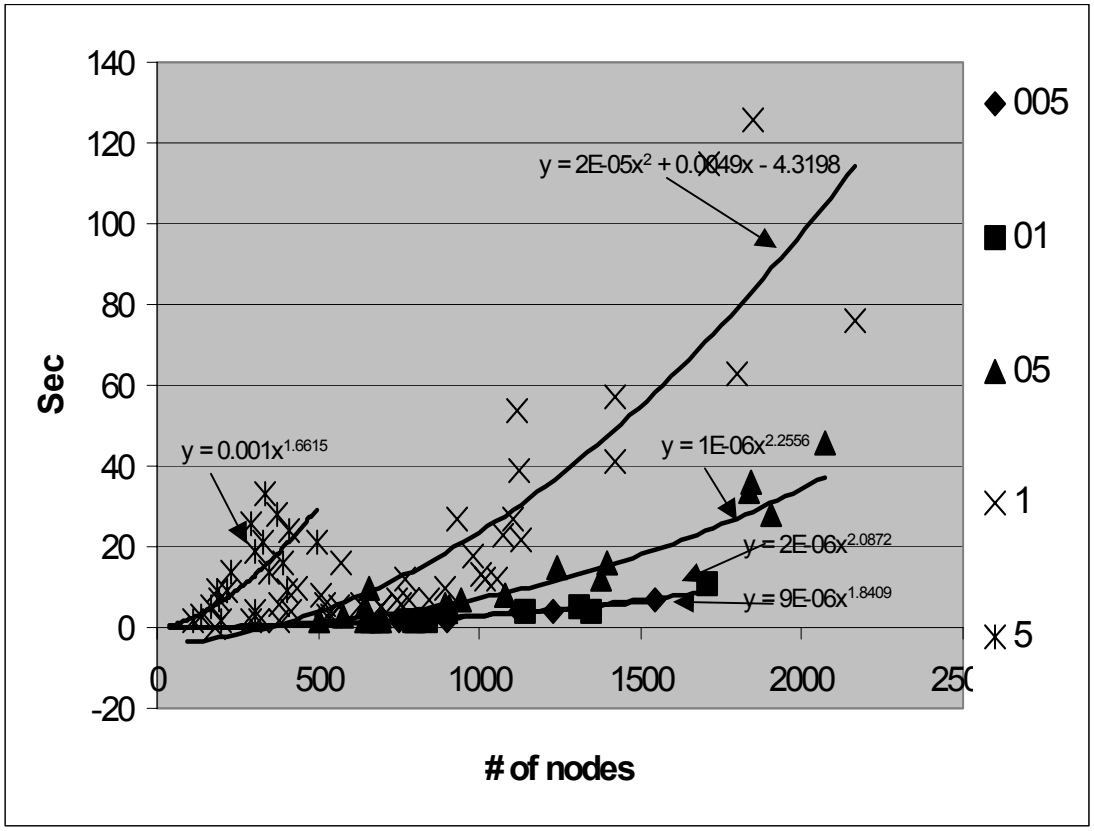


Fig. 7-2. Computation Time for DBW/AT_{Data}^{Sep} Optimization Method of Zip Code Data Sets

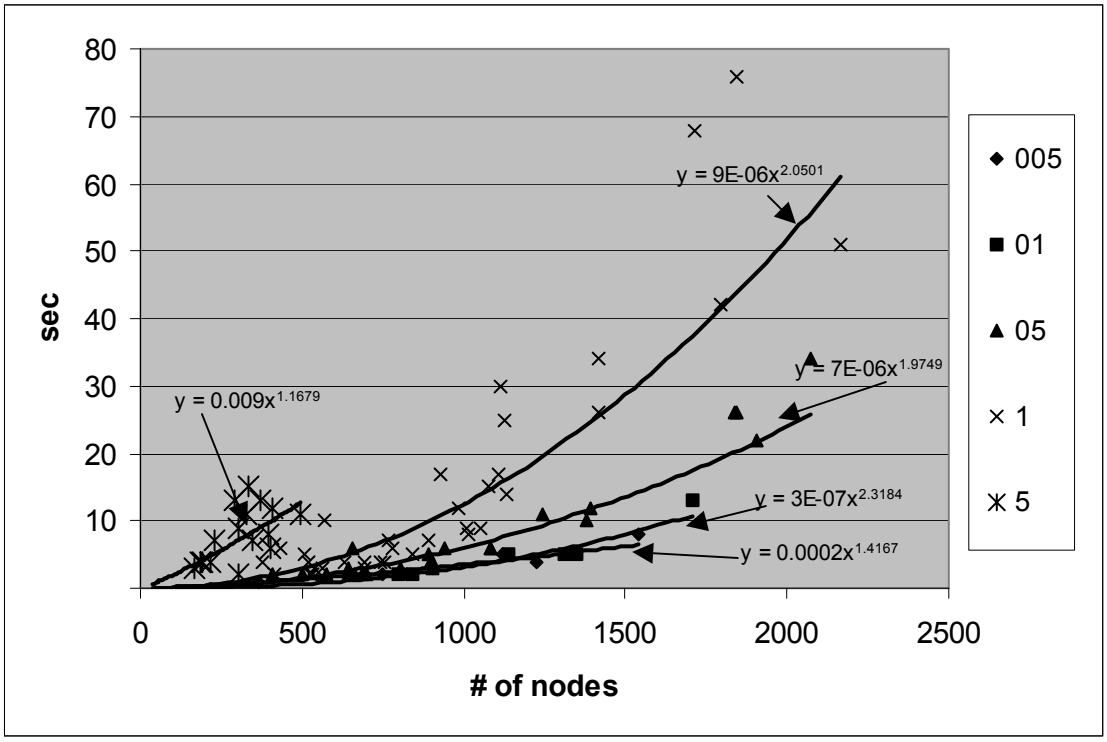


Fig. 7-3. Computation Time for AT_{Data}^{Mul} Optimization Method For Zip Code Data Sets

Although it takes about two minutes to optimize the ordering for the most complex data set (PA under 1.0 by 1.0 query window and running DBW/AT_{Data}^{Sep} optimization) , as we have discussed in the introduction, geographical data changes relatively slow in practice. Furthermore, since the optimizations are done on the server side, we believe that the optimization time can be significantly reduced by using more powerful processors to fulfill the requirements in practice. Due to the divide-and-conquer nature of the optimization methods, it is also possible to explore parallelism to further reduce the computation time.

7.4 Texas Transportation Network Data Set

The data set is shown in Fig 7-4. There are 62 nodes (cities) and 120 edges (major roads) in the transportation network. The number of all-pair shortest paths is 62*(62-1)/2=1891, i.e., the hypergraph representation for network path query has 1891 hyperedges. For spatial range queries, the query window we use is a 100 by 100 square miles area. The resulting hypergraph representation has 62 nodes (cities) and 420 hyperedges.

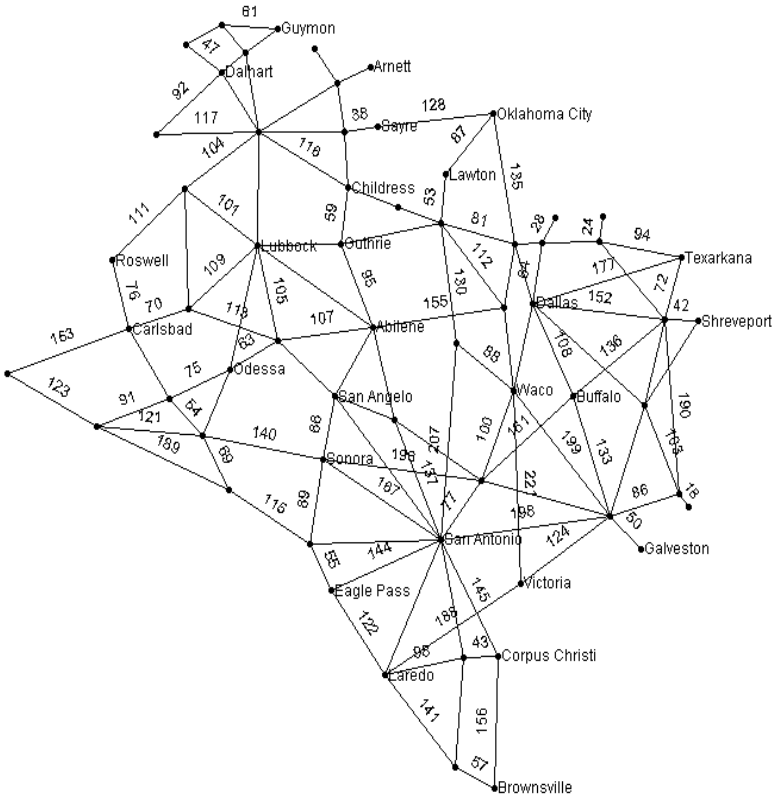


Fig. 7-4. Texas Road Network

We explore the following ordering heuristics for experiments using the DBW cost model (c.f. Fig. 7-1): random, original graph traversal/EAFG traversal (BFS/DFS), traversal of the original graph partition tree, traversal of the derived hypergraph partition tree, traversal of EAFG partition tree, Hilbert SFC and R-Tree traversal. For BFS/DFS traversals, start at different nodes will generate different orderings. We report min, max and average of the n BFS/DFS orderings using different nodes as the starting node where n is the number of vertices in the network. For the R-Tree traversal ordering heuristic, we vary the branch factor from 4 to 9. We also evaluate the optimized orderings using the following BDTs: the original graph partition tree, the EAFG partition tree, the hypergraph partition tree and the decomposed R-Tree built from treating graph nodes as geographical points. These heuristic orderings and optimized orderings based on the hypergraph representation of network path queries on graph data are evaluated first on the network path queries. The same orderings are then evaluated on the hypergraph representation of spatial range queries on point data. The results are shown in Table 7-12.

From the results we can see that for network path queries, traversal of the graph (original graph, EAFG and hypergraph) partition tree orderings and their optimized orderings achieve much better results than both the graph traversal orderings and geometric based heuristic orderings. Among these orderings, traversal of the hypergraph partition tree ordering as an ordering heuristic is the best. On the other hand, the optimized ordering based on the EAFG partition tree is the best

among the three optimized orderings although they are pretty close. The optimized ordering based on the original graph partition tree has the largest improvement ratio over its graph partition tree traversal ordering heuristic.

Table 7-12. Summary of Results of Texas Transportation Network Data Set Under DBW Cost Model

Orderings	Path Query	Range Query
n-Rand-min	35.55	26.68
n-Rand-max	45.10	34.25
n-Rand-Avg	40.79	30.46
n-EAFG-BFS-min	35.16	21.58
n-EAFG-BFS-max	42.57	32.24
n-EAFG-BFS-avg	39.35	27.97
n-EAFG-DFS-min	32.54	18.58
n-EAFG-DFS-max	40.23	27.93
n-EAFG-DFS-avg	37.31	23.50
Traversal of original graph partition tree	30.69	21.33
Optimization based on original graph partition tree	22.56	20.04
Traversal of EAFG partition tree	26.37	20.61
Optimization based on EAFG partition tree	22.26	21.16
Traversal of hypergraph partition tree	24.25	25.19
Optimization based on hypergraph partition tree	22.74	18.69
Hilbert SFC	38.63	25.61
Traversal of R-Tree –min	35.73	17.76
Traversal of R-Tree –max	39.97	25.92
Traversal of R-Tree –avg	37.99	21.24
Optimization R-Tree –min	33.53	13.31
Optimization R-Tree –max	35.87	22.41
Optimization R-Tree –avg	34.30	13.49

Since the EAFG and the original graph has the same topology, their breadth first search orderings and depth first search orderings are also the same. Among the EAFG traversal ordering heuristics, DFS seems to be better for all *n*-min/max/avg cases than those of BFS. The Hilbert ordering, although better than the maximum of

random ordering orderings, is worse than their average. The R-Tree traversal orderings average is slightly worse than the DFS traversals average of EAFG, but is better than BFS traversals average of EAFG. The results suggest that the geometric based heuristic orderings (R-Tree traversal) and their optimized orderings are not as good as graph partition based ones (Original/EAFG/Hypergraph partition tree traversal).

It is interesting to see that geometric-based ordering heuristics and their optimized orderings where optimizations are based on the hypergraph representation of spatial range queries perform better than graph partition orderings and their optimized orderings where the optimizations are based on the hypergraph representation of network path queries. We thus draw our conclusion that geometric based orderings should be used for spatial range queries and graph partition based orderings should be used for network path queries.

We next perform experiments using the AT_{Data}^{Mul} cost model on network path queries. Several new ordering heuristics, such as Maximum Spanning Tree , MAX, MAX-LD, NODE-WEIGHT and EDGE-WEIGHT as discussed in Chapter 5, are available under MUL scheme but not under DBW/SEP scheme.

Although the Maximum Spanning Tree based orderings do not make much sense under the DBW cost model, it works well under the AT_{Data}^{Mul} cost model since they put nodes (or edges) with larger weights as close to the beginning of a broadcast cycle as possible. The similar arguments can be made for MAX, MAX-LD, NODE-WEIGHT and EDGE-WEIGHT heuristics. Note that the Prim's MST algorithm and

the Kruskal MST algorithm although generate the same MST, might have different orderings. For the Prim MST, we set each node in the graph as the source and record the sequence of nodes being visited to obtain *n* Prim's MST orderings. While the MAX and MAX-LD heuristics cannot be extended to hypergraph easily, the NODE-WEIGHT and EDGE-WEIGHT heuristics can be used for both a regular graph and a hypergraph. We also include graph partition tree traversal orderings and their optimized orderings. Note that when applying optimizations, the original graph and the EAFG are used only to generate the BDTs while the hypergraph is still used as the underlying representation in all the three optimizations. Like the experiments under the DBW cost model, we also include the Hilbert and the R-Tree traversal ordering heuristics. Again R-Trees are used only for generating the BDTs . The results are listed in Table 7-13. For the numbers that stride multiple columns, they are the same by nature for the types of graphs denoted by the columns.

From the results we can see that graph partition based heuristic orderings and their optimized orderings remain among the best orderings under AT_{Data}^{Mul} cost model. The Kruskal-MST heuristic on the EAFG, the MAX and the MAX-LD heuristics on the original graph/EAFG are slightly better than the rest heuristic orderings. Although they are still slightly worse than the 1000 random ordering minimum, they are better than the 1000 random ordering average. Considering the computation cost of these heuristics and the cost of examining 1000 random orderings, they are preferred to random orderings. Although the optimized orderings using R-tree as BDTs improve the R-Tree traversal ordering heuristics by 5% on

average, they are still only comparable to the Kruskal-MST heuristic on the EAFG, the MAX and the MAX-LD heuristics on original graph/EAFG. Thus we do not recommend performing optimization using R-Tree as the BDT construction to optimize broadcast ordering for network path query processing.

Table 7-13. Summary of Results of Texas Transportation Network Data Set Under AT_{Data}^{Mul} Cost Model

Orderings	Graph Types		
	ORGN	EAFG	Hyper
1000-Rand-min	47.39		
1000-Rand-max	53.39		
1000-Rand-Avg	51.00		
n-BFS-min	47.78		N/A
n-BFS-max	53.49		N/A
n-BFS-avg	50.41		N/A
n-DFS-min	48.67		N/A
n-DFS-max	53.07		N/A
n-DFS-avg	51.35		N/A
n-Prim-MST-Min	47.89	48.67	N/A
n-Prim-MST-Max	53.09	52.18	N/A
n-Prim-MST-Avg	49.56	50.47	N/A
Kruskal-MST	49.61	47.01	N/A
MAX	47.79		N/A
MAX-LD	47.64		N/A
NODE-WEIGHT	52.28	52.00	52.43
EDGE-WEIGHT	50.91	53.15	51.95
Traversal of partition tree	44.38	47.79	45.01
Optimization based on partition tree	41.76	41.31	41.20
Hilbert SFC	50.71		
Traversal of R-Tree –min	48.50		
Traversal of R-Tree –max	51.09		
Traversal of R-Tree –avg	50.09		
Optimization R-Tree –min	46.04		
Optimization R-Tree –max	48.79		
Optimization R-Tree –avg	47.56		

Chapter 8

Conclusions and Future Work

Geographical information has been widely used in our everyday lives. Most geographical information are public and many of them are frequently requested by a large number of users. We believe broadcasting geographical information over air is an attractive solution for emerging location dependent services, in terms of scalability, mobility management at the server side and power consumption at the client side. In addition, geographical information broadcast may play unique roles in many applications, such as in unusual event monitoring, disaster rescue and military operations.

In this study, our focuses were to develop cost models and optimization algorithms for placing geographical data items onto a broadcast channel based on their spatial semantics to reduce the response time and energy consumption for processing spatial queries over the broadcast channel. Our work can be summarized as follows:

1. We divided data access time into four components, namely IPW, IBW, DPW and DBW. This is an extension to the classic division of access time into Probe Wait and Bcast Wait. The extension allows studying access time to index and access time to data separately. While the classic division mostly targets at the multiplexing broadcast scheme, the extension works for both the multiplexing broadcast scheme (MUL) and the broadcast scheme that uses separate channels for index and data (SEP).
2. We developed the cost models for computing the data access time for processing spatial queries over broadcast geographical data, including DBW, AT_{Data}^{Mul} and

AT_{Data}^{Sep}. Although DBW and AT_{Data}^{Mul} are relatively straightforward, deriving AT_{Data}^{Sep} has gone through much elaborated work. The derived simple quadratic form of AT_{Data}^{Sep} for processing a single complex query is not only easy to use but also theoretically meaningful, which is the base for us to propose using DBW to approximate $AT_{Data}^{Sep.}$

3. Given a query window size, we proposed a method for computing all possible query result sets and their weights for point data. This result also lays the foundation for representing a spatial range query result as a hypergraph edge in a hypergraph and for relating data placement problem with graph MinLA problem.
4. We discussed a family of low-cost heuristics for data placement in a broadcast channel and put them into a cohesive classification structure. These heuristics can be used to generate the orderings of broadcast sequences directly or used as the initial orderings for further optimization. Specifically we discussed the following heuristics in detail: R-tree traversal ordering, Hilbert SFC ordering, graph partition tree traversal ordering, ordering based on degree/weight, and spanning tree ordering.
5. We provided three optimization methods for reducing data access time under the cost models, DBW, AT_{Data}^{Mul} and AT_{Data}^{Sep}, respectively. They can be applied to spatial range queries, network path queries or any other types of complex queries. We first proposed to use an efficient graph MinLA algorithm to optimize DBW. Since our cost model of AT_{Data}^{Sep} shows the monotonic relationship between DBW and AT_{Data}^{Sep}, we proposed to use DBW to approximate AT_{Data}^{Sep} and use the same algorithm to optimize AT_{Data}^{Sep}. Our most significant contribution

related to optimization is the novel method to optimize AT_{Data}^{Mul}. Although following the same divide-and-conquer strategy and using BDT as a global constraint for ordering as in (Bar-Yehuda, 2001), we compute $\max\{\pi(n_1), \pi(n_2), ...\pi(n_k)\}$ directly by efficiently computing the total sizes of the sub-trees to the left of the path from the ending node of $\{n_1, n_2, \ldots n_k\}$ to the root of a BDT. This is quite different from the strategy adopted in (Bar-Yehuda, 2001) which transforms computing $|\pi(u)-\pi(v)|$ for an edge to recursive summations of the sub-tree sizes of a BDT. However, the strategy adopted in (Bar-Yehuda, 2001) can not be applied to computing $\max\{\pi(n_1), \pi(n_2), ...\pi(n_k)\}$ in optimizing AT_{Data}^{Mul} due to its non-linear nature.

