The Pennsylvania State University

The Graduate School

Department of Computer Science and Engineering

# ACCESSING SPATIAL INFORMATION IN RESOURCE-CONSTRAINED AND RESOURCE-RICH ENVIRONMENTS

A Thesis in

Computer Science and Engineering

by

Ning An



Submitted in Partial Fulfillment
of the Requirements
for the Degree of

Doctor of Philosophy

May 2002

We approve the thesis of Ning An.

Date of Signature

Anand Sivasubramaniam
Associate Professor of Computer Science and Engineering
Thesis Adviser
Chair of Committee

Donna Peuquet
Professor of Geography

C. Lee Giles
David Reese Professor
School of Information Sciences and Technology
Professor of Computer Science and Engineering

Vijaykrishnan Narayanan
Assistant Professor of Computer Science Engineering

Raj Acharya
Professor of Computer Science and Engineering
Head of the Department of Computer Science and Engineering

# Abstract

Spatial information has always been a need of the human society. Over the past two decades or so, Spatial Database Management Systems (SDBMS) have achieved a great deal in storing, processing and retrieving spatial information. The continuing advancement of technologies, however, persistently raises the bar for SDBMS to meet demands in various environments, particularly in emerging resource-constrained and evolving resource-rich environments. To investigate various issues in accessing spatial information in these two environments, this thesis conducts the following five studies.

Our first two studies present histogram-based selectivity estimation techniques for spatial selections and spatial joins respectively. In our first study, *Cumulative Density* scheme gives very accurate results for spatial selections (usually less than 5% error) with negligible and constant estimation time cost, regardless of dataset or query window parameters. Similarly, *Geometric Histogram* scheme proposed in our second study can accurately quantify the selectivity of spatial joins.

Using a detailed cycle-accurate energy estimation framework and four different memory resident datasets, our third study examines the pros and cons of three previously proposed spatial indexing alternatives on a mobile device. This study is the first one to analyze spatial index structures from both the energy and performance angles, and its key contribution is in pointing out that performance and energy do not always go hand in hand. It also shows that the nature of

the query also plays an important role in determining the energy-performance trade-offs. Further, technological trends and architectural enhancements are influencing factors on the relative behavior of the index structures.

Our fourth study indicates that a distributed index structure spanning the disks of the workstations in a cluster can provide an efficient shared storage structure to access spatial data in high performance environments. This goal can be attained without significantly compromising the index creation time.

Finally, in our ongoing study of the spatial (location-related) information on the World Wide Web, we offer four hypotheses about this kind of information that can be used to build a geo-spatial crawler to effectively retrieve spatial (location-related) information from the World Wide Web.

# Table of Contents

# List of Tables

# List of Figures

# Acknowledgments

My thesis work has been the most exciting ingredient of my intellectual adventure at Penn State. Considering how easily I can be distracted, my adventure would have ended fruitless without the diligent guidance of my thesis advisor, Dr. Anand Sivasubramaniam. Dr. Anand has not only guided me through his multidisciplinary expertise, but more importantly inspired me to become an independent thinker. By consistently raising the performance bar for himself and for me, he has motivated me to achieve more, and to be more self-confident. I am also very grateful that despite his tight schedule, Dr. Anand always found the time for me. Above all, I am most indebted to Dr. Anand for making my research adventure at Penn State a meaningful and fruitful one.

In addition to Dr. Anand, it is my great fortunate to have other excellent researchers on my thesis committee. Not like conventional committee members, they all have been active participants in my research adventure at Penn State. Professor Donna Peuquet was a Guggenheim Fellow and has an influential voice in Geographic Information Systems (GIS) community. Only after many sparkling conversations with her, I started to appreciate the beauty of GIS and realize many challenges in managing spatial information. To help me understand GIS from a practical perspective, she arranged a summer internship for me at ESRI INC that is the world leading GIS software company. Even today, almost four years later, I often feel benefited from this invaluable experience. Dr. Vijaykrishnan Narayanan, an expert in Low Power System Design and VLSI Systems, offered me opportunities to explore various issues in accessing spatial information holistically. Our collaborative work on examining energy behavior of spatial index structure

was the first of the kind. My last committee member is Dr. C. Lee Giles who is the David Reese Professor at the School of Information Sciences and Technology, and an international figure in several computing related disciplines. Dr. Giles opened my eyes to the intriguing field of web computing. Our ongoing work on geo-spatially crawling the web is very promising and I am excited to pursue it further. All my committee members have contributed a great amount to my thesis work, and I would like to extend my great gratitude toward them for their kind help.