6. We performed experiments on five synthetic point data sets, 51 zip code point data sets of 51 states of US, and the Texas road network graph data set. The results show that the three proposed optimization methods are very effective. For the 51 zip data sets, on average, the data access time based on the optimized ordering is only about 1/3 of that of the 1000 time random orderings average under the DBW cost model. The performances are also improved about 30% under the AT_{Data}^{Mul} and the AT_{Data}^{Sep} cost models, both with acceptable computation overheads. The results from the geometric and graph-based heuristics and their optimizations under the DBW and the AT_{Data}^{Mul} cost models applied to the Texas road network data set show that geometric heuristic should be applied to optimizations of spatial range queries for point data sets and graph heuristic should be applied to optimizations of network path queries for graph data sets.

For future work, we first plan to take access time to the index channel into consideration. Although index placement has been extensively studied, very few techniques are specifically designed for multi-dimensional data that can be applied to geographical information. Although not considered in this study, it is possible that there exists a better scheme to combine index and data other than the MUL and SEP. Specifically, for MUL scheme, it is desirable to consider access frequencies of query result sets and their access paths to an index tree simultaneously. The challenge might be to handle hypergraphs for unordered data accesses and index trees for a combination of ordered data access (parent/ child) and unordered data access (siblings) at the same time.

Second, although the complexities in our proposed optimization methods, either adopted from the graph MinLA problem or developed by ourselves, are the smallest to the best of our knowledge, they are still super-quadratic. They might not be applicable when the number of data items in a data set (or the number of nodes in the data set's hypergraph representation) is large. A solution might be to follow the multi-scale paradigm, i.e., the size of a hypergraph is first reduced by collapsing nodes and edges to generate a higher level graph/hypergraph. The nodes of the higher level graph/hypergraph, the number of which is much smaller than the number of nodes in the original hypergraph, are then ordered. The nodes of the lower level graph/hypergraph are then ordered recursively until all the nodes of the lowest level graph/hypergraph are sequenced, i.e., all the nodes in the original hypergraph are ordered. (Koren, 2002) proposed a multi-scale algorithm for graph MinLA, however, we are not clear of its

applicability and extensibility in data placement of a broadcast channel, especially under the AT_{Data}^{Mul} cost model.

Finally, we plan to investigate on more efficient methods to compute access frequencies (i.e., weights of hyperedges) of point data sets, explore more ordering heuristics, and perform more experiments using both synthetic and real data sets with different sizes and distributions to examine the practical effectiveness and scalabilities of the optimization methods.

Reference

(In DBLP Format)

1. Swarup Acharya, Rafael Alonso, Michael J. Franklin, Stanley B. Zdonik: Broadcast Disks: Data Management for Asymmetric Communications Environments. SIGMOD Conference 1995: 199-210.
2. Pankaj Agarwal, Lars Arge, Jeff Erickson: Indexing Moving Points. PODS, 2000:175-186.
3. Rakesh Agrawal, H. V. Jagadish: Materialization and Incremental Update of Path Information. ICDE, 1989: 374-383.
4. Demet Aksoy, Mehmet Altinel, Rahul Bose, Ugur Çetintemel, Michael J. Franklin, Jane Wang, Stanley B. Zdonik: Research in Data Broadcast and Dissemination. AMCP 1998: 194-207.
5. Charles J. Alpert, Andrew B. Kahng: Recent Directions in Netlist Partitioning. Integration, the VLSI Journal, 19(1-2): 1-81(1995).
6. Charles J. Alpert, Andrew B. Kahng: A General Framework for Vertex Orderings With Applications to Circuit Clustering. IEEE Transactions on Very Large Scale Integration (VLSI) Systems, 4(2): 240 –246(1996).
7. Ning An, Ji Jin, Anand Sivasubramaniam: Toward an Accurate Analysis of Range Queries on Spatial Data. IEEE Transaction on Knowledge and Data Engineering 15(2): 305-323 (2003).
8. Walid G. Aref, Ibrahim Kamel: On Multi-dimensional Sorting Orders. DEXA 2000: 774-783.
9. Franz Aurenhammer: Voronoi Diagrams - A Survey of a Fundamental Geometric Data Structure. ACM Computing Surveys 23(3): 345-405 (1991).
10. Lukas Bachmann, Bernd-Uwe Pagel, Hans-Werner Six: Optimizing Spatial Data Structures For Static Data. IGIS 1994: 247-258.
11. 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.
12. Chee Yong Chan, Beng Chin Ooi: Efficient Scheduling of Page Access in Index-Based Join Processing. IEEE Transaction on Knowledge and Data Engineering 9(6): 1005-1011 (1997).
13. Edward P. F. Chan and Ning Zhang: Finding Shortest Paths in Large Network System. ACM-GIS 2001:160-166.

14. Y. C. Chehadeh, Ali 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).

15. Li Chen, Rupesh Choubey, Elke A. Rundensteiner: Bulk-Insertions Into R-Trees Using the Small-Tree-Large-Tree Approach. ACM-GIS 1998: 161-162.

16. Li Chen, Rupesh Choubey, Elke A. Rundensteiner: Merging R-Trees: Efficient Strategies for Local Bulk Insertion. GeoInformatica 6(1): 7-34 (2002).

17. Richard Steven Chernock, Data Broadcasting: Understanding the ATSC Data Broadcast Standard, McGraw-Hill, 2001.

18. Dae-Soo Cho, Bong-Hee Hong: Optimal Page Ordering for Region Queries in Static Spatial Databases. DEXA 2000: 366-375.

19. Hae Don Chon, Divyakant Agrawal, Amr El Abbadi: Storage and Retrieval of Moving Objects, MDM 2001: 173-184.

20. Rupesh Choubey, Li Chen, Elke A. Rundensteiner: GBI: A Generalized R-Tree Bulk-Insertion Strategy. SSD 1999: 91-108.

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

22. Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Cliff Stein: Introduction to Algorithms (Second Edition), MIT Press and McGraw-Hill, 2001.

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

24. Guy Even, Joseph Naor, Satish Rao, Baruch Schieber: Divide-and-Conquer Approximation Algorithms via Spreading Metrics. Journal of ACM 47(4): 585-616 (2000).

25. Christos Faloustos: Gray-Codes for Partial Match and Range Queries. IEEE Transaction on Software Engineering 14(10): 1381-1393(1988).

26. Christos Faloutsos, Shari Roseman: Fractals for Secondary Key Retrieval. PODS 1989: 247-252.

27. Luca Forlizzi, Ralf Hartmut Güting, Enrico Nardelli, Markus Schneider: A Data Model and Data Structures for Moving Objects Databases. SIGMOD Conference 2000: 319-330.

28. Farshad Fotouhi, Sakti Pramanik: Optimal Secondary Storage Access Sequence for Performing Relational Join. IEEE Transaction on Knowledge and Data Engineering 1(3): 318-328 (1989).

29. Volker Gaede, Oliver Günther: Multidimensional Access Methods. ACM Computing Surveys 30(2): 170-231 (1998).

30. Yvan J. Garcia, Mario A. Lopez, Scott T. Leutenegger: A Greedy Algorithm for Bulk Loading R-Trees. ACM-GIS 1998: 163-164.

31. David K. Gifford, John M. Lucassen, Stephen T. Berlin: The Application of Digital Broadcast Communication to Large Scale Information Systems. IEEE Journal on Selected Areas in Communications 3(3): 457-467 (1985).

32. Veena Gondhalekar, Ravi Jain, John Werth: Scheduling on Airdisks: Efficient Access to Personalized Information Services via Periodic Wireless Data Broadcast. ICC (3) 1997: 1276-1280.

33. Ralf Hartmut Guting, Michael H. Böhlen, Martin Erwig, Christian S. Jensen, Nikos A. Lorentzos, Markus Schneider, Michalis Vazirgiannis: ACM Transactions on Database Systems 25 (1): 2000:1-42.

34. Antonin Guttman: R-Trees: A Dynamic Index Structure for Spatial Searching. SIGMOD Conference 1984: 47-57.

35. Susanne E. Hambrusch, Chuan-Ming Liu, Walid G. Aref, Sunil Prabhakar: Query Processing in Broadcasted Spatial Index Trees. SSTD 2001: 502-521.

36. Gary E. Herman, Gita Gopal, K. C. Lee, Abel Weinrib: The Datacycle Architecture for Very High Throughput Database Systems. SIGMOD Conference 1987: 97-103.

37. Jiawei Han, Micheline Kamber, Anthony K. H. Tung: Spatial Clustering Methods in Data Mining: A Survey, in Harvey J. Miller, Jiawei Han (editors), Geographic Data Mining and Knowledge Discovery, Taylor and Francis, 2001:1-29

38. Wolfgang Hoeg: Digital Audio Broadcasting: Principles and Applications, Wiley, 2001.

39. 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).

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

41. Jiun-Long Huang, Ming-Syan Chen: Broadcast Program Generation for Unordered Queries with Data Replication. SAC 2003: 866-870.

42. Yun-Wu Hung, Ning Jing, Elke A.Rundensteiner: Effective Graph Clustering for Path Queries in Digital Map Databases. CIKM 1996: 215-222.

43. Tomasz Imielinski, B. R. Badrinath: Data Management for Mobile Computing. SIGMOD Record 22(1): 34-39 (1993).

44. Tomasz Imielinski, S. Viswanathan, B. R. Badrinath: Energy Efficient Indexing on Air. SIGMOD Conference 1994: 25-36.

45. Tomasz Imielinski, S. Viswanathan, B. R. Badrinath: Power Efficient Filtering of Data an Air. EDBT 1994: 245-258.

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

47. H. V. Jagadish: Linear Clustering of Objects with Multiple Attributes. SIGMOD Conference 1990: 332-342.

48. Ning Jing, Yun-Wu Huang, Elke A. Rundensteiner: Hierarchical Encoded Path Views for Path Query Processing: An Optimal Model and Its Performance Evaluation. IEEE Transactions on Knowledge and Data Engineering 10(3): 409-432(1998).

49. David S. Johnson, Lyle A. McGeoch: Experimental Analysis of Heuristics for the STSP, in Gregory Gutin, Abraham P. Punnen (Editors), "The Traveling Salesman Problem and its Variations", Kluwer Academic Publishers, 2002:369-443.

50. D. S. Johnson, G. Gutin, Lyle A. McGeoch, A. Yeo, Weixiong Zhang, A. Zverovich: Experimental Analysis of Heuristics for the ATSP, in Gregory Gutin, Abraham P. Punnen (Editors), "The Traveling Salesman Problem and its Variations", Kluwer Academic Publishers, 2002: 445-487.

51. Ibrahim Kamel, Christos Faloutsos: On Packing R-trees. CIKM 1993: 490-499.

52. Ibrahim Kamel, Christos Faloutsos: Hilbert R-tree: An Improved R-tree using Fractals. VLDB 1994: 500-509.

53. George Kollios, Dimitrios Gunopulos, Vassilis J. Tsotras: On Indexing Mobile Objects. PODS 1999: 261-272.

54. Birgitta Konig-Ries, Kia Makki, S. A. M. Makki, Charles E. Perkins, Niki Pissinou, Peter L. Reiher, Peter Scheuermann, Jari Veijalainen, Ouri Wolfson: Report on the NSF Workshop on Building an Infrastructure for Mobile and Wireless Systems. SIGMOD Record 31(2): 73-79 (2002).

55. Yehuda Koren, David Harel: A Multi-scale Algorithm for the Linear Arrangement Problem. WG 2002: 296-309.

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

57. Akhil Kumar, Waleed A. Muhanna, Raymond A. Patterson: Mean-Variance Analysis of the Performance of Spatial Ordering Methods. International Journal of Geographical Information Science 12(3): 269-289 (1998).

58. Ming-Ter Kuo, Chung-Kuan Cheng: A Network Flow Approach for Hierarchical Tree Partitioning. DAC 1997: 512-517.

59. Kam-yiu Lam, Özgür Ulusoy, Tony S. H. Lee, Edward Chan, Guohui Li: An Efficient Method for Generating Location Updates for Processing of Location-Dependent Continuous Queries. DASFAA, 2001: 218-225.

60. 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).

61. Guanling Lee, Shou-Chih Lo: Broadcast Data Allocation for Efficient Access of Multiple Data Items in Mobile Environments. Mobile Networks and Applications 8(4): 365-375 (2003).

62. 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).

63. Alexander Leonhardi, Kurt Rothermel: Architecture of a Large-Scale Location Service. ICDCS 2002: 465-466.

64. Scott T. Leutenegger, J. M. Edgington, Mario A. Lopez: STR: A Simple and Efficient Algorithm for R-Tree Packing. ICDE 1997: 497-506.

65. Vincenzo Liberatore: Multicast Scheduling for List Requests. INFOCOM 2002: 1129 – 1137.

66. Andrew Lim, Jennifer Lai-Pheng Kwan, Wee-Chong Oon: Page Access Scheduling in Join Processing. CIKM 1999: 276-283.

67. Martin Mauve, Jörg Widmer, Hannes Hartenstein: A survey on position-based routing in mobile ad hoc networks. IEEE Network Magazine 15(6):0-39 (2001).

68. G. Morton: A Computer Oriented Geodetic Database and a New Technique in file Sequencing, IBM Ltd., Ottawa, Canada, 1966.

69. Jack A. Orenstein, T. H. Merrett: A Class of Data Structures for Associative Searching. PODS 1984: 181-190.

70. 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.

71. Wen-Chih Peng, Ming-Syan Chen: Dynamic Generation of Data Broadcasting Programs for a Broadcast Disk Array. CIKM 2000: 38-45.

72. Dieter Pfoser, Christian S. Jensen, Yannis Theodoridis: Novel Approaches to the Indexing of Moving Object Trajectories. VLDB, 2000: 395-406.

73. Dieter Pfoser, Christian S. Jensen: Querying the trajectories of on-line mobile objects. MobiDE 2001: 66-73.

74. Sakti Pramanik, David Ittner: Use of Graph-Theoretic Models for Optimal Relational Database Accesses to Perform Join. ACM Transaction on Database Systems 10(1): 57-74 (1985).

75. Satish Rao, Andréa W. Richa: New Approximation Techniques for Some Ordering Problems. SODA 1998: 211-218.

76. Ulrich Reimers: Digital Video Broadcasting (DVB): The International Standard for Digital Television, Springer, 2001.

77. Qun Ren, Margaret H. Dunham: Using Semantic Caching to Manage Location Dependent Data in Mobile Computing. MOBICOM 2000: 210-221.

78. Philippe Rigaux, Michel O. Scholl, Agnes Voisard, Spatial Databases: With Application to GIS, Academic Press, 2002 .

79. Nick Roussopoulos, Daniel Leifker: Direct Spatial Search on Pictorial Databases Using Packed R-Trees. SIGMOD Conference 1985: 17-31.

80. Simonas Saltenis, Christian S. Jensen, Scott T. Leutenegger, Mario A. Lopez: Indexing the Positions of Continuously Moving Objects. SIGMOD Conference 2000: 331-342.

81. Hanan Samet: The Design and Analysis of Spatial Data Structures, Addison-Wesley, 1990.

82. Kirk Schloegel, George Karypis, Vipin Kumar: Graph Partitioning for High Performance Scientific Simulations, in J. Dongarra, I. Foster, G. Fox, K. Kennedy, A. White (editors), "CRPC Parallel Computing Handbook", Morgan Kaufmann, 2000.

83. Timos K. Sellis, Nick Roussopoulos, Christos Faloutsos: The R+-Tree: A Dynamic Index for Multi-Dimensional Objects. VLDB 1987: 507-518.

84. Ayse Y. Seydim, Margaret H. Dunham, Vijay Kumar: Location dependent query processing. MobiDE 2001: 47-53.

85. Shashi Shekhar, Duen-Ren Liu: CCAM: A Connectivity-Clustered Access Method for Networks and Network Computations. IEEE Transactions on Knowledge and Data Engineering 9(1): 102-119(1997).

86. Shashi Shekhar, Andrew Fetterer, Bjajesh Goyal: Materialization Trade-Offs in Hierarchical Shortest Path Algorithms. SSD 1997:94-11.

87. Shashi Shekhar, Chang-Tien Lu, Sanjay Chawla, Sivakumar Ravada: Efficient Join-Index-Based Spatial-Join Processing: A Clustering Approach. IEEE Transactions on Knowledge and Data Engineering 14(6): 1400-1421 (2002).

88. Shashi Shekhar, Sanjay Chawla: Spatial Databases: A Tour, Prentice Hall, 2003.

89. Narayanan Shivakumar, Suresh Venkatasubramanian: Energy-efficient Indexing for Information Dissemination in Wireless Systems. Mobile Networks and Applications 1(4), 1996: 433-446.

90. Antonio Si, Hong Va Leong: Query Optimization for Broadcast Database. Data and Knowledge Engineering 29(3): 351-380 (1999).

91. A. Prasad Sistla, Ouri Wolfson, Sam Chamberlain, Son Dao: Modeling and Querying Moving Objects. ICDE 1997: 422-432.

92. Kian-Lee Tan, Jeffrey Xu Yu: Generating Broadcast Programs that Support Range Queries. IEEE Transactions on Knowledge and Data Engineering 10(4): 668-672 (1998).

93. Yannis Theodoridis, Timos K. Sellis: A Model for the Prediction of R-tree Performance. PODS 1996: 161-171.

94. Yannis Theodoridis, Emmanuel Stefanakis, Timos K. Sellis: Efficient Cost Models for Spatial Queries Using R-Trees. IEEE Transactions on Knowledge and Data Engineering 12(1): 19-32 (2000).

95. Michalis Vazirgiannis, Ouri Wolfson: A Spatiotemporal Model and Language for Moving Objects on Road Networks. SSTD 2001: 20-35.