Many other people have also assisted me notably during my stay at Penn State. Liujian Qian, Ji Jin, Rongqing Lu, Zhenyu Yang and Sudhanva Gurumurthi worked closely with me on various projects. Ajit Banerjee, Chun Liu, Gokul Kandiraju, Mangesh Kasbekar, Shailabh Nagar, Murali N. V, Aniruda Vidya and Yanyong Zhang have expanded my intellectual horizon beyond my imagination. Ann Cavanaugh, Karen Corl, Vicki Keller and Suzie Mostoller have always brought smiles to my face. All these people and many others have made my Penn State journey a very enjoyable one.

No matter what adventure I take, there are two places I can always seek for inspiration, support and love: my parent's family in China and my American family at New Jersey, US. Despite enduring my long absence from home, my parents always think for my interests and encourage me to pursue whatever interests me. My American family, Hank and Milly, are truly godsend. Their broad spectrum of knowledge, unyielding characters and explorative spirit has awakened my dormant curiosity, and will always inspire me to take many adventures to come.

It is to them - my families, I dedicate this work.

# Chapter 1

# Introduction

## 1.1 Motivation

Spatial information has always been a need of the human society. The long history and continuing evolvement of maps well illustrates this point. Spatial information is becoming prevalent in numerous applications[108]. Geographical Information Systems (GIS), image processing, navigation/positioning, demography, epidemiology, terrain analysis, mining, military planning and logistics, computer-aided design and robotics, are just few of the domains that can benefit from efficiently managing spatial information. Over the last two decades or so, Spatial Database Management Systems (SDBMS) [108] have achieved a great deal in storing, processing and retrieving spatial information. The continuing advancement of technologies, however, persistently pushes the envelope and need for of accessing information, particularly in emerging resource-constrained and evolving resource-rich environments.

With users demanding access to computational resources and information whenever and wherever they choose, computing is becoming a pervasive and ubiquitous part of everyday life. This trend has stressed the need to access spatial information, which is a natural component of the ubiquitous computing, in a resource-constrained environment: limited computational power, storage capacity, battery energy, and connectivity. These operating conditions are very different to those in regular desktop environments, and need to be examined carefully to enable us deliver the desired result efficiently.

On the other hand, accessing the spatial information in the resource-rich environments also evolves: the volume and complexity of the spatial data has continuously grown as is apparent from the NASA's EOSDIS project that has been collecting raster images continuously arriving at a rate of 3-5 Mbytes/second from satellites orbiting the earth. In addition to just being able to handle these large data sets, we need to effectively query and analyze them. Despite the increase of computing power and storage capacity in the resource-rich environment, utilizing them to meet the growing demands is not a trivial task.

Further, the exponential growth of the World Wide Web has created a new resource-rich environment, where the information itself is abundant. Accessing and utilize spatial information on the web remains unknown to a great extent.

Striving to have a better understanding of accessing spatial information in these different environments, this thesis offers convincing solutions to two open problems on estimating selectivity of spatial operations; examines, for the first time, the energy behavior of spatial index structure in a resource-constrained environment; develops various methods on distributing a spatial index structure in a resource-rich cluster environment; and posts a novel problem on crawling the web for spatial (location related) information.

## 1.2 State-of-the-Art

Before specifying the contribution of this thesis, we discuss its related start-of-the-art techniques.

### 1.2.1 Selectivity Estimation

Accurate selectivity estimation is crucial for drawing the optimal query execution plan, and is also useful when implementing data mining functionalities. Since selectivity estimation deals with the general property of a particular operation, it could be useful in both resource-constrained and resource-contained environment. Most previous analysis attempts have made certain simplifying assumptions about the datasets and/or queries to keep the analysis tractable. As a result, they may not be universally applicable. To develop more universally applicable selectivity estimations techniques, we should let the spatial data speak for itself, and avoid making assumptions as much as possible.