96. Kirsi Virrantaus, Jouni Markkula, Artem Garmash, Vagan Y. Terziyan, Jari Veijalainen, Artem Katanosov, Henry Tirri: Developing GIS-Supported Location-Based Services. WISE (2) 2001: 66-75.

97. Mark A. Weiss, Data Structures and Algorithm Analysis in C, Addison-Wesley, 1997.

98. Jeffrey E. Wieselthier, Gam D. Nguyen, Anthony Ephremides: Energy-Aware Wireless Networking with Directional Antennas: The Case of Session-Based Broadcasting and Multicasting. IEEE Transactions on Mobile Computing 1(3): 176-191 (2002).

99. Ouri Wolfson, A. Prasad Sistla, Sam Chamberlain, Yelena Yesha: Updating and Querying Databases that Track Mobile Units. Distributed and Parallel Databases 7(3): 257-387 (1999).

100. Vincent W.-S. Wong, Victor C. M. Leung**:** Location management for next-generation personal communications networks**.** IEEE Network, 14(5): 18 –24(2000).

101. Sung-Ho Woo, Sung-Bong Yang: An improved network clustering method for I/O-efficient query processing. ACM-GIS, 2000:62-68.

102. Jitian Xiao, Yanchun Zhang, Xiaohua Jia, Xiaofang Zhou: A schedule of join operations to reduce I/O cost in spatial database systems. Data and Knowledge Engineering 35(3): 299-317 (2000).

103. Yonglong Xu: Development of Transport Telematics in Europe, GeoInformatica 4(2): 179-200 (2000).

104. 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).

105. Jianting Zhang, Le Gruenwald: Prioritized sequencing for efficient query on broadcast geographical information in mobile-computing. ACM-GIS 2002: 88-93.

106. J. Leon Zhao, Hsing K. Cheng: Graph Indexing for Spatial Data Traversal in Road Map Databases. Computers and Operations Research 28 (3): 223-241(2001).

107. Baihua Zheng, Dik Lun Lee: Processing location-dependent queries in a multi-cell wireless environment. MobiDE 2001: 54-65.

[HREF 1] The Proposed .geo Top-Level Domain Name executive summary, http://www.dotgeo.org/summary.html
[HREF 2] http://download-west.oracle.com/docs/cd/B10501_01/appdev.920/a96630/toc.htm
[HREF 3] Traffic & Travel Information Broadcasting, http://www.tpeg.org/
[HREF 4] http://www.nist.gov/dads/HTML/simulatedAnnealing.html
[HREF 5] http://www.boost.org/libs/graph/doc/index.html
[HREF 6] http://www-users.cs.umn.edu/~karypis/metis/hmetis/
[HREF 7] http://www.cs.ucr.edu/~marioh/spatialindex/index.html
[HREF 8] http://www.caam.rice.edu/~dougm/twiddle/Hilbert/

Table A-1. Hypergraph Parameters Under the Five Query Window Sizes for Zip Code Data Sets

Data Set	0.05*0.05			0.1*0.1			0.5*0.5			1.0*1.0			5.0*5.0		
	Node	Edge	ANPE	Node	Edge	ANPE	Node	Edge	ANPE	Node	Edge	ANPE	Node	Edge	ANPE
AK	109	36	3.03	109	36	3.03	115	79	5.71	130	137	8.18	250	860	10.03
AL	364	135	2.70	393	142	3.01	643	797	5.85	780	3230	6.27			
AR	200	82	2.44	200	82	2.44	517	468	3.03	694	3196	4.26	46	679	20.29
AZ	310	110	2.89	314	115	2.93	387	952	6.32	414	1813	12.16	305	2304	11.68
CA	1543	597	2.64	1707	745	2.98	1905	4628	6.63	1797	8451	8.45	393	4883	15.74
CO	316	115	2.75	331	120	2.83	422	816	6.33	527	1555	9.13	348	5903	12.55
CT	211	74	2.85	250	90	3.43	410	1511	6.03	380	3736	9.43			
DC	53	15	3.53	47	16	3.88									
DE	43	14	3.07	58	15	3.87	95	156	7.26	97	411	9.28	50	1026	20.71
FL	821	319	2.57	840	350	2.69	1242	4221	7.02	1125	6632	9.17	46	179	17.93
GA	417	162	2.57	434	167	2.87	697	926	5.48	841	3492	5.54	45	431	22.26
HI	60	22	2.73	57	21	2.71	89	130	3.49	96	328	5.60	68	1124	12.77
IA	238	99	2.40	235	101	2.35	802	885	3.41	1052	6656	3.96			
ID	113	43	2.63	129	44	2.93	187	98	3.23	281	427	4.44	292	6507	15.14
IL	532	207	2.62	591	240	2.81	1378	3780	5.53	1420	11143	5.86			
IN	432	164	2.63	455	171	2.78	904	1404	4.64	983	6343	6.21			
KS	135	55	2.45	138	59	2.85	393	1065	6.37	767	4421	9.15	185	4682	15.09
KY	504	193	2.61	550	205	2.83	1080	2699	4.19	1114	9616	8.04	46	362	22.06
LA	412	154	2.68	419	156	2.79	594	818	5.91	696	2882	6.91			
MA	354	137	2.78	377	183	2.92	643	3077	5.71	568	6206	9.68			
MD	289	111	2.62	317	126	2.90	578	2053	5.35	507	4596	8.16	36	424	14.09
ME	185	76	2.43	188	78	2.45	452	743	3.26	519	3995	5.69	137	1958	18.86
MI	428	177	2.42	470	189	2.70	942	2615	5.55	1075	6808	6.76	190	3418	17.74
MN	190	81	2.35	255	103	3.43	687	1373	4.99	1007	4596	6.31	229	4104	16.43
MO	319	125	2.67	342	149	3.05	894	1880	5.14	1132	7310	6.04	50	604	25.74
MS	271	98	2.77	273	102	2.90	381	297	5.72	531	1229	5.74	95	744	21.62
MT	145	52	2.79	146	53	2.81	158	66	2.86	259	206	2.92	409	7826	11.62

NC	521	204	2.55	546	208	2.75	901	964	4.25	1016	4848	5.81	90	1277	20.27
ND	61	22	2.77	62	23	2.78	129	70	2.71	390	635	2.73	328	7365	15.14
NE	104	43	2.42	104	43	2.42	342	894	6.85	565	3220	7.17	217	4951	12.66
NH	97	40	2.43	99	41	2.41	277	537	2.90	292	2859	5.62	44	531	23.48
NJ	300	120	2.58	433	228	2.66	657	5110	7.10	436	5131	11.08			
NM	231	85	2.72	244	88	2.81	304	257	6.53	379	746	10.02	402	4925	11.85
NV	145	48	3.02	145	48	3.02	169	232	7.69	178	299	10.74	127	461	11.24
NY	901	362	2.62	1140	703	3.16	1845	6149	5.51	1712	17056	7.65			
OH	748	304	2.49	797	337	2.64	1393	3915	6.32	1418	11404	8.07			
OK	336	138	2.43	336	150	2.81	502	840	6.96	757	2442	7.05	170	2868	19.04
OR	155	63	2.46	159	70	3.01	291	415	4.94	400	1235	4.92	303	5843	13.31
PA	1120	424	2.66	1312	561	3.06	2075	6862	5.62	1847	17447	8.69			
RI	38	17	2.24	41	20	2.40	90	533	6.89	90	1339	11.23			
SC	273	105	2.60	290	107	2.73	437	385	4.66	537	2108	6.49	48	454	23.71
SD	70	29	2.41	72	30	2.40	148	124	4.75	371	811	4.61	337	8495	14.70
TN	317	122	2.64	340	134	3.01	662	909	5.21	744	3956	5.91	48	631	23.56
TX	1227	488	2.51	1347	534	2.90	1837	4624	6.35	2167	9231	8.46	497	5921	15.15
UT	197	61	3.23	199	64	3.59	269	326	5.00	279	826	6.06	196	1718	12.97
VA	543	182	2.98	570	202	3.17	1082	2091	5.76	1106	7017	6.52	50	597	22.79
VT	109	44	2.48	113	47	2.49	307	464	2.76	314	2835	5.13			
WA	304	116	2.62	327	128	3.14	503	1163	5.71	633	2614	8.50	370	5706	16.81
WI	317	131	2.42	343	144	2.72	694	1056	4.82	893	4335	5.38	111	1554	19.97
WV	331	122	2.71	396	131	3.11	900	1850	3.52	928	8843	7.05	50	1017	23.58
WY	91	34	2.68	91	34	2.68	105	44	2.70	138	81	2.64	198	2098	7.77
Avg.	344	132	2.65	375	156	2.88	650	1547	5.21	728	4415	7.02	184	2822	17.04

Table A-2. Definitions of the Meanings of Columns Used in Table A3-A18

Rand-Min-1000	Minimum access time of 1000 random orderings
Rand-Max1000	Maximum access time of 1000 random orderings
Rand-Avg-1000	Average access time of 1000 random orderings
Hilbert	Access time of Hilbert space filling curve ordering
R-Tree	Access time of R-Tree traversal ordering
Optimized	Access time of Optimized ordering using R-Tree as BDT
Hilbert-Improv.	Improvement of the Hilbert ordering compared with 1000 random orderings average (Hilbert/Rand-Avg-1000)
R-Tree Improv.	Improvement of the R-Tree traversal ordering compared with 1000 random orderings average (R-Tree/ Rand-Avg-1000)
Opt Improv.	Improvement of the optimized ordering compared with the R-Tree traversal ordering (Optimized/ R-Tree)
R-Tree+Opt Improv.	Improvement of the optimized ordering compared with 1000 random orderings average (Optimized/ Rand-Avg-1000)

Table A-3. Results of DBW Under Query Window (0.05*0.05) for Zip Code Data Sets

	Rand-Min -1000	Rand-Max -1000	Rand-Avg -1000	Hilbert	R-Tree	Opt	Rand-Improv.	Hilbert-Improv.	R-tree-Improv.	Opt-Improv.	R-Tree+Opt--Improv.
AK	48.44	66.31	57.60	60.30	22.45	10.69	31.02%	-4.48%	1.57	1.10	4.39
AL	156.07	191.75	173.30	179.21	80.29	48.01	20.59%	-3.30%	1.16	0.67	2.61
AR	68.43	98.00	85.98	91.87	43.35	27.29	34.39%	-6.41%	0.98	0.59	2.15
AZ	136.01	171.21	154.91	160.27	69.16	55.30	22.72%	-3.34%	1.24	0.25	1.80
CA	675.02	747.45	711.71	698.85	207.46	172.72	10.18%	1.84%	2.43	0.20	3.12
CO	134.72	171.88	152.47	154.27	62.56	48.78	24.37%	-1.17%	1.44	0.28	2.13
CT	93.36	119.01	105.80	109.70	53.16	30.06	24.24%	-3.56%	0.99	0.77	2.52
DC	22.89	35.72	30.83	29.26	14.17	8.28	41.62%	5.37%	1.18	0.71	2.72
DE	14.37	28.07	23.13	23.37	10.28	6.88	59.23%	-1.03%	1.25	0.49	2.36
FL	345.06	399.22	371.33	363.69	65.49	49.13	14.59%	2.10%	4.67	0.33	6.56
GA	165.13	212.13	189.93	191.67	57.60	43.05	24.75%	-0.91%	2.30	0.34	3.41
HI	20.13	34.65	28.99	26.63	10.37	6.02	50.09%	8.86%	1.80	0.72	3.82
IA	86.92	115.64	102.09	105.24	39.33	25.75	28.13%	-2.99%	1.60	0.53	2.96
ID	40.96	61.12	52.42	49.52	31.70	18.64	38.46%	5.86%	0.65	0.70	1.81
IL	214.58	266.72	239.59	225.61	111.13	69.85	21.76%	6.20%	1.16	0.59	2.43
IN	180.38	218.71	201.14	205.79	83.28	48.83	19.06%	-2.26%	1.42	0.71	3.12
KS	46.99	70.45	59.49	56.95	20.01	10.15	39.44%	4.46%	1.97	0.97	4.86
KY	206.29	251.09	232.40	213.77	123.82	95.84	19.28%	8.71%	0.88	0.29	1.42
LA	174.96	213.47	194.36	183.50	63.50	44.28	19.81%	5.92%	2.06	0.43	3.39
MA	146.98	184.21	166.77	170.48	95.54	50.21	22.32%	-2.18%	0.75	0.90	2.32
MD	119.15	151.92	134.39	139.57	50.29	35.91	24.38%	-3.71%	1.67	0.40	2.74
ME	65.71	94.83	79.85	68.80	36.78	28.40	36.47%	16.06%	1.17	0.30	1.81
MI	163.73	202.90	182.43	185.69	67.80	43.94	21.47%	-1.76%	1.69	0.54	3.15
MN	64.50	91.96	78.44	76.79	20.90	14.61	35.01%	2.15%	2.75	0.43	4.37
MO	129.95	165.64	148.09	149.46	50.66	34.00	24.10%	-0.92%	1.92	0.49	3.36
MS	116.39	148.20	131.71	132.37	61.04	40.63	24.15%	-0.50%	1.16	0.50	2.24

MT	58.97	83.77	71.50	71.43	23.38	15.68	34.69%	0.10%	2.06	0.49	3.56
NC	22.03	36.41	30.10	26.75	11.74	8.00	47.77%	12.52%	1.56	0.47	2.76
ND	34.38	53.30	44.72	44.63	16.63	11.22	42.31%	0.20%	1.69	0.48	2.99
NE	210.12	257.74	236.14	236.58	77.66	66.74	20.17%	-0.19%	2.04	0.16	2.54
NH	32.04	49.29	41.72	41.40	16.10	10.79	41.35%	0.77%	1.59	0.49	2.87
NJ	122.26	152.99	137.68	132.14	95.14	56.12	22.32%	4.19%	0.45	0.70	1.45
NM	90.64	124.60	110.15	112.66	60.53	45.96	30.83%	-2.23%	0.82	0.32	1.40
NV	64.00	87.24	76.12	82.97	18.57	14.94	30.53%	-8.26%	3.10	0.24	4.10
NY	378.71	441.87	410.29	425.63	132.48	102.11	15.39%	-3.60%	2.10	0.30	3.02
OH	297.02	355.51	326.96	318.06	150.89	102.84	17.89%	2.80%	1.17	0.47	2.18
OK	128.07	160.95	144.63	145.29	54.98	39.03	22.73%	-0.45%	1.63	0.41	2.71
OR	56.65	82.25	67.87	69.97	24.34	17.21	37.72%	-3.00%	1.79	0.41	2.94
PA	484.33	558.74	523.50	508.99	191.64	146.91	14.21%	2.85%	1.73	0.30	2.56
RI	9.39	20.05	15.04	13.45	9.08	5.21	70.88%	11.82%	0.66	0.74	1.89
SC	106.51	139.33	124.99	123.15	43.94	31.20	26.26%	1.49%	1.84	0.41	3.01
SD	20.40	37.51	30.48	30.36	8.86	4.80	56.14%	0.40%	2.44	0.85	5.35
TN	129.62	163.97	147.90	153.96	65.62	41.05	23.23%	-3.94%	1.25	0.60	2.60
TX	506.39	571.12	542.37	541.71	154.03	95.03	11.93%	0.12%	2.52	0.62	4.71
UT	93.67	121.30	108.52	112.47	51.08	32.64	25.46%	-3.51%	1.12	0.57	2.32
VA	260.89	306.64	281.44	279.41	76.12	48.09	16.26%	0.73%	2.70	0.58	4.85
VT	38.66	56.88	48.16	47.03	27.61	16.22	37.83%	2.40%	0.74	0.70	1.97
WA	122.09	155.98	139.37	138.95	82.96	60.65	24.32%	0.30%	0.68	0.37	1.30
WI	118.66	156.58	135.55	147.86	71.96	49.61	27.97%	-8.33%	0.88	0.45	1.73
WV	142.30	172.81	158.74	166.93	81.68	45.92	19.22%	-4.91%	0.94	0.78	2.46
WY	30.79	51.47	43.23	30.08	12.92	10.21	47.84%	43.72%	2.35	0.27	3.23
Avg.	141.09	174.72	158.56	157.93	61.02	42.07	29.35%	1.47%	1.60	0.52	2.90

Table A-4. Results of DBW Under Query Window (0.1*0.1) for Zip Code Data Sets

	Rand-Min -1000	Rand-Max -1000	Rand-Avg -1000	Hilbert	R-Tree	Opt	Rand-Improv.	Hilbert-Improv.	R-tree-Improv.	Opt-Improv.	R-Tree+Opt--Improv.
AK	48.44	66.31	57.60	60.30	22.45	10.69	31.02%	-4.48%	1.57	1.10	4.39
AL	176.36	212.80	195.41	196.44	78.54	55.56	18.65%	-0.52%	1.49	0.41	2.52
AR	68.43	98.00	85.98	91.87	43.35	27.29	34.39%	-6.41%	0.98	0.59	2.15
AZ	139.25	172.97	157.57	166.21	66.23	55.27	21.40%	-5.20%	1.38	0.20	1.85
CA	768.71	849.32	812.33	802.73	265.47	213.28	9.92%	1.20%	2.06	0.24	2.81
CO	143.31	178.44	162.95	159.80	85.56	61.72	21.56%	1.97%	0.90	0.39	1.64
CT	111.26	146.52	131.00	131.42	65.57	46.47	26.92%	-0.32%	1.00	0.41	1.82
DC	18.00	31.31	25.58	26.86	13.65	7.69	52.03%	-4.77%	0.87	0.78	2.33
DE	26.40	42.09	36.41	23.60	16.64	12.41	43.09%	54.28%	1.19	0.34	1.93
FL	352.46	407.21	381.02	384.99	152.47	98.30	14.37%	-1.03%	1.50	0.55	2.88
GA	176.17	216.20	196.02	193.24	104.97	81.64	20.42%	1.44%	0.87	0.29	1.40
HI	20.25	33.89	27.65	26.19	6.44	4.02	49.33%	5.57%	3.29	0.60	5.88
IA	83.29	114.43	97.98	97.68	38.36	27.62	31.78%	0.31%	1.55	0.39	2.55
ID	54.91	75.81	66.57	65.90	39.24	32.56	31.40%	1.02%	0.70	0.21	1.04
IL	249.97	297.33	276.88	271.30	105.13	74.82	17.10%	2.06%	1.63	0.41	2.70
IN	197.05	241.46	218.73	211.66	73.98	47.62	20.30%	3.34%	1.96	0.55	3.59
KS	50.67	70.73	61.12	60.19	16.91	12.07	32.82%	1.55%	2.61	0.40	4.06
KY	241.52	290.67	266.20	272.64	113.43	77.16	18.46%	-2.36%	1.35	0.47	2.45
LA	175.15	216.07	195.37	190.62	83.39	58.76	20.94%	2.49%	1.34	0.42	2.32
MA	144.01	190.63	169.31	161.22	65.55	46.84	27.54%	5.02%	1.58	0.40	2.61
MD	136.48	170.30	153.20	157.53	91.68	51.49	22.08%	-2.75%	0.67	0.78	1.98
ME	68.36	91.99	80.83	75.66	32.76	19.50	29.23%	6.83%	1.47	0.68	3.15
MI	188.98	239.10	214.21	212.72	65.88	49.60	23.40%	0.70%	2.25	0.33	3.32
MN	114.40	145.62	131.08	134.17	74.77	49.01	23.82%	-2.30%	0.75	0.53	1.67
MO	140.98	174.97	159.19	162.54	92.03	73.73	21.35%	-2.06%	0.73	0.25	1.16
MS	117.00	146.89	133.01	136.33	64.84	45.21	22.47%	-2.44%	1.05	0.43	1.94

MT	57.84	83.13	71.93	72.42	24.34	15.99	35.16%	-0.68%	1.96	0.52	3.50
NC	234.51	279.56	257.11	258.74	132.56	106.89	17.52%	-0.63%	0.94	0.24	1.41
ND	22.67	37.30	30.52	27.64	12.37	9.25	47.94%	10.42%	1.47	0.34	2.30
NE	34.38	53.30	44.72	44.63	16.63	11.22	42.31%	0.20%	1.69	0.48	2.99
NH	32.38	50.22	42.28	42.62	15.89	7.72	42.19%	-0.80%	1.66	1.06	4.48
NJ	183.43	233.06	205.18	200.67	93.17	79.05	24.19%	2.25%	1.20	0.18	1.60
NM	105.87	135.35	120.31	112.71	55.66	38.40	24.50%	6.74%	1.16	0.45	2.13
NV	64.00	87.24	76.12	82.97	18.57	14.94	30.53%	-8.26%	3.10	0.24	4.10
NY	486.21	554.90	522.27	523.88	208.76	163.38	13.15%	-0.31%	1.50	0.28	2.20
OH	330.36	381.00	356.34	338.11	166.80	134.44	14.21%	5.39%	1.14	0.24	1.65
OK	128.20	161.86	145.86	148.64	60.12	43.25	23.08%	-1.87%	1.43	0.39	2.37
OR	57.43	80.80	70.85	73.47	24.60	17.14	32.99%	-3.57%	1.88	0.44	3.13
PA	607.16	674.88	642.25	659.88	250.54	206.76	10.54%	-2.67%	1.56	0.21	2.11
RI	8.98	21.42	16.22	18.91	6.63	2.97	76.70%	-14.23%	1.45	1.23	4.46
SC	125.39	155.20	139.36	136.36	56.91	39.76	21.39%	2.20%	1.45	0.43	2.51
SD	22.58	37.61	31.37	28.47	11.83	9.71	47.91%	10.19%	1.65	0.22	2.23
TN	136.12	175.95	156.67	154.60	84.40	48.97	25.42%	1.34%	0.86	0.72	2.20
TX	602.04	665.53	634.37	643.69	380.21	299.13	10.01%	-1.45%	0.67	0.27	1.12
UT	94.19	121.28	110.08	105.59	59.50	39.65	24.61%	4.25%	0.85	0.50	1.78
VA	273.26	320.31	299.00	304.33	146.69	98.26	15.74%	-1.75%	1.04	0.49	2.04
VT	37.64	58.58	49.96	50.36	14.61	8.38	41.91%	-0.79%	2.42	0.74	4.96
WA	139.45	171.44	156.83	158.42	66.57	49.51	20.40%	-1.00%	1.36	0.34	2.17
WI	131.62	169.93	153.91	144.16	70.55	59.42	24.89%	6.76%	1.18	0.19	1.59
WV	191.71	232.39	211.47	206.97	99.47	82.02	19.24%	2.17%	1.13	0.21	1.58
WY	30.79	51.47	43.23	30.08	12.92	10.21	47.84%	43.72%	2.35	0.27	3.23
Avg.	159.80	194.41	178.07	177.30	77.84	57.58	27.85%	2.17%	1.45	0.45	2.55

Table A-5. Results of DBW Under Query Window (0.5*0.5) for Zip Code Data Sets

	Rand-Min -1000	Rand-Max -1000	Rand-Avg -1000	Hilbert	R-Tree	Opt	Rand-Improv.	Hilbert-Improv.	R-tree-Improv.	Opt-Improv.	R-Tree+Opt--Improv.
AK	52.97	73.50	64.19	65.14	19.61	15.41	31.98%	-1.46%	2.27	0.27	3.17
AL	284.42	353.00	317.62	324.39	129.06	82.91	21.59%	-2.09%	1.46	0.56	2.83
AR	195.68	261.63	227.09	234.41	88.93	65.46	29.04%	-3.12%	1.55	0.36	2.47
AZ	197.52	235.69	216.94	215.53	69.30	57.59	17.59%	0.65%	2.13	0.20	2.77
CA	971.52	1067.38	1027.06	1039.00	480.89	378.83	9.33%	-1.15%	1.14	0.27	1.71
CO	196.79	243.05	222.38	224.22	101.71	73.96	20.80%	-0.82%	1.19	0.38	2.01
CT	231.08	263.76	246.93	241.27	131.16	101.90	13.23%	2.35%	0.88	0.29	1.42
DC											
DE	37.70	65.03	54.66	26.33	20.65	14.45	50.00%	107.60%	1.65	0.43	2.78
FL	672.49	755.95	716.29	734.65	317.04	234.60	11.65%	-2.50%	1.26	0.35	2.05
GA	297.57	355.36	329.09	311.87	140.98	97.69	17.56%	5.52%	1.33	0.44	2.37
HI	28.90	49.25	40.12	41.85	10.67	5.86	50.72%	-4.13%	2.76	0.82	5.85
IA	295.20	370.71	331.37	308.70	183.64	130.39	22.79%	7.34%	0.80	0.41	1.54
ID	81.25	107.74	95.77	88.83	46.43	24.68	27.66%	7.81%	1.06	0.88	2.88
IL	631.23	724.62	676.05	677.34	240.02	195.20	13.81%	-0.19%	1.82	0.23	2.46
IN	408.95	474.94	443.88	437.35	252.16	203.20	14.87%	1.49%	0.76	0.24	1.18
KS	180.92	225.77	204.57	199.54	147.61	83.88	21.92%	2.52%	0.39	0.76	1.44
KY	498.60	571.41	538.49	533.11	238.94	164.52	13.52%	1.01%	1.25	0.45	2.27
LA	270.78	327.96	301.55	301.83	128.94	110.73	18.96%	-0.09%	1.34	0.16	1.72
MA	336.29	376.36	357.63	339.57	227.55	205.97	11.20%	5.32%	0.57	0.10	0.74
MD	292.39	338.90	314.80	312.82	153.95	119.29	14.77%	0.63%	1.04	0.29	1.64
ME	188.25	228.60	211.02	214.75	100.16	72.68	19.12%	-1.74%	1.11	0.38	1.90
MI	444.99	514.84	482.82	480.69	262.97	208.85	14.47%	0.44%	0.84	0.26	1.31
MN	319.61	389.41	353.91	327.91	133.38	78.83	19.72%	7.93%	1.65	0.69	3.49
MO	371.18	439.97	407.77	408.35	148.83	87.72	16.87%	-0.14%	1.74	0.70	3.65

MS	173.54	212.42	193.51	191.66	92.19	78.75	20.09%	0.97%	1.10	0.17	1.46
MT	64.73	89.39	77.88	76.18	24.93	18.07	31.66%	2.23%	2.12	0.38	3.31
NC	403.14	469.04	436.44	438.82	266.71	185.27	15.10%	-0.54%	0.64	0.44	1.36
ND	47.75	77.49	63.28	61.48	30.51	16.02	47.00%	2.93%	1.07	0.90	2.95
NE	142.16	185.26	167.72	150.05	94.37	56.72	25.70%	11.78%	0.78	0.66	1.96
NH	107.00	138.83	123.00	122.92	65.51	51.96	25.88%	0.07%	0.88	0.26	1.37
NJ	371.87	416.24	396.26	390.99	167.37	127.26	11.20%	1.35%	1.37	0.32	2.11
NM	136.45	172.30	154.44	152.84	49.96	40.36	23.21%	1.05%	2.09	0.24	2.83
NV	77.88	104.63	93.26	99.10	39.77	29.70	28.68%	-5.89%	1.35	0.34	2.14
NY	863.52	948.15	907.38	888.53	368.01	187.46	9.33%	2.12%	1.47	0.96	3.84
OH	636.94	709.68	670.91	669.16	330.05	271.67	10.84%	0.26%	1.03	0.21	1.47
OK	189.44	249.98	225.48	225.53	85.04	69.44	26.85%	-0.02%	1.65	0.22	2.25
OR	102.82	150.24	127.70	125.26	51.82	33.56	37.13%	1.95%	1.46	0.54	2.81
PA	1107.13	1195.13	1147.08	1141.64	591.92	540.70	7.67%	0.48%	0.94	0.09	1.12
RI	45.05	58.67	53.14	52.23	32.58	21.56	25.63%	1.74%	0.63	0.51	1.46
SC	197.19	253.57	220.78	223.55	90.45	78.80	25.54%	-1.24%	1.44	0.15	1.80
SD	55.11	82.42	69.33	69.57	37.23	27.11	39.39%	-0.34%	0.86	0.37	1.56
TN	278.63	346.52	314.72	300.89	100.88	60.00	21.57%	4.60%	2.12	0.68	4.25
TX	912.25	1004.69	960.25	972.93	451.01	349.98	9.63%	-1.30%	1.13	0.29	1.74
UT	119.70	158.42	141.54	142.29	69.32	42.63	27.36%	-0.53%	1.04	0.63	2.32
VA	524.28	590.48	558.12	560.57	267.00	193.24	11.86%	-0.44%	1.09	0.38	1.89
VT	121.52	155.62	138.26	151.08	48.89	40.01	24.66%	-8.49%	1.83	0.22	2.46
WA	233.43	281.78	258.78	250.87	139.48	91.27	18.68%	3.15%	0.86	0.53	1.84
WI	271.32	339.78	306.78	292.17	126.09	87.53	22.32%	5.00%	1.43	0.44	2.50
WV	439.08	504.91	473.52	457.67	276.35	196.85	13.90%	3.46%	0.71	0.40	1.41
WY	37.09	58.95	49.85	39.94	17.03	8.39	43.85%	24.81%	1.93	1.03	4.94
Avg.	302.91	355.39	330.23	326.75	153.78	114.38	22.16%	3.65%	1.30	0.43	2.30

Table A-6. Results of DBW Under Query Window (1.0*1.0) for Zip Code Data Sets

	Rand-Min -1000	Rand-Max -1000	Rand-Avg -1000	Hilbert	R-Tree	Opt	Rand-Improv.	Hilbert-Improv.	R-tree-Improv.	Opt-Improv.	R-Tree+Opt--Improv.
AK	63.63	84.47	75.40	78.23	42.92	26.52	27.64%	-3.62%	0.76	0.62	1.84
AL	396.83	441.29	421.83	426.41	257.10	227.65	10.54%	-1.07%	0.64	0.13	0.85
AR	327.28	372.47	351.65	353.46	166.60	114.75	12.85%	-0.51%	1.11	0.45	2.06
AZ	209.57	263.10	240.25	236.61	150.41	106.25	22.28%	1.54%	0.60	0.42	1.26
CA	1008.75	1087.33	1051.61	1065.39	382.50	338.12	7.47%	-1.29%	1.75	0.13	2.11
CO	256.99	308.43	285.55	303.43	144.76	126.28	18.01%	-5.89%	0.97	0.15	1.26
CT	263.48	302.81	284.27	269.78	210.72	167.92	13.84%	5.37%	0.35	0.25	0.69
DC											
DE	55.66	70.62	63.27	61.24	41.96	33.06	23.64%	3.31%	0.51	0.27	0.91
FL	710.43	770.15	742.47	732.95	439.94	367.45	8.04%	1.30%	0.69	0.20	1.02
GA	419.76	467.68	444.91	441.73	179.47	142.08	10.77%	0.72%	1.48	0.26	2.13
HI	42.64	60.77	52.17	51.97	19.37	13.42	34.75%	0.38%	1.69	0.44	2.89
IA	515.91	556.94	536.85	539.69	272.41	236.14	7.64%	-0.53%	0.97	0.15	1.27
ID	130.44	164.13	148.36	151.10	88.34	67.08	22.71%	-1.81%	0.68	0.32	1.21
IL	787.16	835.90	811.66	813.48	387.80	323.27	6.00%	-0.22%	1.09	0.20	1.51
IN	579.82	628.31	601.23	603.96	265.93	227.46	8.07%	-0.45%	1.26	0.17	1.64
KS	317.22	362.26	338.35	364.37	134.51	113.00	13.31%	-7.14%	1.52	0.19	1.99
KY	671.97	724.03	699.24	697.09	345.99	286.59	7.45%	0.31%	1.02	0.21	1.44
LA	365.09	417.63	394.10	403.24	238.81	190.63	13.33%	-2.27%	0.65	0.25	1.07
MA	399.30	436.38	417.93	413.15	125.18	109.16	8.87%	1.16%	2.34	0.15	2.83
MD	326.53	367.54	346.51	352.38	181.13	138.11	11.84%	-1.67%	0.91	0.31	1.51
ME	301.64	331.29	316.87	324.69	178.17	137.45	9.36%	-2.41%	0.78	0.30	1.31
MI	587.02	636.29	613.49	618.92	282.59	237.01	8.03%	-0.88%	1.17	0.19	1.59
MN	464.92	517.43	493.14	488.90	290.79	243.38	10.65%	0.87%	0.70	0.19	1.03
MO	544.52	596.26	570.92	564.76	343.23	273.06	9.06%	1.09%	0.66	0.26	1.09
MS	251.96	295.00	274.65	286.93	150.94	123.79	15.67%	-4.28%	0.82	0.22	1.22

MT	107.92	144.34	127.14	130.99	68.96	47.50	28.65%	-2.94%	0.84	0.45	1.68
NC	554.46	602.34	577.78	568.39	323.77	228.35	8.29%	1.65%	0.78	0.42	1.53
ND	149.25	193.50	169.72	167.84	108.56	91.17	26.07%	1.12%	0.56	0.19	0.86
NE	247.89	280.78	266.19	247.07	153.03	109.78	12.36%	7.74%	0.74	0.39	1.42
NH	176.86	198.71	187.90	184.02	141.76	107.98	11.63%	2.11%	0.33	0.31	0.74
NJ	286.97	333.72	314.94	308.55	210.28	134.80	14.84%	2.07%	0.50	0.56	1.34
NM	169.52	214.06	196.07	193.54	101.91	58.31	22.72%	1.31%	0.92	0.75	2.36
NV	82.97	115.27	99.96	99.95	44.48	33.71	32.31%	0.01%	1.25	0.32	1.97
NY	1086.11	1153.78	1117.40	1113.09	674.70	521.48	6.06%	0.39%	0.66	0.29	1.14
OH	851.90	915.17	883.49	871.84	553.29	446.11	7.16%	1.34%	0.60	0.24	0.98
OK	322.91	371.44	348.74	352.27	230.49	179.04	13.92%	-1.00%	0.51	0.29	0.95
OR	168.19	209.94	188.82	187.03	104.05	75.71	22.11%	0.96%	0.81	0.37	1.49
PA	1240.22	1308.13	1276.63	1280.00	738.48	528.05	5.32%	-0.26%	0.73	0.40	1.42
RI	56.55	72.33	65.15	65.15	33.69	24.06	24.22%	0.00%	0.93	0.40	1.71
SC	295.30	329.29	314.07	309.31	176.42	146.61	10.82%	1.54%	0.78	0.20	1.14
SD	142.20	181.31	162.82	168.15	107.59	77.34	24.02%	-3.17%	0.51	0.39	1.11
TN	385.59	426.19	404.01	405.35	191.22	148.73	10.05%	-0.33%	1.11	0.29	1.72
TX	1105.12	1214.36	1167.14	1159.44	559.64	446.74	9.36%	0.66%	1.09	0.25	1.61
UT	139.71	173.35	158.74	163.86	90.89	61.71	21.19%	-3.12%	0.75	0.47	1.57
VA	638.38	694.00	669.65	661.07	394.32	331.84	8.31%	1.30%	0.70	0.19	1.02
VT	188.68	209.96	199.17	197.31	124.53	102.35	10.68%	0.94%	0.60	0.22	0.95
WA	317.11	366.79	343.70	331.80	162.59	126.32	14.45%	3.59%	1.11	0.29	1.72
WI	432.96	488.04	459.89	468.46	296.62	217.36	11.98%	-1.83%	0.55	0.36	1.12
WV	579.32	634.41	609.72	611.47	302.79	260.28	9.04%	-0.29%	1.01	0.16	1.34
WY	52.46	77.74	65.29	57.91	31.78	23.16	38.72%	12.74%	1.05	0.37	1.82
Avg.	395.54	440.15	419.02	418.95	224.95	177.96	14.92%	0.17%	0.89	0.30	1.45

Table A-7. Results of DBW Under Query Window (5.0*5.0) for Zip Code Data Sets

	Rand-Min -1000	Rand-Max -1000	Rand-Avg -1000	Hilbert	R-Tree	Opt	Rand-Improv.	Hilbert-Improv.	R-tree-Improv.	Opt-Improv.	R-Tree+Opt--Improv.
AK	136.87	163.87	150.52	155.15	78.41	41.28	17.94%	-2.98%	0.92	0.90	2.65
AL											
AR	30.18	43.60	37.82	34.72	26.77	23.68	35.48%	8.93%	0.41	0.13	0.60
AZ	214.58	246.46	233.51	233.47	163.09	82.70	13.65%	0.02%	0.43	0.97	1.82
CA	258.62	315.86	288.84	300.13	170.54	103.66	19.82%	-3.76%	0.69	0.65	1.79
CO	256.64	290.28	277.07	274.60	206.36	143.29	12.14%	0.90%	0.34	0.44	0.93
CT											
DC											
DE	38.33	47.69	44.12	38.64	37.33	28.54	21.21%	14.18%	0.18	0.31	0.55
FL	18.83	43.82	34.45	39.22	22.23	11.37	72.54%	-12.16%	0.55	0.96	2.03
GA	29.41	43.11	36.86	39.01	27.90	23.85	37.17%	-5.51%	0.32	0.17	0.55
HI	51.35	60.26	57.56	56.71	38.79	35.43	15.48%	1.50%	0.48	0.09	0.62
IA											
ID	219.08	244.77	232.40	236.60	139.71	90.31	11.05%	-1.78%	0.66	0.55	1.57
IL											
IN											
KS	130.24	160.38	146.21	141.51	112.64	64.00	20.61%	3.32%	0.30	0.76	1.28
KY	30.19	43.71	36.98	40.18	33.95	28.30	36.56%	-7.96%	0.09	0.20	0.31
LA											
MA											
MD	29.41	33.91	32.60	31.83	27.95	26.83	13.80%	2.42%	0.17	0.04	0.22
ME	100.07	123.15	113.99	117.29	91.33	51.78	20.25%	-2.81%	0.25	0.76	1.20
MI	140.74	167.19	155.80	160.82	102.51	88.39	16.98%	-3.12%	0.52	0.16	0.76
MN	151.63	191.43	173.57	186.78	91.08	69.28	22.93%	-7.07%	0.91	0.31	1.51
MO	25.93	48.33	36.72	37.48	28.39	18.54	61.00%	-2.03%	0.29	0.53	0.98
MS	61.86	92.15	78.85	81.39	68.50	39.86	38.41%	-3.12%	0.15	0.72	0.98
MT	297.69	324.36	312.13	315.42	227.66	174.70	8.54%	-1.04%	0.37	0.30	0.79