### 1.2.2 Accessing Spatial Information in Resource-constrained Environment

The proliferation of mobile and pervasive computing devices has brought energy constraints into the limelight, together with performance considerations. Energy-conscious design is important at all levels of the system architecture, and the software has a key role to play in conserving the battery energy on these devices. With the increasing popularity of spatial database applications, and their anticipated deployment on mobile devices (such as road atlases and GPS based applications), it is critical to examine the energy implications of spatial data storage and access methods for memory resident datasets on such devices.

### 1.2.3 Accessing Spatial Information in Resource-rich Environment

A SDBMS system must efficiently store, retrieve and process the voluminous data that it needs to handle. Employing processing and storage parallelism is essential to provide us scalable long-term solutions. With the demise of many custom-built parallel machines, it is imperative

that we use off-the-shelf technology to provide this parallelism. One viable alternative we need consider is a closely coupled network of workstations.

### 1.2.4 Accessing Spatial Information on World Wide Web

Just as many other kinds of information, spatial information can also be found on the World Wide Web. McCurley [79] estimated that 4.5% of all the web pages contained a US zip code, 8.5% contained a recognizable phone number, and 9.5% contained at least one of them. Even if a web page doesn't explicitly contain location information such as a zip code, it may still be spatially related. For instance, Ding et al. [29] map web pages into a geographic hierarchy. However, it is not clear how to automatically retrieve location-related information from the World Wide Web, and how to utilize them.

## 1.3 Overview of the thesis

The objective of this thesis is to investigate various issues in accessing spatial information in both resource-constrained and resource-rich environments. We will detail this investigation in the rest of the thesis.

Chapter 2 proposes a set of five analysis techniques to estimate the selectivity and number of index nodes accessed in serving a spatial selection (range query). The underlying philosophy behind these techniques is to maintain an auxiliary data structure called a density file, whose creation is a one-time cost, which can be quickly consulted when the query is given. The schemes differ in what information is kept in the density file, how it is maintained, and how this information is looked up. It is shown that one of the proposed schemes, called *Cumulative Density (CD)*, gives very accurate results (usually less than 5% error) using a diverse suite of point and

rectangular datasets, that are uniform or skewed, and a wide range of query window parameters. The estimation takes a constant amount of time, which is typically lower than 1% of the query execution time, regardless of dataset or query window parameters.

After the scrutiny of selectivity estimation techniques for spatial selections, this thesis explores the problem of estimating selectivity for spatial joins in Chapter 3. Apart from extensively evaluating the accuracy of sampling techniques for the very first time, we present two novel histogram based solutions for spatial join estimation. Using a wide spectrum of both real and synthetic datasets, it is shown that one of our proposed schemes, called *Geometric Histograms (GH)*, can accurately quantify the selectivity of spatial joins.

While there has been extensive prior research on spatial access methods on resource-rich environments, we are, perhaps, the first one to study their suitability for resource-constrained environments, especially from an energy angle. Armed with a detailed cycle-accurate energy estimation tool called SimplePower, Chapter 4 examines the pros and cons of three previously proposed spatial indexing alternatives from both the energy and performance angles. The results show that there are both performance and energy trade-offs between the indexing schemes for the different queries. The nature of the query and the actual dataset also play an important role in determining the energy-performance trade-offs. Further, technological trends and architectural enhancements are influencing factors on the relative behavior of the index structures. The results from this study can be beneficial to the design and implementation of embedded spatial databases, accelerating their deployment on numerous mobile devices.

In Chapter 5, we explore techniques for distributing a spatial data structure (R-tree) across a cluster of workstations. In addition to providing a framework to explore design alternatives in distributing the R-tree across workstations, we develop an extensive system to implement

and evaluate these alternatives. Specifically, we implement four distribution schemes, and evaluate their performance (pros and cons) for insert and spatial search operations on different data sets. We also experimentally study the impact of different search parameters and network speed on the performance of the distribution schemes. It is shown that a distributed index structure spanning these workstations can significantly lower the response times and increase the throughput for spatial searches compared to a traditional implementation.

Finally, before summarizing our contributions and pointing out a few research directions in Chapter 7, we report the current status of our ongoing work on building a geo-spatial crawler in Chapter 6. Here, we offer four interesting hypotheses that can be used in the further development of the geo-spatial crawling techniques.