NC	57.42	84.72	73.75	74.11	53.87	31.67	37.02%	-0.49%	0.37	0.70	1.33
ND	237.45	275.60	258.28	252.75	235.03	118.66	14.77%	2.19%	0.10	0.98	1.18
NE	153.29	182.99	170.71	167.40	95.85	79.52	17.40%	1.98%	0.78	0.21	1.15
NH	27.24	42.83	36.11	32.93	32.98	29.01	43.17%	9.66%	0.09	0.14	0.24
NJ											
NM	300.01	339.36	320.13	313.03	218.53	180.36	12.29%	2.27%	0.46	0.21	0.78
NV	52.41	80.27	67.94	63.06	32.86	27.71	41.01%	7.74%	1.07	0.19	1.45
NY											
OH											
OK	102.40	142.52	125.41	108.74	53.68	34.59	31.99%	15.33%	1.34	0.55	2.63
OR	212.85	248.37	232.56	218.94	141.27	114.49	15.27%	6.22%	0.65	0.23	1.03
PA											
RI											
SC	29.88	46.80	38.99	35.33	35.56	21.73	43.40%	10.36%	0.10	0.64	0.79
SD	251.76	283.55	269.83	275.59	236.90	164.00	11.78%	-2.09%	0.14	0.44	0.65
TN	30.66	45.39	39.20	37.52	32.15	25.12	37.58%	4.48%	0.22	0.28	0.56
TX	334.06	398.70	370.27	351.09	187.91	152.77	17.46%	5.46%	0.97	0.23	1.42
UT	135.67	160.98	149.26	147.56	91.10	75.91	16.96%	1.15%	0.64	0.20	0.97
VA	28.72	47.27	39.24	36.98	26.69	19.34	47.27%	6.11%	0.47	0.38	1.03
VT											
WA	270.59	319.65	296.77	290.20	165.79	130.36	16.53%	2.26%	0.79	0.27	1.28
WI	75.97	99.29	88.61	90.44	71.28	40.68	26.32%	-2.02%	0.24	0.75	1.18
WV	37.68	48.28	44.63	47.22	46.91	30.15	23.75%	-5.48%	-0.05	0.56	0.48
WY	134.71	151.57	144.14	147.12	74.32	66.64	11.70%	-2.03%	0.94	0.12	1.16
Avg.	126.88	153.58	141.78	140.84	95.35	67.26	25.98%	1.11%	0.47	0.43	1.09

Table A-8. Results of AT_{Data}^{Sep} Under Query Window (0.05*0.05) for Zip Code Data Sets

	Rand-Min -1000	Rand-Max -1000	Rand-Avg -1000	Hilbert	R-Tree	Opt	Rand-Improv.	Hilbert-Improv.	R-tree-Improv.	Opt-Improv.	R-Tree+Opt--Improv.
AK	88.16	97.79	93.15	94.76	70.97	64.25	10.34%	-1.70%	31.25%	10.46%	44.98%
AL	291.75	311.51	301.31	305.43	240.04	221.69	6.56%	-1.35%	25.52%	8.28%	35.92%
AR	151.46	169.26	161.37	165.57	131.23	122.90	11.03%	-2.54%	22.97%	6.78%	31.30%
AZ	250.91	268.79	260.29	262.93	209.83	200.03	6.87%	-1.00%	24.05%	4.90%	30.13%
CA	1243.99	1287.36	1266.48	1261.68	930.88	907.63	3.42%	0.38%	36.05%	2.56%	39.54%
CO	254.59	273.78	263.31	264.30	207.45	197.64	7.29%	-0.37%	26.93%	4.96%	33.23%
CT	170.77	184.76	177.47	178.84	145.41	130.98	7.88%	-0.77%	22.05%	11.02%	35.49%
DC	43.36	49.57	47.06	45.99	37.62	34.09	13.20%	2.33%	25.09%	10.35%	38.05%
DE	32.63	39.57	37.20	37.96	29.79	27.82	18.66%	-2.00%	24.87%	7.08%	33.72%
FL	655.79	688.61	670.95	667.73	466.75	453.85	4.89%	0.48%	43.75%	2.84%	47.84%
GA	328.27	353.65	340.93	341.27	255.94	244.03	7.44%	-0.10%	33.21%	4.88%	39.71%
HI	45.63	53.53	50.26	48.93	38.65	35.79	15.72%	2.72%	30.04%	7.99%	40.43%
IA	182.31	199.29	190.86	192.93	149.59	141.07	8.90%	-1.07%	27.59%	6.04%	35.29%
ID	86.83	98.16	93.28	91.16	79.11	72.12	12.15%	2.33%	17.91%	9.69%	29.34%
IL	419.12	448.18	433.20	425.77	347.07	322.36	6.71%	1.75%	24.82%	7.67%	34.38%
IN	344.80	364.99	355.63	358.68	276.96	257.94	5.68%	-0.85%	28.40%	7.37%	37.87%
KS	102.63	114.70	109.12	108.37	83.34	76.68	11.06%	0.69%	30.93%	8.69%	42.31%
KY	398.49	423.73	413.51	402.69	340.67	324.68	6.10%	2.69%	21.38%	4.92%	27.36%
LA	330.82	352.06	340.47	335.87	253.84	240.85	6.24%	1.37%	34.13%	5.39%	41.36%
MA	280.78	301.86	291.78	293.75	242.71	217.29	7.22%	-0.67%	20.22%	11.70%	34.28%
MD	229.78	247.51	237.62	240.57	182.28	173.20	7.46%	-1.23%	30.36%	5.24%	37.19%
ME	140.61	157.37	149.20	142.29	120.18	116.10	11.23%	4.86%	24.15%	3.51%	28.51%
MI	333.15	355.71	343.52	346.30	266.59	250.91	6.57%	-0.80%	28.86%	6.25%	36.91%
MN	142.43	159.20	151.34	150.16	111.58	107.78	11.08%	0.79%	35.63%	3.53%	40.42%
MO	251.43	271.16	261.60	261.66	196.36	187.41	7.54%	-0.02%	33.22%	4.78%	39.59%
MS	218.45	234.36	225.96	227.29	181.17	168.52	7.04%	-0.59%	24.72%	7.51%	34.08%

MT	113.89	127.76	121.38	121.69	91.01	85.78	11.43%	-0.25%	33.37%	6.10%	41.50%
NC	408.24	437.18	425.01	424.74	321.31	314.58	6.81%	0.06%	32.27%	2.14%	35.10%
ND	46.54	54.49	51.34	48.54	40.50	37.46	15.49%	5.77%	26.77%	8.12%	37.05%
NE	77.68	88.79	83.86	83.87	65.95	62.11	13.25%	-0.01%	27.16%	6.18%	35.02%
NH	73.39	82.77	78.49	78.90	61.86	58.05	11.95%	-0.52%	26.88%	6.56%	35.21%
NJ	235.85	255.22	245.85	244.65	216.60	194.67	7.88%	0.49%	13.50%	11.27%	26.29%
NM	180.26	199.37	191.97	192.85	160.86	151.07	9.95%	-0.46%	19.34%	6.48%	27.07%
NV	116.80	129.92	123.53	127.25	89.08	86.16	10.62%	-2.92%	38.67%	3.39%	43.37%
NY	718.48	754.03	736.26	744.07	554.56	531.70	4.83%	-1.05%	32.76%	4.30%	38.47%
OH	587.50	619.68	603.94	599.20	484.69	455.30	5.33%	0.79%	24.60%	6.46%	32.65%
OK	261.24	279.61	270.26	269.39	210.30	199.22	6.80%	0.32%	28.51%	5.56%	35.66%
OR	119.10	132.54	125.38	127.14	97.68	92.72	10.72%	-1.38%	28.36%	5.35%	35.22%
PA	900.00	940.30	923.15	917.44	704.95	675.58	4.37%	0.62%	30.95%	4.35%	36.65%
RI	26.60	33.02	30.26	29.83	26.64	23.85	21.22%	1.44%	13.59%	11.70%	26.88%
SC	212.86	232.43	224.17	222.95	170.12	161.16	8.73%	0.55%	31.77%	5.56%	39.10%
SD	50.63	60.52	56.46	56.02	42.83	39.78	17.52%	0.79%	31.82%	7.67%	41.93%
TN	251.26	270.56	260.94	265.29	207.87	191.17	7.40%	-1.64%	25.53%	8.74%	36.50%
TX	975.40	1010.49	995.63	996.54	735.60	695.44	3.52%	-0.09%	35.35%	5.77%	43.17%
UT	163.31	176.63	170.28	173.20	134.81	124.22	7.82%	-1.69%	26.31%	8.53%	37.08%
VA	448.62	475.25	460.86	461.13	330.60	311.66	5.78%	-0.06%	39.40%	6.08%	47.87%
VT	82.34	93.87	88.53	88.72	74.31	68.35	13.02%	-0.21%	19.14%	8.72%	29.52%
WA	240.13	258.50	249.95	251.43	210.82	198.83	7.35%	-0.59%	18.56%	6.03%	25.71%
WI	243.33	266.61	254.60	263.69	210.51	196.79	9.14%	-3.45%	20.94%	6.97%	29.38%
WV	265.14	283.54	274.81	279.28	223.27	204.40	6.70%	-1.60%	23.08%	9.23%	34.45%
WY	68.53	80.04	75.63	68.31	56.43	54.59	15.22%	10.72%	34.02%	3.37%	38.54%
Avg.	272.28	291.17	282.15	282.14	218.02	206.12	9.24%	0.21%	27.66%	6.65%	36.05%

Table A-9. Results of AT_{Data}^{Sep} Under Query Window (0.1*0.1) for Zip Code Data Sets

	Rand-Min -1000	Rand-Max -1000	Rand-Avg -1000	Hilbert	R-Tree	Opt	Rand-Improv.	Hilbert-Improv.	R-tree-Improv.	Opt-Improv.	R-Tree+Opt--Improv.
AK	88.17	97.79	93.15	94.76	70.97	64.25	10.33%	-1.70%	31.25%	10.46%	44.98%
AL	319.13	337.83	328.71	329.13	254.15	238.77	5.69%	-0.13%	29.34%	6.44%	37.67%
AR	151.46	169.26	161.37	165.57	131.23	122.89	11.03%	-2.54%	22.97%	6.79%	31.31%
AZ	255.09	272.28	263.93	269.50	209.27	201.68	6.51%	-2.07%	26.12%	3.76%	30.87%
CA	1389.19	1433.63	1411.41	1403.64	1052.91	1019.20	3.15%	0.55%	34.05%	3.31%	38.48%
CO	266.61	285.14	277.06	275.57	227.94	211.71	6.69%	0.54%	21.55%	7.67%	30.87%
CT	203.20	222.40	212.40	212.42	171.71	159.21	9.04%	-0.01%	23.70%	7.85%	33.41%
DC	37.14	43.79	40.98	41.58	34.02	30.41	16.23%	-1.44%	20.46%	11.87%	34.76%
DE	47.11	54.37	51.96	45.47	41.11	38.32	13.97%	14.27%	26.39%	7.28%	35.59%
FL	671.20	703.97	686.96	689.29	532.30	501.57	4.77%	-0.34%	29.06%	6.13%	36.96%
GA	342.22	364.35	353.94	352.30	290.79	277.54	6.25%	0.47%	21.72%	4.77%	27.53%
HI	43.73	51.04	47.79	46.99	34.70	32.66	15.30%	1.70%	37.72%	6.25%	46.33%
IA	178.10	197.06	187.11	187.49	147.21	140.23	10.13%	-0.20%	27.10%	4.98%	33.43%
ID	102.76	113.98	109.10	109.12	94.37	89.52	10.28%	-0.02%	15.61%	5.42%	21.87%
IL	471.37	498.20	485.80	483.93	373.42	355.33	5.52%	0.39%	30.09%	5.09%	36.72%
IN	366.07	390.27	377.20	371.84	281.48	266.32	6.42%	1.44%	34.01%	5.69%	41.63%
KS	105.76	117.02	111.56	112.15	83.77	79.89	10.09%	-0.53%	33.17%	4.86%	39.64%
KY	443.97	468.99	456.22	458.09	355.38	337.15	5.48%	-0.41%	28.38%	5.41%	35.32%
LA	334.63	356.79	345.06	342.42	271.84	256.97	6.42%	0.77%	26.93%	5.79%	34.28%
MA	293.59	318.42	307.10	301.18	240.97	227.16	8.09%	1.97%	27.44%	6.08%	35.19%
MD	253.67	272.86	263.05	265.93	220.90	200.15	7.30%	-1.08%	19.08%	10.37%	31.43%
ME	143.76	158.72	151.43	148.33	119.99	111.08	9.88%	2.09%	26.20%	8.02%	36.33%
MI	370.20	395.99	382.99	382.63	287.78	277.32	6.73%	0.09%	33.08%	3.77%	38.10%
MN	204.79	222.76	213.97	215.46	180.67	165.39	8.40%	-0.69%	18.43%	9.24%	29.37%
MO	270.18	290.91	280.47	281.87	237.64	223.74	7.39%	-0.50%	18.02%	6.21%	25.36%
MS	218.53	235.86	227.80	230.44	183.22	171.49	7.61%	-1.15%	24.33%	6.84%	32.84%

MT	114.83	128.38	122.21	122.71	92.24	86.58	11.09%	-0.41%	32.49%	6.54%	41.15%
NC	436.82	461.55	449.50	451.00	369.50	354.09	5.50%	-0.33%	21.65%	4.35%	26.95%
ND	46.95	55.54	52.14	49.91	41.13	39.21	16.47%	4.47%	26.77%	4.90%	32.98%
NE	77.68	88.79	83.86	83.87	65.95	62.11	13.25%	-0.01%	27.16%	6.18%	35.02%
NH	73.54	84.99	79.97	79.87	62.91	56.83	14.32%	0.13%	27.12%	10.70%	40.72%
NJ	344.49	371.13	357.27	354.62	286.96	278.94	7.46%	0.75%	24.50%	2.88%	28.08%
NM	196.64	212.35	204.44	200.70	162.83	153.08	7.68%	1.86%	25.55%	6.37%	33.55%
NV	116.80	129.92	123.53	127.25	89.08	86.16	10.62%	-2.92%	38.67%	3.39%	43.37%
NY	914.76	950.19	932.51	932.00	724.40	693.61	3.80%	0.05%	28.73%	4.44%	34.44%
OH	631.81	661.29	646.87	635.04	525.68	504.20	4.56%	1.86%	23.05%	4.26%	28.30%
OK	259.68	279.99	270.86	272.12	213.43	203.02	7.50%	-0.46%	26.91%	5.13%	33.42%
OR	121.76	134.96	129.04	130.21	99.48	94.71	10.23%	-0.90%	29.71%	5.04%	36.25%
PA	1070.17	1111.18	1093.13	1102.59	858.83	828.54	3.75%	-0.86%	27.28%	3.66%	31.93%
RI	28.20	35.68	32.64	34.02	26.48	23.70	22.92%	-4.06%	23.26%	11.73%	37.72%
SC	232.55	248.22	240.72	240.63	186.91	177.62	6.51%	0.04%	28.79%	5.23%	35.53%
SD	53.14	62.10	58.08	55.59	45.89	44.36	15.43%	4.48%	26.56%	3.45%	30.93%
TN	267.82	290.81	278.82	278.60	227.01	208.18	8.25%	0.08%	22.82%	9.05%	33.93%
TX	1092.34	1126.74	1109.37	1115.71	943.46	904.75	3.10%	-0.57%	17.59%	4.28%	22.62%
UT	165.11	178.53	172.07	170.53	141.08	131.18	7.80%	0.90%	21.97%	7.55%	31.17%
VA	471.72	496.01	485.16	487.79	390.60	362.35	5.01%	-0.54%	24.21%	7.80%	33.89%
VT	85.44	97.10	91.75	91.59	69.62	64.49	12.71%	0.17%	31.79%	7.95%	42.27%
WA	262.23	280.45	271.84	272.96	215.36	203.87	6.70%	-0.41%	26.23%	5.64%	33.34%
WI	266.55	288.45	278.47	272.60	224.59	216.93	7.86%	2.15%	23.99%	3.53%	28.37%
WV	326.00	349.99	337.34	336.37	268.88	261.01	7.11%	0.29%	25.46%	3.02%	29.24%
WY	68.53	80.04	75.63	68.31	56.43	54.59	15.22%	10.72%	34.02%	3.37%	38.54%
Avg.	299.94	319.18	309.92	309.48	245.46	233.22	8.89%	0.55%	26.52%	6.09%	34.20%

Table A-10. Results of AT_{Data}^{Sep} Under Query Window (0.5*0.5) for Zip Code Data Sets