# Chapter 2

# Selectivity Estimation for Spatial Selections

## 2.1 Introduction

Analysis of the performance of spatial operations [108] becomes even more essential with the choice of numerous index structures [82, 36, 48, 53, 52, 97, 106]. Performance analysis can help better understand the suitability of a data structure for different input datasets (both size and spatial distribution). Given a dataset, we could use analysis results to objectively choose between different indexing alternatives. After choosing an index structure, analysis results could then be used to efficiently build/layout/fine-tune the structure within the purview of its definition. Finally, analysis is extremely important for query optimization. The cost and number of data items that are retrieved by a query would be very useful to determine the execution plan of a query for best performance.

There are several interesting queries that could be posed to a spatial database. These include range queries (selecting items that overlap a given query window), nearest neighbor queries, joins and other topological queries. Of these, the spatial selection is, perhaps, the most common, and has been widely used as the subject of analysis in other related studies [116, 32, 96, 117, 68, 119, 87] as well. With range queries, one is interested in finding out how many data items will be retrieved (*selectivity*) and what will be the I/O complexity (number of *nodes accessed* in the index structure) in servicing the query. These two measures reflect the I/O and CPU processing costs that would be incurred by the query, with the former factor usually

being more dominant. This chapter focuses on estimating (analyzing) the selectivity of range queries on spatial databases. Our proposed techniques can also be used to estimate the nodes that would be accessed in servicing a spatial selection in the associated spatial index structure." We demonstrate this by using the packed R-tree [68] as a case-study.

There have been several previous attempts at analyzing R-tree performance for range queries [35, 116, 16, 32, 96, 117, 119, 87, 68]. Most of these studies are either limited by the kind of datasets that they can be applied to, or make some simplifying assumptions along the way to keep the model tractable. As a result, they can give inaccurate results when the datasets deviate from such assumptions. Further, many previous studies examine average case behaviors (average the errors in estimation over numerous query windows), and this may not necessarily reflect certain gross errors for specific windows (and such windows could be important for an application).

The underlying philosophy of the analysis techniques presented in this chapter [66, 7]is that they should make little or no assumption about the dataset. Any information that is dataset dependent should be drawn from the dataset itself. As a result, these techniques are universally applicable, regardless of the dataset or the application that they are used for. The common theme between these different techniques is to use an auxiliary data structure, called the *density file*, which maintains sufficient information (*histograms*) about the dataset that is necessary to conduct the estimation. The density file creation is a one-time cost. When the query is given, the density file is "quickly" looked up to determine selectivity and nodes accessed. There has been prior work [107, 93] on using histograms for estimating query performance in relational databases, but there are only a few forays [81, 1] on such techniques for spatial/multidimensional databases. [81] is an early study that has proposed a technique for building equi-depth histograms

for multidimensional point datasets. There has been a recent study [1], concurrently undertaken with the work that is presented in this chapter, on different ways of maintaining histograms for spatial databases. The differences between these studies are explained in Section 2.2.

This chapter considers both point (sizeless and shapeless objects) and rectangular datasets, and proposes five analysis schemes for the former and two for the latter (the rectangular schemes can be used for points as well without any loss of generality). It should be noted that rectangles can also be used to abstract more complex spatial objects (as Minimum Bounding Rectangles), and the proposed schemes can be used in such cases. These schemes differ in what information is maintained in the density file, how it is maintained, and how it is looked up for estimation. Using a diverse (both real and synthetic, that are uniform or skewed) suite of datasets and different query window parameters (size, location and aspect ratio), it is shown that one of our schemes (called *Cumulative Density*), gives very accurate estimations for selectivity and nodes accessed for each query window, with errors less than 5%. It gives much lower errors than most of the previously proposed techniques. It provides this accuracy at a (time) cost that is less than 1% of the actual query execution time. The storage overheads for maintaining the density file are tolerable as well.

The rest of this chapter is organized as follows. The next section gives a quick overview of previous analysis attempts on spatial selection performance. Section 2.3 presents the proposed analysis techniques for point and rectangular datasets. Section 2.4 gives results from the analyses using a spectrum of datasets and query windows. Finally, Section 2.5 summarizes the contributions of this chapter and outlines directions for future work.

## 2.2 Related Work

Estimation of spatial selection performance on spatial data has been shown to be extremely important [12, 81]. Consequently, there is a large body of literature on this topic. These techniques, however, are limited either to the kind of datasets that they can analyze (points, rectangles, etc.), and/or make simplifying assumptions about the dataset (uniform, skewed following a certain rule, etc.) or query windows.

The first set of techniques [35, 116] make an assumption about the dataset being more or less uniformly distributed in space. [35] presents the first known analysis of R-trees, by transforming objects into higher dimensional space. [116] uses the size of the query window to find out how many leaf nodes will be covered by the query for the uniform dataset. The internal nodes that will be touched can then be examined recursively to estimate the disk accesses. Since these studies assume uniform distribution of the data, they do not have to be concerned about the location of the query window. These estimations can, however, become inaccurate for skewed datasets.

Some other studies [32, 16], though not restricted to uniform datasets, focus specifically on point datasets. Faloutsos and Kamel [32] show that certain point datasets behave as mathematical fractals, and calculate the fractal dimension of the datasets. Using this, they can find out the number of disk accesses at each level of the R-tree. However, this analysis is restricted to point datasets, and also requires that the R-tree be well-built and the aspect ratio of the MBRs of its nodes be close to 1.

Similarly, Proietti and Faloutsos [96] focus on region datasets for their analysis. They show that certain region datasets can be packed into MBRs having a quite uniform aspect ratio,

and the areas of these MBRs obey the REGAL law. This observation is used to estimate the number of MBRs intersecting the given query window.

Finally, there are studies [117, 119, 87, 68] which have looked at both point and region datasets, that are both uniform and skewed. [117, 119] use the concept of density (average number of data entries that contain a given point) to analyze R-tree performance. The density is calculated level-by-level, beginning at the leaf. This technique assumes that the density is uniform within the dataset. The authors suggest that it could be extended to non-uniform datasets, by splitting the spatial extent into multiple regions, each of uniform density. This can, however, become a non-trivial decomposition. Further, their technique assumes that the node MBRs have unit aspect ratios.

With the bulk-loaded packed R-tree structure, Kamel and Faloutsos [68] propose a simple formula to estimate the number of pages (nodes accessed) that will be retrieved by a spatial selection, which is a function of the query window size and the average size of the Minimum Bounding Rectangles (MBR) of the R-tree nodes. This formula would apply to any R-tree (built using any technique), as long as the MBRs of the nodes are available. Since this technique does not take query window location into account, it may not always be accurate, though the accuracy may be acceptable in the average case. Aref and Samet [13] later extended this model for selectivity estimation of spatial joins. A probabilistic model for different spatial selection classes on spatial data structures, that is relatively independent of the choice of the data structure and dataset, is conducted in [87]. This study makes a couple of assumptions in deriving the model: all query windows have unit aspect ratio, and all query windows are either of the same area or request the same number of data items. Such assumptions may not necessarily be valid for real applications.

Most of the above techniques fall under what has been characterized as parametric techniques [1], which try to mathematically model the data based on certain assumptions. Spatial datasets are likely to be very diverse. Consequently, not all of the above techniques can be used to analyze the performance of all datasets. We believe that an estimation technique should make little or no assumptions about the input dataset. Any information that it would need should come from the dataset being analyzed itself, and this information should be provided without adding significant overheads. Techniques adhering to this philosophy would be universally applicable, regardless of the dataset or the application that it is being used for. These techniques, typically, use auxiliary data structures called histograms, which partition the space into buckets and keep track of how many data items fall within each bucket.

A very recent study [1], undertaken concurrently with this work, has examined different ways of constructing histograms for spatial databases to estimate selectivity, and has proposed a novel scheme (called *MinSkew*) that is shown to give fairly accurate estimations. There are several similarities between some of their suggestions and the techniques presented here. For instance, our DH scheme uses the equi-area partitioning suggested in [1], with the difference that it does not optimize empty spaces/regions. Our DHC scheme is intended for such optimizations. The NDH scheme discussed here, is almost identical to the R-tree index-based grouping suggested in [1]. However, there is a key difference between the two studies. Except for the NDH scheme, all the others in this chapter use equi-width (equi-area) and non-overlapping buckets unlike the ones used in [1]. As a result, it is rather straightforward in our schemes to find the relevant buckets for a query window. Query estimation is thus fast and has very low memory requirements. Estimation for the schemes in [1], on the other hand, requires a search to find the relevant buckets. As a result, those schemes try to keep the number of buckets relatively small

so that they fit in main memory. The techniques detailed here do not have such restrictions, and we can potentially go for a large number of buckets for better accuracy. We are able to maintain non-overlapping buckets even with rectangular data items, using a novel idea (derived from simple geometric properties of rectangles) whereby a rectangular object is counted in exactly one bucket. To our best knowledge, no previous study has pursued such an idea.