	Rand-Min -1000	Rand-Max -1000	Rand-Avg -1000	Hilbert	R-Tree	Opt	Rand-Improv.	Hilbert-Improv.	R-tree-Improv.	Opt-Improv.	R-Tree+Opt--Improv.
AK	93.37	104.42	99.73	100.08	74.49	71.02	11.08%	-0.35%	33.88%	4.89%	40.43%
AL	517.05	553.65	536.38	535.54	418.52	389.47	6.82%	0.16%	28.16%	7.46%	37.72%
AR	400.86	438.38	418.02	422.30	327.60	310.42	8.98%	-1.01%	27.60%	5.53%	34.66%
AZ	326.74	344.84	336.37	334.62	248.40	239.06	5.38%	0.52%	35.41%	3.91%	40.71%
CA	1606.49	1653.97	1634.25	1638.71	1300.69	1242.15	2.91%	-0.27%	25.64%	4.71%	31.57%
CO	348.04	371.20	360.76	363.36	285.04	269.93	6.42%	-0.72%	26.56%	5.60%	33.65%
CT	355.88	370.30	362.78	360.97	295.50	284.46	3.97%	0.50%	22.77%	3.88%	27.53%
DC											
DE	74.75	88.25	83.23	69.26	64.74	60.31	16.22%	20.17%	28.56%	7.35%	38.00%
FL	1066.42	1105.90	1089.09	1099.03	849.83	799.15	3.63%	-0.90%	28.15%	6.34%	36.28%
GA	558.96	589.92	575.00	565.42	450.85	427.60	5.38%	1.69%	27.54%	5.44%	34.47%
HI	67.19	77.85	73.08	74.71	54.05	50.23	14.59%	-2.18%	35.21%	7.61%	45.49%
IA	616.03	657.18	636.51	622.39	535.86	503.48	6.46%	2.27%	18.78%	6.43%	26.42%
ID	149.83	164.22	157.57	153.26	126.80	114.63	9.13%	2.81%	24.27%	10.62%	37.46%
IL	1128.41	1177.86	1149.44	1152.30	869.44	841.59	4.30%	-0.25%	32.20%	3.31%	36.58%
IN	737.86	772.31	754.51	754.53	639.99	608.01	4.57%	0.00%	17.89%	5.26%	24.09%
KS	321.47	342.50	332.00	328.98	293.58	263.14	6.33%	0.92%	13.09%	11.57%	26.17%
KY	887.41	924.99	907.41	907.07	718.07	665.80	4.14%	0.04%	26.37%	7.85%	36.29%
LA	481.96	513.33	500.01	499.13	397.09	384.48	6.27%	0.18%	25.92%	3.28%	30.05%
MA	548.54	568.58	558.80	550.14	482.26	470.78	3.59%	1.57%	15.87%	2.44%	18.70%
MD	485.99	508.69	498.27	497.17	406.52	385.15	4.56%	0.22%	22.57%	5.55%	29.37%
ME	360.22	382.29	372.33	374.45	300.65	284.07	5.93%	-0.57%	23.84%	5.84%	31.07%
MI	778.56	813.47	795.65	796.52	660.74	634.65	4.39%	-0.11%	20.42%	4.11%	25.37%
MN	559.67	592.90	576.64	560.31	444.38	410.25	5.76%	2.91%	29.76%	8.32%	40.56%
MO	710.48	747.65	729.79	728.03	559.58	519.05	5.09%	0.24%	30.42%	7.81%	40.60%
MS	310.09	331.54	321.15	320.79	259.60	251.37	6.68%	0.11%	23.71%	3.27%	27.76%
MT	126.25	138.56	132.23	130.58	100.06	94.87	9.31%	1.26%	32.15%	5.47%	39.38%

NC	731.23	765.21	748.30	746.89	641.54	590.09	4.54%	0.19%	16.64%	8.72%	26.81%
ND	99.53	115.04	107.88	106.88	87.37	77.84	14.38%	0.94%	23.47%	12.24%	38.59%
NE	270.91	295.43	283.94	275.63	238.67	216.76	8.64%	3.01%	18.97%	10.11%	30.99%
NH	217.09	234.65	225.61	225.85	189.20	179.88	7.78%	-0.11%	19.24%	5.18%	25.42%
NJ	573.58	594.06	584.96	582.46	458.40	432.29	3.50%	0.43%	27.61%	6.04%	35.32%
NM	247.13	266.43	256.54	256.79	190.33	184.95	7.52%	-0.10%	34.79%	2.91%	38.71%
NV	137.39	152.61	145.61	148.67	115.78	107.63	10.45%	-2.06%	25.76%	7.57%	35.29%
NY	1520.35	1567.40	1544.88	1532.60	1194.12	1087.50	3.05%	0.80%	29.37%	9.80%	42.06%
OH	1136.94	1175.87	1156.24	1153.24	943.18	906.49	3.37%	0.26%	22.59%	4.05%	27.55%
OK	387.46	420.94	408.49	409.76	315.93	305.12	8.20%	-0.31%	29.30%	3.54%	33.88%
OR	221.74	247.72	235.16	232.80	185.88	174.40	11.05%	1.01%	26.51%	6.58%	34.84%
PA	1776.30	1820.37	1797.00	1790.46	1478.43	1443.91	2.45%	0.37%	21.55%	2.39%	24.45%
RI	76.09	82.57	79.56	78.44	67.72	62.52	8.14%	1.43%	17.48%	8.32%	27.26%
SC	354.79	385.11	368.23	368.63	287.40	279.07	8.23%	-0.11%	28.12%	2.98%	31.95%
SD	113.07	128.40	120.74	122.44	100.57	94.69	12.70%	-1.39%	20.06%	6.21%	27.51%
TN	527.99	565.44	547.18	539.50	411.96	385.21	6.84%	1.42%	32.82%	6.94%	42.05%
TX	1540.12	1587.13	1560.94	1568.73	1240.95	1184.95	3.01%	-0.50%	25.79%	4.73%	31.73%
UT	219.95	238.70	230.06	228.56	184.19	168.86	8.15%	0.66%	24.90%	9.08%	36.24%
VA	898.02	932.09	916.16	919.71	732.85	694.35	3.72%	-0.39%	25.01%	5.54%	31.94%
VT	240.67	259.17	250.33	257.30	194.10	185.86	7.39%	-2.71%	28.97%	4.43%	34.69%
WA	414.33	438.79	426.83	421.55	353.06	324.69	5.73%	1.25%	20.89%	8.74%	31.46%
WI	542.75	579.18	562.36	553.31	438.17	414.88	6.48%	1.64%	28.34%	5.61%	35.55%
WV	747.62	780.15	765.49	762.36	645.45	605.85	4.25%	0.41%	18.60%	6.54%	26.35%
WY	81.25	92.13	87.18	81.12	66.28	60.34	12.48%	7.47%	31.53%	9.84%	44.48%
Avg.	533.90	561.19	548.01	546.07	438.52	414.86	6.88%	0.86%	25.50%	6.24%	33.31%

Table A-11. Results of AT_{Data}^{Sep} Under Query Window (1.0*1.0) for Zip Code Data Sets

	Rand-Min -1000	Rand-Max -1000	Rand-Avg -1000	Hilbert	R-Tree	Opt	Rand-Improv.	Hilbert-Improv.	R-tree-Improv.	Opt-Improv.	R-Tree+Opt--Improv.
AK	108.15	118.23	113.69	114.80	95.53	87.11	8.87%	-0.97%	19.01%	9.67%	30.51%
AL	656.86	680.74	669.97	670.26	576.85	559.56	3.56%	-0.04%	16.14%	3.09%	19.73%
AR	573.21	597.76	586.20	588.43	473.07	442.65	4.19%	-0.38%	23.91%	6.87%	32.43%
AZ	348.13	372.47	362.47	361.48	310.50	286.85	6.72%	0.27%	16.74%	8.24%	26.36%
CA	1561.55	1600.59	1580.70	1585.74	1207.31	1176.20	2.47%	-0.32%	30.93%	2.64%	34.39%
CO	438.19	463.92	452.86	460.87	368.69	356.96	5.68%	-1.74%	22.83%	3.29%	26.87%
CT	354.90	367.45	361.59	355.97	333.10	312.21	3.47%	1.58%	8.55%	6.69%	15.82%
DC											
DE	84.73	91.41	88.33	87.67	76.17	72.89	7.56%	0.75%	15.96%	4.50%	21.18%
FL	1012.31	1037.54	1025.01	1023.29	870.92	832.74	2.46%	0.17%	17.69%	4.58%	23.09%
GA	705.72	728.92	718.72	719.25	554.19	529.72	3.23%	-0.07%	29.69%	4.62%	35.68%
HI	77.95	86.83	83.12	83.22	63.79	59.70	10.68%	-0.12%	30.30%	6.85%	39.23%
IA	881.04	901.33	892.00	891.83	725.17	702.87	2.27%	0.02%	23.01%	3.17%	26.91%
ID	229.51	246.59	238.36	239.24	203.83	192.07	7.17%	-0.37%	16.94%	6.12%	24.10%
IL	1232.35	1256.27	1245.85	1247.45	1000.16	966.20	1.92%	-0.13%	24.57%	3.51%	28.94%
IN	869.48	891.16	879.04	880.85	692.86	666.88	2.47%	-0.21%	26.87%	3.90%	31.81%
KS	607.35	633.77	620.86	632.49	491.31	479.26	4.26%	-1.84%	26.37%	2.51%	29.55%
KY	992.71	1014.65	1003.95	1004.81	807.39	776.76	2.19%	-0.09%	24.35%	3.94%	29.25%
LA	594.46	617.65	607.22	610.42	521.20	495.93	3.82%	-0.52%	16.50%	5.10%	22.44%
MA	531.17	543.81	537.44	535.55	389.34	376.12	2.35%	0.35%	38.04%	3.51%	42.89%
MD	461.21	477.39	469.71	470.77	385.70	356.48	3.44%	-0.23%	21.78%	8.20%	31.76%
ME	456.31	470.75	463.84	466.68	384.49	362.70	3.11%	-0.61%	20.64%	6.01%	27.89%
MI	926.17	950.68	939.29	942.02	751.53	720.37	2.61%	-0.29%	24.98%	4.33%	30.39%
MN	821.65	851.34	836.46	833.58	711.78	680.25	3.55%	0.35%	17.52%	4.64%	22.96%
MO	942.03	969.30	954.68	951.39	815.71	774.97	2.86%	0.35%	17.04%	5.26%	23.19%
MS	438.99	459.77	450.53	455.48	374.31	358.93	4.61%	-1.09%	20.36%	4.28%	25.52%

MT	206.26	224.40	216.28	219.33	181.77	168.18	8.39%	-1.39%	18.99%	8.08%	28.60%
NC	877.90	899.02	889.03	885.73	734.33	683.07	2.38%	0.37%	21.07%	7.50%	30.15%
ND	302.70	326.78	313.88	313.24	276.17	263.35	7.67%	0.20%	13.65%	4.87%	19.19%
NE	456.31	474.85	466.31	454.62	390.28	368.18	3.98%	2.57%	19.48%	6.00%	26.65%
NH	261.58	270.51	266.31	263.63	242.05	224.59	3.35%	1.02%	10.02%	7.77%	18.58%
NJ	402.66	416.55	410.57	407.75	356.71	319.40	3.38%	0.69%	15.10%	11.68%	28.54%
NM	308.16	332.51	321.84	321.66	263.23	238.50	7.57%	0.06%	22.27%	10.37%	34.94%
NV	145.49	161.96	153.74	154.53	122.53	116.82	10.71%	-0.51%	25.47%	4.89%	31.60%
NY	1550.34	1579.69	1564.49	1562.66	1327.22	1245.28	1.88%	0.12%	17.88%	6.58%	25.63%
OH	1264.08	1290.45	1278.52	1272.91	1104.94	1044.99	2.06%	0.44%	15.71%	5.74%	22.35%
OK	607.62	634.44	621.73	623.41	543.21	513.27	4.31%	-0.27%	14.45%	5.83%	21.13%
OR	319.10	341.52	330.71	332.01	275.56	259.89	6.78%	-0.39%	20.01%	6.03%	27.25%
PA	1701.69	1728.22	1715.23	1716.47	1426.09	1324.75	1.55%	-0.07%	20.28%	7.65%	29.48%
RI	81.80	87.24	84.96	84.64	70.97	64.54	6.40%	0.38%	19.71%	9.96%	31.64%
SC	462.90	479.84	472.06	468.29	392.54	376.90	3.59%	0.81%	20.26%	4.15%	25.25%
SD	284.41	307.51	298.00	301.41	263.95	245.62	7.75%	-1.13%	12.90%	7.46%	21.33%
TN	631.94	651.47	640.84	642.29	519.58	493.29	3.05%	-0.23%	23.34%	5.33%	29.91%
TX	1830.10	1881.82	1858.34	1857.77	1485.57	1419.04	2.78%	0.03%	25.09%	4.69%	30.96%
UT	234.04	250.93	243.91	245.86	204.66	188.17	6.92%	-0.79%	19.18%	8.76%	29.62%
VA	973.36	996.81	985.96	980.07	832.48	792.34	2.38%	0.60%	18.44%	5.07%	24.44%
VT	280.26	289.77	285.04	283.89	244.11	232.74	3.34%	0.41%	16.77%	4.89%	22.47%
WA	531.82	556.35	544.46	537.04	434.92	416.15	4.51%	1.38%	25.19%	4.51%	30.83%
WI	744.96	771.76	757.59	760.96	655.88	615.96	3.54%	-0.44%	15.51%	6.48%	22.99%
WV	836.49	858.39	849.36	852.37	692.11	665.01	2.58%	-0.35%	22.72%	4.08%	27.72%
WY	107.35	121.67	114.33	110.10	93.77	87.72	12.53%	3.84%	21.93%	6.90%	30.34%
Avg.	626.19	647.30	637.31	637.32	526.39	499.88	4.58%	0.04%	20.52%	5.82%	27.49%

Table A-12. Results of AT_{Data}^{Sep} Under Query Window (5*5) for Zip Code Data Sets

	Rand-Min -1000	Rand-Max -1000	Rand-Avg -1000	Hilbert	R-Tree	Opt	Rand-Improv.	Hilbert-Improv.	R-tree-Improv.	Opt-Improv.	R-Tree+Opt--Improv.
AK	213.47	227.24	220.47	222.41	178.16	159.53	6.25%	-0.87%	23.75%	11.68%	38.20%
AL											
AR	42.19	45.85	44.62	43.58	39.92	39.01	8.20%	2.39%	11.77%	2.33%	14.38%
AZ	285.20	295.22	291.09	290.52	259.27	218.83	3.44%	0.20%	12.27%	18.48%	33.02%
CA	357.91	380.22	370.82	375.36	315.85	278.91	6.02%	-1.21%	17.40%	13.24%	32.95%
CO	330.15	340.16	336.09	335.59	307.12	283.40	2.98%	0.15%	9.43%	8.37%	18.59%
CT											
DC											
DE	47.83	49.89	49.24	48.52	47.08	44.49	4.18%	1.48%	4.59%	5.82%	10.68%
FL	37.72	45.92	43.98	44.89	38.71	32.90	18.64%	-2.03%	13.61%	17.66%	33.68%
GA	40.67	44.86	43.53	44.11	40.45	38.25	9.63%	-1.31%	7.61%	5.75%	13.80%
HI	64.93	67.15	66.58	66.43	60.90	59.46	3.33%	0.23%	9.33%	2.42%	11.97%
IA											
ID	276.05	284.73	281.40	282.81	241.41	218.34	3.08%	-0.50%	16.57%	10.57%	28.88%
IL											
IN											
KS	173.31	181.81	178.32	176.23	161.70	140.39	4.77%	1.19%	10.28%	15.18%	27.02%
KY	41.38	45.90	44.09	44.90	42.21	40.81	10.25%	-1.80%	4.45%	3.43%	8.04%
LA											
MA											
MD	34.91	35.88	35.64	35.49	34.82	34.41	2.72%	0.42%	2.35%	1.19%	3.57%
ME	128.82	135.46	132.91	133.70	124.65	108.27	5.00%	-0.59%	6.63%	15.13%	22.76%
MI	179.90	187.39	184.56	186.10	163.17	156.95	4.06%	-0.83%	13.11%	3.96%	17.59%
MN	210.38	222.88	217.80	222.24	182.27	169.22	5.74%	-2.00%	19.49%	7.71%	28.71%
MO	42.33	49.98	46.71	46.19	43.54	38.69	16.38%	1.13%	7.28%	12.54%	20.73%
MS	86.66	94.84	92.58	93.77	88.61	77.08	8.84%	-1.27%	4.48%	14.96%	20.11%

MT	385.57	394.11	390.45	391.22	352.93	328.93	2.19%	-0.20%	10.63%	7.30%	18.70%
NC	81.93	89.57	87.12	87.78	81.39	69.84	8.77%	-0.75%	7.04%	16.54%	24.74%
ND	309.54	320.50	315.77	313.96	305.55	256.60	3.47%	0.58%	3.34%	19.08%	23.06%
NE	204.31	212.61	209.10	207.73	175.94	165.99	3.97%	0.66%	18.85%	5.99%	25.97%
NH	38.53	44.00	42.74	42.13	41.98	40.97	12.80%	1.45%	1.81%	2.47%	4.32%
NJ											
NM	381.44	392.75	387.91	385.54	352.92	334.47	2.92%	0.61%	9.91%	5.52%	15.98%
NV	101.51	115.22	109.56	107.98	89.87	86.39	12.51%	1.46%	21.91%	4.03%	26.82%
NY											
OH											
OK	151.04	165.69	160.48	155.16	122.88	113.58	9.13%	3.43%	30.60%	8.19%	41.29%
OR	284.36	294.72	290.18	287.18	250.05	234.68	3.57%	1.04%	16.05%	6.55%	23.65%
PA											
RI											
SC	41.76	47.96	46.24	45.37	44.71	38.52	13.41%	1.92%	3.42%	16.07%	20.04%
SD	321.27	330.00	326.19	326.86	312.90	285.26	2.68%	-0.20%	4.25%	9.69%	14.35%
TN	42.57	47.87	46.22	46.10	42.19	39.99	11.47%	0.26%	9.55%	5.50%	15.58%
TX	459.03	479.70	470.47	466.34	380.10	360.12	4.39%	0.89%	23.78%	5.55%	30.64%
UT	180.55	189.94	185.85	184.96	163.18	153.35	5.05%	0.48%	13.89%	6.41%	21.19%
VA	43.23	49.89	47.72	46.77	42.92	38.68	13.96%	2.03%	11.18%	10.96%	23.37%
VT											
WA	349.11	363.23	357.21	355.50	303.38	280.68	3.95%	0.48%	17.74%	8.09%	27.27%
WI	102.04	109.84	106.74	107.54	100.58	86.45	7.31%	-0.74%	6.12%	16.34%	23.47%
WV	48.04	49.90	49.42	49.79	49.77	45.60	3.76%	-0.74%	-0.70%	9.14%	8.38%
WY	184.06	189.38	186.90	187.99	155.22	150.32	2.85%	-0.58%	20.41%	3.26%	24.33%
Avg.	170.37	178.98	175.59	175.37	155.09	141.87	6.80%	0.18%	11.47%	9.11%	21.56%

Table A-13. Results of AT_{Data}^{Mul} Under Query Window (0.05*0.05) for Zip Code Data Sets