Another common observation about all the above studies, is that the estimation accuracy is evaluated using average case behavior (i.e. numerous query windows are fed to the model and the error in estimation is averaged over all these queries). While this may be a viable approach to discuss the overall quality of different modeling techniques, it is important to note that there can be gross inaccuracies for certain specific windows (and such windows may be important workloads for an application). Instead, one should try to conduct studies with different query window sizes and locations, and try to understand the accuracy of the estimation for each of these windows.

## 2.3 Analysis Techniques

There are two main costs in searching for objects intersecting a rectangular query window. The first is the cost of computing the intersection between the data entries and the query window. The second is the cost of retrieving the items from the disks. The number of data items that will be retrieved (called the *selectivity (s)*) has a direct bearing on both these costs. Further, the retrieval cost will also depend on the *number of nodes (n)* in the index structure that will be touched by the query. In the rest of this discussion, we present a set of techniques for estimating the selectivity for point and rectangular spatial data sets. We also illustrate how these techniques

can be used to estimate the number of nodes in the index structure that will be accessed, using the packed R-tree structure [68] as a case-study.

The R-tree [48, 74], proposed as an extension to the B-Tree structure, is one of the most popular spatial data structures. Many variants of the R-Tree, such as $R$+-Tree [106] and $R^*$-Tree [82] have been proposed. They differ in the algorithm that is used for insertion, specifically in splitting a node of the tree when its subtree is filled. They attempt to give better balanced (and efficient) trees by dynamically adapting to the insertion pattern/sequence. However, these structures can become inefficient when the database of spatial items is static (and known *a priori*). In such cases, one should use bulk-loading techniques rather than insert item by item to build the data structure. Roussopoulos and Leifker [99] use packed R-trees for such static databases to lower response times. Further, Kamel and Faloutsos [68] suggest using Hilbert value (a linearization technique for multidimensional space [45, 64]) as the index for the bulk-loaded R-tree. This would help optimize (localize) the number of nodes traversed in serving a query. Typically, such R-trees are built in a bottom-up fashion, level by level.

There are three main reasons for using bulk-loaded, packed R-trees to illustrate node access estimation. First, our research project [8] is exploring efficient database support for geographic information (GIS), and the datasets in this domain are usually static. Many other applications use static spatial datasets as well, and our techniques would apply to all these domains. Second, packed R-trees have high space utilization (close to 100%) compared with their dynamic counterparts. This reduces the size of the tree, localizes (optimizes) the nodes traversed in serving a query, and reduces response times. Finally, without loss of generality, bulk-loaded R-trees help us keep our analysis of node access estimation relatively simple. It is for these reasons that other similar studies [68] analyzing the performance of R-trees have used bulk-loaded bottom-up

R-trees as well. The reader should note that the use of a packed R-tree to illustrate node access estimation does not in any way mitigate the generalization of our schemes. One could very well feed the bounding boxes of the nodes in the index structure to the selectivity estimation process to find the number of nodes that will be accessed. In fact, our rectangle dataset schemes follow this approach. The bulk-loaded R-tree, that we use, is built from a sorted list (based on Hilbert ordering as suggested in [68]) of the data items in a bottom-up fashion.

For the datasets, we focus on points and rectangles, which capture interesting facets of spatial information. Points are shapeless/sizeless objects, while this is not the case for rectangles. Rectangles can also be used to represent/abstract other objects (such as lines, polylines, polygons etc.). In such cases, the rectangles represent the Minimum Bounding Box of the objects, and there needs to be an additional step (called the refinement step) to process the retrieved rectangles (from the filtering step) and find out which actually pertain to the query. The techniques presented here will then apply to an analysis of the performance of the filtering step. It has often been argued that this is, perhaps, a more dominant cost of a spatial selection since it requires I/O.