	Rand-Min -1000	Rand-Max -1000	Rand-Avg -1000	Hilbert	R-Tree	Opt	Rand-Improv.	Hilbert-Improv.	R-tree-Improv.	Opt-Improv.	R-Tree+Opt--Improv.
AK	75.96	89.8	82.81	84.37	66.65	58.57	16.71%	-1.85%	24.25%	13.80%	41.39%
AL	258.17	281.16	268.26	271.49	224.15	208.28	8.57%	-1.19%	19.68%	7.62%	28.80%
AR	133.19	149.4	142.45	145.01	119.06	111.42	11.38%	-1.77%	19.65%	6.86%	27.85%
AZ	220.93	241.01	232.01	231.5	188.45	179.93	8.65%	0.22%	23.11%	4.74%	28.94%
CA	1106.34	1148.33	1126.91	1124.82	873.89	856.38	3.73%	0.19%	28.95%	2.04%	31.59%
CO	224.81	243.34	233.8	234.58	188.81	181.38	7.93%	-0.33%	23.83%	4.10%	28.90%
CT	151.18	165.74	157.87	160.6	128.45	118.12	9.22%	-1.70%	22.90%	8.75%	33.65%
DC	37.42	44.91	41.39	40.21	33.72	29.91	18.10%	2.93%	22.75%	12.74%	38.38%
DE	28.7	35.84	32.55	32.7	25.16	23.77	21.94%	-0.46%	29.37%	5.85%	36.94%
FL	580.92	611.68	595.65	591.99	442.61	433.68	5.16%	0.62%	34.58%	2.06%	37.35%
GA	291.63	313.74	303.06	304.89	239.27	229.85	7.30%	-0.60%	26.66%	4.10%	31.85%
HI	39.42	47.48	44	42.77	35.55	32.93	18.32%	2.88%	23.77%	7.96%	33.62%
IA	160.72	177.97	169.55	170.85	138.58	131.12	10.17%	-0.76%	22.35%	5.69%	29.31%
ID	75.84	87.61	82.23	80.67	72.91	65.85	14.31%	1.93%	12.78%	10.72%	24.87%
IL	369.17	399.15	385.35	377.48	321.97	308.89	7.78%	2.08%	19.69%	4.23%	24.75%
IN	304.63	327.35	316.09	317.94	256.09	239.9	7.19%	-0.58%	23.43%	6.75%	31.76%
KS	90.8	102.73	96.73	94.53	78.09	72.18	12.33%	2.33%	23.87%	8.19%	34.01%
KY	353.12	379.82	367.76	357.1	312.73	299.85	7.26%	2.99%	17.60%	4.30%	22.65%
LA	291.33	314.11	302.74	299.49	237.57	227.42	7.52%	1.09%	27.43%	4.46%	33.12%
MA	247.93	269.99	259.96	260.87	226.73	200.74	8.49%	-0.35%	14.66%	12.95%	29.50%
MD	201.42	222.56	211.11	214.25	169.7	161.03	10.01%	-1.47%	24.40%	5.38%	31.10%
ME	123.49	140.09	131.98	127.34	110.34	104.91	12.58%	3.64%	19.61%	5.18%	25.80%
MI	294.65	315.97	304.64	304.27	246.06	235.37	7.00%	0.12%	23.81%	4.54%	29.43%
MN	125.69	141.53	133.68	133.35	102.32	100.53	11.85%	0.25%	30.65%	1.78%	32.98%
MO	222.69	244.01	233.04	234.25	178.91	173.46	9.15%	-0.52%	30.26%	3.14%	34.35%
MS	192.84	209.64	200.76	199.34	164.47	153.94	8.37%	0.71%	22.06%	6.84%	30.41%
MT	100.47	114.24	107.76	106.43	82.81	79.26	12.78%	1.25%	30.13%	4.48%	35.96%

NC	365.36	390.14	378.09	380.94	301.29	295.39	6.55%	-0.75%	25.49%	2.00%	28.00%
ND	40.28	48.97	45.05	43.56	36.38	33.85	19.29%	3.42%	23.83%	7.47%	33.09%
NE	67.41	78.86	73.87	73.08	59.1	57.06	15.50%	1.08%	24.99%	3.58%	29.46%
NH	64.09	73.36	68.88	68.9	55.78	53.31	13.46%	-0.03%	23.49%	4.63%	29.21%
NJ	207.75	228.19	218.36	214.02	194.56	174.42	9.36%	2.03%	12.23%	11.55%	25.19%
NM	160.5	177.95	170.07	168.27	142.07	137.68	10.26%	1.07%	19.71%	3.19%	23.53%
NV	103.27	117.26	110.08	110.18	81.12	79.62	12.71%	-0.09%	35.70%	1.88%	38.26%
NY	637.21	672.7	655.01	666.3	527.77	506.74	5.42%	-1.69%	24.11%	4.15%	29.26%
OH	519.44	550.5	537.02	530.78	449.84	423.42	5.78%	1.18%	19.38%	6.24%	26.83%
OK	230.68	248.67	239.81	240.8	194.24	184.99	7.50%	-0.41%	23.46%	5.00%	29.63%
OR	104.17	118.59	110.94	112.54	88.5	85.01	13.00%	-1.42%	25.36%	4.11%	30.50%
PA	800.92	842.42	821.33	817.88	657.64	637.76	5.05%	0.42%	24.89%	3.12%	28.78%
RI	23.34	28.74	26.02	25.24	23.11	21.18	20.75%	3.09%	12.59%	9.11%	22.85%
SC	187.99	207.31	198.43	197.7	158	151.14	9.74%	0.37%	25.59%	4.54%	31.29%
SD	43.24	55	49.73	48.64	39.33	36.86	23.65%	2.24%	26.44%	6.70%	34.92%
TN	220.22	241.05	231.95	234.09	190.46	177.03	8.98%	-0.91%	21.78%	7.59%	31.02%
TX	866.94	900.25	884.22	882.45	690.64	658.89	3.77%	0.20%	28.03%	4.82%	34.20%
UT	142.43	160.11	152.19	153.73	120.38	111.1	11.62%	-1.00%	26.42%	8.35%	36.98%
VA	397.32	425.37	411.78	411.49	306.02	291.12	6.81%	0.07%	34.56%	5.12%	41.45%
VT	72.69	83.08	78.1	78.74	70.09	62.45	13.30%	-0.81%	11.43%	12.23%	25.06%
WA	211.14	230.08	221.2	221.54	188.47	176.9	8.56%	-0.15%	17.37%	6.54%	25.04%
WI	216.82	237.14	225.73	231.82	189.29	180.03	9.00%	-2.63%	19.25%	5.14%	25.38%
WV	234.75	254.28	244.38	249.3	204.6	186.47	7.99%	-1.97%	19.44%	9.72%	31.06%
WY	60.62	71.45	66.67	60.48	51.02	49.33	16.24%	10.23%	30.67%	3.43%	35.15%
Avg.	240.82	260.01	250.65	250.31	201.66	191.97	10.75%	0.49%	23.50%	6.07%	30.89%

Table A-14. Results of AT_{Data}^{Mul} Under Query Window (0.1*0.1) for Zip Code Data Sets

	Rand-Min -1000	Rand-Max -1000	Rand-Avg -1000	Hilbert	R-Tree	Opt	Rand-Improv.	Hilbert-Improv.	R-tree-Improv.	Opt-Improv.	R-Tree+Opt--Improv.
AK	75.96	89.8	82.81	84.38	66.65	58.57	16.71%	-1.86%	24.25%	13.80%	41.39%
AL	282.38	303.98	293.68	296.77	227.12	217.41	7.35%	-1.04%	29.31%	4.47%	35.08%
AR	133.19	149.4	142.45	145.01	119.06	111.42	11.38%	-1.77%	19.65%	6.86%	27.85%
AZ	224.23	243.69	235.26	234.98	190.09	181.59	8.27%	0.12%	23.76%	4.68%	29.56%
CA	1234.6	1285.02	1259.15	1253.78	986.41	963.21	4.00%	0.43%	27.65%	2.41%	30.72%
CO	235.03	255.61	246.58	244.55	207.86	198.62	8.35%	0.83%	18.63%	4.65%	24.15%
CT	180.23	199.38	190.1	189.79	158.24	144.57	10.07%	0.16%	20.13%	9.46%	31.49%
DC	29.39	39.63	35.76	36.62	29.11	26.5	28.64%	-2.35%	22.84%	9.85%	34.94%
DE	41.5	50.45	46.67	38.6	37.41	34.9	19.18%	20.91%	24.75%	7.19%	33.72%
FL	590.32	624.16	610.07	611.81	490.1	465.51	5.55%	-0.28%	24.48%	5.28%	31.05%
GA	301.67	325.93	314.47	311.43	270.33	255.13	7.71%	0.98%	16.33%	5.96%	23.26%
HI	37.6	45.49	41.8	41.53	31.35	29.77	18.88%	0.65%	33.33%	5.31%	40.41%
IA	157.85	173.3	165.92	168.22	135.64	129.59	9.31%	-1.37%	22.32%	4.67%	28.03%
ID	90.34	102.91	97.19	97.74	85.47	81.91	12.93%	-0.56%	13.71%	4.35%	18.65%
IL	416.63	446.73	433.47	430.29	349.25	336.89	6.94%	0.74%	24.11%	3.67%	28.67%
IN	323.94	347.69	336.3	330.34	261.65	248.02	7.06%	1.80%	28.53%	5.50%	35.59%
KS	92.81	105.02	99.04	98.66	75	73.43	12.33%	0.39%	32.05%	2.14%	34.88%
KY	394.37	419.53	407.58	404.82	333.58	319.12	6.17%	0.68%	22.18%	4.53%	27.72%
LA	292.56	318.22	306.67	309.66	242.48	234.26	8.37%	-0.97%	26.47%	3.51%	30.91%
MA	254.98	284.38	272.66	266.65	207.43	192.74	10.78%	2.25%	31.45%	7.62%	41.47%
MD	222.59	245.88	234.6	235.74	196.46	179.56	9.93%	-0.48%	19.41%	9.41%	30.65%
ME	126.96	141.56	133.95	130.43	108.7	102.1	10.90%	2.70%	23.23%	6.46%	31.19%
MI	329.15	357.13	341.62	337.48	268.33	257.89	8.19%	1.23%	27.31%	4.05%	32.47%
MN	181.99	202.03	192.57	193.56	166.66	154.34	10.41%	-0.51%	15.55%	7.98%	24.77%
MO	236.92	261.68	250.14	251.58	221.55	210.79	9.90%	-0.57%	12.90%	5.10%	18.67%
MS	193.37	212.06	202.48	200.85	169.51	157.87	9.23%	0.81%	19.45%	7.37%	28.26%
MT	101.6	115.62	108.45	107.66	83.79	79.53	12.93%	0.73%	29.43%	5.36%	36.36%

NC	389.03	415.87	401.07	401.33	335.53	320.58	6.69%	-0.06%	19.53%	4.66%	25.11%
ND	41.4	49.87	45.78	45.95	37.71	35.95	18.50%	-0.37%	21.40%	4.90%	27.34%
NE	67.41	78.86	73.87	73.08	59.1	57.06	15.50%	1.08%	24.99%	3.58%	29.46%
NH	64.37	74.69	70.13	69.85	56.64	52.85	14.72%	0.40%	23.82%	7.17%	32.70%
NJ	298.33	335.73	318.69	316.31	267.21	254.52	11.74%	0.75%	19.27%	4.99%	25.21%
NM	173.39	189.47	181.63	178.83	150.37	141.06	8.85%	1.57%	20.79%	6.60%	28.76%
NV	103.27	117.26	110.08	110.18	81.12	79.62	12.71%	-0.09%	35.70%	1.88%	38.26%
NY	808.74	851.95	830.65	834.21	674.31	636.41	5.20%	-0.43%	23.19%	5.96%	30.52%
OH	557.45	594.1	575.94	567.88	475.08	455.35	6.36%	1.42%	21.23%	4.33%	26.48%
OK	232.16	251	240.41	243.25	197.56	189.18	7.84%	-1.17%	21.69%	4.43%	27.08%
OR	107.77	121.36	114.42	113.54	91.45	87.05	11.88%	0.78%	25.12%	5.05%	31.44%
PA	955.85	1002.05	976.58	986.82	782.69	762.35	4.73%	-1.04%	24.77%	2.67%	28.10%
RI	24.53	31.25	28.12	28.98	23.37	21.45	23.90%	-2.97%	20.33%	8.95%	31.10%
SC	204.73	222.76	214.2	214.33	170.62	163.48	8.42%	-0.06%	25.54%	4.37%	31.03%
SD	46.83	55.99	51.17	49.12	41.56	40.52	17.90%	4.17%	23.12%	2.57%	26.28%
TN	236.13	259.52	248.05	245.44	209.46	198.93	9.43%	1.06%	18.42%	5.29%	24.69%
TX	968.72	1009.81	990.16	997.1	858.55	812	4.15%	-0.70%	15.33%	5.73%	21.94%
UT	143.32	161.88	153.97	152.35	126.67	116.57	12.05%	1.06%	21.55%	8.66%	32.08%
VA	417.19	447.94	433.94	437.89	348.24	333.16	7.09%	-0.90%	24.61%	4.53%	30.25%
VT	73.47	86.72	81.01	82.34	62.65	59.27	16.36%	-1.62%	29.31%	5.70%	36.68%
WA	231.49	252.05	241.44	243.59	195.36	187.69	8.52%	-0.88%	23.59%	4.09%	28.64%
WI	237.55	257.73	248.03	243.56	202.77	196.34	8.14%	1.84%	22.32%	3.27%	26.33%
WV	289.47	314.8	303.14	298.39	242	232.54	8.36%	1.59%	25.26%	4.07%	30.36%
WY	60.62	71.45	66.67	60.48	51.02	49.33	16.24%	10.23%	30.67%	3.43%	35.15%
Avg.	265.01	286.28	276.09	275.45	224.71	214.28	10.92%	0.77%	23.43%	5.46%	30.14%

Table A-15. Results of AT_{Data}^{Mul} Under Query Window (0.5*0.5) for Zip Code Data Sets

	Rand-Min -1000	Rand-Max -1000	Rand-Avg -1000	Hilbert	R-Tree	Opt	Rand-Improv.	Hilbert-Improv.	R-tree-Improv.	Opt-Improv.	R-Tree+Opt--Improv.
AK	81.88	95.42	89.02	89.5	66.22	63.65	15.21%	-0.54%	34.43%	4.04%	39.86%
AL	452.81	500.73	480.08	477.13	404.2	366.2	9.98%	0.62%	18.77%	10.38%	31.10%
AR	350.18	390.82	371.74	368.35	311.29	291.71	10.93%	0.92%	19.42%	6.71%	27.43%
AZ	287.82	313.67	301.7	300.22	218.84	206.45	8.57%	0.49%	37.86%	6.00%	46.14%
CA	1423.01	1497.23	1465.45	1482.29	1160.3	1072.39	5.06%	-1.14%	26.30%	8.20%	36.65%
CO	305.75	334.52	321.86	322.23	251.57	235.19	8.94%	-0.11%	27.94%	6.96%	36.85%
CT	315.71	339.35	327.9	323.14	265.95	244.33	7.21%	1.47%	23.29%	8.85%	34.20%
DC											
DE	62.77	81.09	74.31	56.1	59.45	50.78	24.65%	32.46%	25.00%	17.07%	46.34%
FL	952.28	1002.85	978.57	1001.06	750.14	699.7	5.17%	-2.25%	30.45%	7.21%	39.86%
GA	485.08	536.4	512.59	504.39	434.58	412.66	10.01%	1.63%	17.95%	5.31%	24.22%
HI	56.83	70.3	64.13	66.91	54.06	49.19	21.00%	-4.15%	18.63%	9.90%	30.37%
IA	533.01	591.49	566.27	555.47	501.07	476.23	10.33%	1.94%	13.01%	5.22%	18.91%
ID	131.43	150.04	140.89	138.9	115.28	106.13	13.21%	1.43%	22.22%	8.62%	32.75%
IL	995.9	1060.41	1026.53	1027.91	816.42	790.91	6.28%	-0.13%	25.74%	3.23%	29.79%
IN	645.42	695.77	673.42	681.43	588.77	556.9	7.48%	-1.18%	14.38%	5.72%	20.92%
KS	277.05	315.47	298.3	292.33	264.66	252.65	12.88%	2.04%	12.71%	4.75%	18.07%
KY	783.29	832.49	808.65	800.74	655.25	623.95	6.08%	0.99%	23.41%	5.02%	29.60%
LA	427.22	466.56	447.26	455	365.91	356.51	8.80%	-1.70%	22.23%	2.64%	25.46%
MA	486.46	514.55	499.77	489.52	417.88	399.39	5.62%	2.09%	19.60%	4.63%	25.13%
MD	430.39	464.13	445.99	441.56	366.58	348.28	7.57%	1.00%	21.66%	5.25%	28.06%
ME	316.44	344.65	330.84	331.97	277.81	263.55	8.53%	-0.34%	19.09%	5.41%	25.53%
MI	690.58	732.73	712.11	702.75	605.4	575.02	5.92%	1.33%	17.63%	5.28%	23.84%
MN	492.8	547.31	520.03	503.07	391.95	355.11	10.48%	3.37%	32.68%	10.37%	46.44%
MO	621.55	675.07	650.23	651.65	529.21	457.64	8.23%	-0.22%	22.87%	15.64%	42.08%
MS	272.3	299.95	286.94	291.28	228.03	220.19	9.64%	-1.49%	25.83%	3.56%	30.31%
MT	110.09	125.29	117.46	117.51	93.39	86.93	12.94%	-0.04%	25.77%	7.43%	35.12%

NC	643.71	692.51	668.34	673.93	566	532.1	7.30%	-0.83%	18.08%	6.37%	25.60%
ND	84.96	104.95	95.74	96.11	83.05	71.16	20.88%	-0.38%	15.28%	16.71%	34.54%
NE	237.44	268.71	254.51	240.48	202.09	178.3	12.29%	5.83%	25.94%	13.34%	42.74%
NH	184.26	211.33	199.39	195.88	170.26	162.54	13.58%	1.79%	17.11%	4.75%	22.67%
NJ	510.03	540.04	526.29	526.26	455.73	423.29	5.70%	0.01%	15.48%	7.66%	24.33%
NM	218.06	239.32	228.81	225.09	168.98	158.38	9.29%	1.65%	35.41%	6.69%	44.47%
NV	120.27	140.01	130.52	133.54	110.09	101.72	15.12%	-2.26%	18.56%	8.23%	28.31%
NY	1343.2	1406.19	1375.85	1358.15	1091.47	992.74	4.58%	1.30%	26.05%	9.95%	38.59%
OH	1003.42	1061.78	1031.22	1032.22	838.64	807.27	5.66%	-0.10%	22.96%	3.89%	27.74%
OK	343.88	386.99	363.1	361.44	286.46	277	11.87%	0.46%	26.75%	3.42%	31.08%
OR	190.47	222.13	208.9	209.33	156.81	146.2	15.16%	-0.21%	33.22%	7.26%	42.89%
PA	1583.88	1639.66	1610.7	1611.58	1319.13	1288.67	3.46%	-0.05%	22.10%	2.36%	24.99%
RI	65.85	74.63	71.11	71.83	62.9	54.17	12.35%	-1.00%	13.05%	16.12%	31.27%
SC	311.04	346.17	328.38	326.91	260.39	252.01	10.70%	0.45%	26.11%	3.33%	30.30%
SD	96.23	118.13	108.14	111.16	96.78	90.53	20.25%	-2.72%	11.74%	6.90%	19.45%
TN	461.4	515.55	488.14	477.88	385.57	349.42	11.09%	2.15%	26.60%	10.35%	39.70%
TX	1366.03	1429.34	1398.61	1403.2	1077.36	1019.87	4.53%	-0.33%	29.82%	5.64%	37.14%
UT	190.37	220.09	204.76	207.44	174.93	164.65	14.51%	-1.29%	17.05%	6.24%	24.36%
VA	796.38	843.2	819.84	818.99	674.93	629.86	5.71%	0.10%	21.47%	7.16%	30.16%
VT	207.17	233.58	222.4	227.97	176.4	166.97	11.88%	-2.44%	26.08%	5.65%	33.20%
WA	360.71	397.26	380.22	376.63	324.61	290.48	9.61%	0.95%	17.13%	11.75%	30.89%
WI	476.02	522.65	499.75	492.13	418.3	384.43	9.33%	1.55%	19.47%	8.81%	30.00%
WV	666.12	704.31	686.4	681.21	580.91	534.77	5.56%	0.76%	18.16%	8.63%	28.35%
WY	70.3	82.78	76.84	72.9	61.53	57.62	16.24%	5.40%	24.88%	6.79%	33.36%
Avg.	470.39	507.59	489.8	488.09	398.75	373.92	10.35%	0.99%	22.55%	7.43%	31.62%

Table A-16. Results of AT_{Data}^{Mul} Under Query Window (1*1) for Zip Code Data Sets

	Rand-Min -1000	Rand-Max -1000	Rand-Avg -1000	Hilbert	R-Tree	Opt	Rand-Improv.	Hilbert-Improv.	R-tree-Improv.	Opt-Improv.	R-Tree+Opt--Improv.
AK	94.21	109.68	102.14	102.25	91.12	81.12	15.15%	-0.11%	12.09%	12.33%	25.91%
AL	582.05	616.23	600.48	601.78	543.68	523.05	5.69%	-0.22%	10.45%	3.94%	14.80%
AR	508.31	534.04	522.38	520.13	433.27	396.63	4.93%	0.43%	20.57%	9.24%	31.70%
AZ	309.14	341.7	326.63	322.29	264.93	254.69	9.97%	1.35%	23.29%	4.02%	28.25%
CA	1383.79	1453.19	1423.52	1431.52	1107.75	1080.58	4.88%	-0.56%	28.51%	2.51%	31.74%
CO	387.51	424.77	405.79	419.53	349.68	335.69	9.18%	-3.28%	16.05%	4.17%	20.88%
CT	313.46	345.71	331.41	321.66	315.83	273.45	9.73%	3.03%	4.93%	15.50%	21.20%
DC											
DE	72.84	85.75	79.67	82.19	71.53	69.16	16.20%	-3.07%	11.38%	3.43%	15.20%
FL	913.64	955.5	933.23	926.55	785.55	730.84	4.49%	0.72%	18.80%	7.49%	27.69%
GA	626.04	658.58	642.47	639.14	498.18	478.54	5.06%	0.52%	28.96%	4.10%	34.26%
HI	66.03	79.2	73.62	76.04	57.32	51.59	17.89%	-3.18%	28.44%	11.11%	42.70%
IA	781.21	805.89	793.98	793.68	661.81	639.27	3.11%	0.04%	19.97%	3.53%	24.20%
ID	200.52	225.1	214.15	214.79	176.84	168.42	11.48%	-0.30%	21.10%	5.00%	27.15%
IL	1093.03	1134.04	1115.26	1121.8	883.99	842.27	3.68%	-0.58%	26.16%	4.95%	32.41%
IN	776.95	806.95	791.74	794.52	614.79	596.86	3.79%	-0.35%	28.78%	3.00%	32.65%
KS	532.61	569.78	552.45	562.33	438.33	422.33	6.73%	-1.76%	26.04%	3.79%	30.81%
KY	886.37	923.88	906.02	905.2	688.61	659.4	4.14%	0.09%	31.57%	4.43%	37.40%
LA	525.34	560.05	544.73	550.69	480.03	449.99	6.37%	-1.08%	13.48%	6.68%	21.05%
MA	471.82	508.82	492.5	493.92	314.03	297.03	7.51%	-0.29%	56.83%	5.72%	65.81%
MD	410.68	439.33	426.03	429.4	318.43	290.56	6.72%	-0.78%	33.79%	9.59%	46.62%
ME	406.61	427.84	417.45	420.6	338.1	318.67	5.09%	-0.75%	23.47%	6.10%	31.00%
MI	822.35	860.3	843.6	839.93	656.19	633.89	4.50%	0.44%	28.56%	3.52%	33.08%
MN	725.07	769.26	749.63	741.44	668.99	588.39	5.89%	1.10%	12.05%	13.70%	27.40%
MO	833.02	866.92	851.05	847.95	738.53	702.64	3.98%	0.37%	15.24%	5.11%	21.12%
MS	388.23	415.32	402.52	408.71	331.75	308.21	6.73%	-1.51%	21.33%	7.64%	30.60%
MT	178.97	206.32	192.53	191.41	162.75	152.03	14.21%	0.59%	18.30%	7.05%	26.64%

NC	778.19	810.65	796.36	788.23	650.67	602.68	4.08%	1.03%	22.39%	7.96%	32.14%
ND	263.1	294.54	279.38	272.82	247.02	223.43	11.25%	2.40%	13.10%	10.56%	25.04%
NE	400.9	427.61	415.29	411.2	376.82	360.68	6.43%	0.99%	10.21%	4.47%	15.14%
NH	230.67	245.91	239.46	238.19	216.05	192.62	6.36%	0.53%	10.84%	12.16%	24.32%
NJ	356.88	390.06	374.95	374.11	326.34	263.93	8.85%	0.22%	14.90%	23.65%	42.06%
NM	271.09	301.06	287.06	286.33	240.71	221.64	10.44%	0.25%	19.26%	8.60%	29.52%
NV	124.66	150.25	138.49	139.86	109.88	89.47	18.48%	-0.98%	26.04%	22.81%	54.79%
NY	1385.39	1436.02	1413.95	1408.46	1197.06	1090.12	3.58%	0.39%	18.12%	9.81%	29.71%
OH	1125.92	1172.72	1150.28	1139.85	1010.84	914.22	4.07%	0.92%	13.79%	10.57%	25.82%
OK	531.4	569.78	552.52	542.6	496.99	473.75	6.95%	1.83%	11.17%	4.91%	16.63%
OR	280.37	308.41	294.03	290.04	252.37	234.93	9.54%	1.38%	16.51%	7.42%	25.16%
PA	1533.13	1586.56	1561.46	1561.23	1281.2	1189.17	3.42%	0.01%	21.87%	7.74%	31.31%
RI	68.76	82.31	77.16	75.54	72.08	66.46	17.56%	2.14%	7.05%	8.46%	16.10%
SC	411.32	435.99	425.17	423.04	344.26	331.45	5.80%	0.50%	23.50%	3.86%	28.28%
SD	253.01	280.02	266.6	268.31	247.03	233.23	10.13%	-0.64%	7.92%	5.92%	14.31%
TN	558.71	587.25	573.56	568.92	458.22	442.95	4.98%	0.82%	25.17%	3.45%	29.49%
TX	1618.99	1707.75	1666.87	1674.69	1292.3	1267.05	5.32%	-0.47%	28.98%	1.99%	31.56%
UT	207.37	228.97	218.4	222.68	178.67	171.63	9.89%	-1.92%	22.24%	4.10%	27.25%
VA	862.3	908.54	887.29	875.62	786.43	738.67	5.21%	1.33%	12.83%	6.47%	20.12%
VT	249.36	261.6	256.04	256.43	218.36	205.96	4.78%	-0.15%	17.26%	6.02%	24.32%
WA	467.26	504.75	487.81	481.15	370.28	348.87	7.69%	1.38%	31.74%	6.14%	39.83%
WI	655.61	692.94	675.88	678.82	602.6	554.75	5.52%	-0.43%	12.16%	8.63%	21.84%
WV	744.14	786.35	768.28	766.53	622.45	593.87	5.49%	0.23%	23.43%	4.81%	29.37%
WY	91.08	108.81	101.17	97.82	87.95	80.45	17.52%	3.42%	15.03%	9.32%	25.76%
Avg.	555.19	588.06	572.85	572.04	473.57	444.75	7.81%	0.12%	20.11%	7.35%	28.84%

Table A-17. Results of AT_{Data}^{Mul} Under Query Window (5*5) for Zip Code Data Sets

	Rand-Min -1000	Rand-Max -1000	Rand-Avg -1000	Hilbert	R-Tree	Opt	Rand-Improv.	Hilbert-Improv.	R-tree-Improv.	Opt-Improv.	R-Tree+Opt--Improv.
AK	189.58	208.84	199.73	203.53	156.9	145.61	9.64%	-1.87%	27.30%	7.75%	37.17%
AL											
AR	34.02	44.86	41.38	41.9	40.03	39.53	26.20%	-1.24%	3.37%	1.26%	4.68%
AZ	256.1	278.54	268.83	267.56	217.37	184.6	8.35%	0.47%	23.67%	17.75%	45.63%
CA	312.43	362.14	340.32	349.45	267.19	222.19	14.61%	-2.61%	27.37%	20.25%	53.17%
CO	298.62	322.79	312.12	309.79	283.38	270.35	7.74%	0.75%	10.14%	4.82%	15.45%
CT											
DC											
DE	40.64	48.63	46.55	40.08	38.84	37.65	17.16%	16.14%	19.85%	3.16%	23.64%
FL	24.27	44.63	39.69	41.85	23.48	19.63	51.30%	-5.16%	69.04%	19.61%	102.19%
GA	31.78	44	40.39	40.83	33.41	27.6	30.26%	-1.08%	20.89%	21.05%	46.34%
HI	58.34	63.96	62.26	61.48	54.58	52.65	9.03%	1.27%	14.07%	3.67%	18.25%
IA											
ID	248.55	270.29	261.81	261.92	208.16	185.7	8.30%	-0.04%	25.77%	12.09%	40.99%
IL											
IN											
KS	148.06	173.66	165.11	161.86	137.81	118.34	15.50%	2.01%	19.81%	16.45%	39.52%
KY	33.65	45	40.89	43.48	41.85	39.73	27.76%	-5.96%	-2.29%	5.34%	2.92%
LA											
MA											
MD	31.58	34.61	33.82	33.81	34.06	32.84	8.96%	0.03%	-0.70%	3.71%	2.98%
ME	113.75	132.26	124.99	129.7	121.08	118.27	14.81%	-3.63%	3.23%	2.38%	5.68%
MI	155.94	180.4	172.34	176.41	152.3	136.09	14.19%	-2.31%	13.16%	11.91%	26.64%
MN	183.62	214.95	200.93	207.69	156.52	142.49	15.59%	-3.25%	28.37%	9.85%	41.01%
MO	30.52	49	43.08	49	38.47	37.12	42.90%	-12.08%	11.98%	3.64%	16.06%
MS	69.24	93.66	86.29	83.32	80.3	57.92	28.30%	3.56%	7.46%	38.64%	48.98%
MT	349.48	369.38	360.09	358.96	315.02	273.75	5.53%	0.31%	14.31%	15.08%	31.54%

NC	60.43	87.95	81.48	83.72	83.46	74.33	33.78%	-2.68%	-2.37%	12.28%	9.62%
ND	274.82	306.58	292.75	290.73	277.34	232.48	10.85%	0.69%	5.56%	19.30%	25.92%
NE	180.31	202.55	193.31	189.93	147.43	117.74	11.50%	1.78%	31.12%	25.22%	64.18%
NH	29.86	43	39.53	36.08	39.98	38.68	33.24%	9.56%	-1.13%	3.36%	2.20%
NJ											
NM	346.23	372.68	360.44	351.26	304.55	271.37	7.34%	2.61%	18.35%	12.23%	32.82%
NV	81.48	109.28	97.01	91.05	55.59	48.27	28.66%	6.55%	74.51%	15.16%	100.97%
NY											
OH											
OK	123.24	159.9	147.23	130.95	97.97	56.74	24.90%	12.43%	50.28%	72.66%	159.48%
OR	250.11	276.33	267.2	257.78	237.58	220.66	9.81%	3.65%	12.47%	7.67%	21.09%
PA											
RI											
SC	35.35	46.98	42.92	40.24	41.85	36.43	27.10%	6.66%	2.56%	14.88%	17.81%
SD	287.36	315.66	302.8	298.26	279.52	228.56	9.35%	1.52%	8.33%	22.30%	32.48%
TN	32.59	46.71	43.1	43.19	32.66	27.71	32.76%	-0.21%	31.97%	17.86%	55.54%
TX	402.1	453.51	433.23	416.2	316.02	305.14	11.87%	4.09%	37.09%	3.57%	41.98%
UT	160.94	180.85	172.12	172.56	125.85	121	11.57%	-0.25%	36.77%	4.01%	42.25%
VA	33.68	48.77	44.25	48.56	33.7	29.1	34.10%	-8.88%	31.31%	15.81%	52.06%
VT											
WA	307.11	348.22	332.99	330.9	247.23	239.56	12.35%	0.63%	34.69%	3.20%	39.00%
WI	86.94	107.98	99.26	103.55	81.81	63.08	21.20%	-4.14%	21.33%	29.69%	57.36%
WV	41.25	48.84	46.86	47.79	48.47	35.67	16.20%	-1.95%	-3.32%	35.88%	31.37%
WY	164.99	175.14	170.56	172.4	137.83	134.08	5.95%	-1.07%	23.75%	2.80%	27.21%
Avg.	148.89	170.61	162.37	161.29	134.85	119.53	18.88%	0.44%	20.27%	14.49%	38.28%

Table A-18. Computation Time for Optimizing DBW and AT_{Data}^{Mul} for Zip Code Data Sets

Data Set	0.05*0.05			0.1*0.1			0.5*0.5			1.0*1.0			5.0*5.0		
	Node	DBW	MUL	Node	DBW	MUL	Node	DBW	MUL	Node	DBW	MUL	Node	DBW	MUL
AK	109	0	0	109	0	0	115	0	0	130	0	0	250	1	0
AL	364	1	1	393	0	1	643	2	2	780	7	6	/	/	/
AR	200	0	0	200	0	0	517	1	1	694	5	3	46	0	0
AZ	310	0	0	314	0	0	387	1	0	414	4	2	305	4	2
CA	1543	7	8	1707	11	13	1905	28	22	1797	63	42	393	16	8
CO	316	0	1	331	1	0	422	1	0	527	3	2	348	14	7
CT	211	1	0	250	0	0	410	1	2	380	6	4			
DC	53	0	0	47	0	0									
DE	43	0	0	58	0	0	95	0	0	97	0	0	50	0	0
FL	821	1	2	840	2	2	1242	15	11	1125	39	25	46	0	0
GA	417	1	0	434	0	1	697	3	2	841	7	5	45	0	0
HI	60	0	0	57	0	0	89	0	0	96	0	0	68	1	0
IA	238	0	1	235	0	0	802	2	3	1052	12	9			
ID	113	0	0	129	0	0	187	0	0	281	0	0	292	26	13
IL	532	1	1	591	1	1	1378	12	10	1420	41	26			
IN	432	0	0	455	1	1	904	5	4	983	18	12			
KS	135	0	0	138	0	0	393	1	1	767	12	7	185	9	4
KY	504	1	1	550	1	1	1080	8	6	1114	54	30	46	0	0
LA	412	0	1	419	0	1	594	1	1	696	5	4			
MA	354	1	0	377	1	0	643	5	3	568	16	10			
MD	289	0	0	317	0	0	578	3	2	507	8	5	36	0	0
ME	185	0	0	188	0	1	452	1	1	519	6	4	137	3	1
MI	428	1	1	470	1	0	942	7	6	1075	23	15	190	7	4
MN	190	0	0	255	0	1	687	2	2	1007	13	9	229	14	7
MO	319	0	0	342	0	0	894	6	5	1132	22	14	50	0	0
MS	271	0	1	273	0	0	381	0	0	531	1	2	95	1	0
MT	145	0	0	146	0	0	158	0	0	259	1	0	409	24	12

NC	104	0	0	104	0	0	342	1	1	565	5	3	217	9	4
ND	521	1	1	546	0	1	901	4	3	1016	12	8	90	1	1
NE	61	0	0	62	0	0	129	0	0	390	1	1	328	21	11
NH	97	0	0	99	0	0	277	0	1	292	2	1	44	1	0
NJ	300	0	0	433	1	1	657	10	6	436	10	6			
NM	231	1	0	244	0	0	304	0	0	379	2	1	402	9	6
NV	145	0	0	145	0	0	169	0	0	178	0	0	127	1	0
NY	901	2	3	1140	4	5	1845	36	26	1712	115	68			
OH	748	2	2	797	2	2	1393	16	12	1418	57	34			
OK	336	0	0	336	0	0	502	1	1	757	6	4	170	5	3
OR	155	0	0	159	0	0	291	1	0	400	1	1	303	19	9
PA	1120	4	5	1312	5	5	2075	46	34	1847	126	76			
RI	38	0	0	41	0	0	90	0	0	90	1	0			
SC	273	1	1	290	0	1	437	1	1	537	3	3	48	0	0
SD	70	0	0	72	0	0	148	0	0	371	1	0	337	33	15
TN	317	0	0	340	0	0	662	2	2	744	7	4	48	1	0
TX	1227	4	4	1347	4	5	1837	34	26	2167	76	51	497	21	11
UT	197	0	0	199	0	0	269	0	1	279	1	1	196	2	1
VA	543	1	1	570	1	1	1082	8	6	1106	27	17	50	0	0
VT	109	0	0	113	0	0	307	0	0	314	2	1			
WA	304	0	1	327	1	0	503	2	2	633	6	4	370	28	13
WI	317	0	0	343	0	0	694	2	1	893	10	7	111	2	1
WV	331	1	0	396	0	1	900	4	4	928	27	17	50	1	0
WY	91	0	0	91	0	0	105	0	0	138	0	0	198	1	1