### 2.3.1 Overview

Our techniques can be briefly summarized as follows. We construct an auxiliary data structure (which we call the *density file*) from the original dataset, in addition to the R-Tree at the time of building. This density file contains sufficient information - how many data items are contained in different regions/cells of the spatial extent - about the dataset. When a query is given, the density file is *quickly* looked up to procure the necessary information. The size of the density file, the time for constructing this file, and the time for looking up the required information are the issues that one needs to keep in mind, as will be discussed for each technique.

The techniques differ in what information is kept in the density file, how it is kept, and how the information is looked up. Some common symbols/definitions that are used in the following discussion are given in Table 2.1.

| Symbols | Definitions |
|---|---|
| $h$ | Hilbert order ($4^h$ cells) for point datasets<br>Level of gridding ($4^h$ cells) for rectangle datasets |
| $S_e$ | Size in bytes of a density value |
| F | Node capacity or fanout |
| $N_s$ | Dataset size |

Table 2.1. Common Symbols and definitions

### 2.3.2 Techniques for Point Datasets

Point data has only position information, making them easier to process. When a (rectangular) query window is given, we need to find out how many points of the dataset fall within this window (selectivity). The leaf nodes that the selected points fall on is determined by the Hilbert values of the points. Subsequently, we use a recursive procedure to find out how many internal nodes of the R-tree will be accessed to get to these leaf nodes.

The common theme in the following schemes is to first partition the spatial extent into grid cells in the same way that is used to assign a Hilbert ordering (number) for the data items [45]. The density file is then just a histogram of the number of points that fall within a specific Hilbert range. It is important for the reader to note that the Hilbert order [45] (the level to which the spatial extent is recursively broken down) will have an important effect on the size

of the density file as well as on the accuracy of the estimation. The schemes differ in how the histogram is maintained within the density file.

#### 2.3.2.1 The Density Histogram (DH) Scheme

The DH scheme uses a straightforward representation of the density information. The density file contains the number of points that fall within each Hilbert grid cell, and the file is maintained in increasing Hilbert cell order. The Hilbert cell number can be used as an offset into the file to directly get the corresponding density (number of points within this cell). The size of the file is thus dependent on the number of grid cells, which in turn is a function of the Hilbert order that is used to recursively break down the spatial extent. For a given query window, the selectivity ($s$) and number of nodes traversed ($n$) can be estimated as follows.

**Selectivity:** The query window is broken down into a sequence of Hilbert ranges that it covers, based on the Hilbert order that has been used to create the density file. There is an approximation being made here in aligning/extending the query window to the boundaries of grid cells. It should also be noted that there is a sequence of (potentially non-contiguous) Hilbert ranges that are generated from the query window. These ranges are looked up in the density file to find out how many points fall within each range (density), and then the densities are added up to get the selectivity. To make it more efficient, we keep cumulative densities (sum of densities from grid cell 0 until that grid cell) in the density file, so that finding the density within a range will require looking up just two values (the ends of the range) and subtracting one from the other.

**Nodes Accessed:** We could find the number of nodes accessed by a query if we had some knowledge about which leaf nodes the selected data items reside on. It is rather easy to figure

out this information from the packed R-tree algorithm, since the leaf nodes contain data items sorted by Hilbert order (and each leaf node contains the same number of data items). As with the selectivity method, we can break the query window into Hilbert ranges. For a range, we could look up the density (specified as a cumulative density from grid cell 0) information for the lower end of the range. This number divided by the number of data items pointed to by a leaf node, would specifically identify the leaf node where the range starts. Similarly, the density information for the end of the range can be used to find out on which leaf node the range ends. Once we identify all the leaf nodes that will be accessed, we can use the same method to recursively move up the tree to find out what nodes will be accessed at each level.

#### 2.3.2.2 The Density Histogram Compression (DHC) Scheme

The problem of the DH Scheme is the storage space. For each Hilbert cell, a (cumulative) density value is stored in the file, regardless of whether there are any data items present in that cell or not. The size of the density file grows exponentially with the Hilbert order. Note that higher the order, higher would be the level of accuracy most of the time. Hence, if one wants to tolerate a relatively low estimation error, then the DH scheme would not be a very scalable alternative. The DHC scheme tries to compress the density file information of the DH scheme, with the same underlying algorithms used to find selectivity and nodes accessed.

Examining the density file of the DH scheme carefully, one can find characteristics that can help optimize space requirements. First, when the Hilbert order is high, the spatial extent gets divided into very small grid cells, with a large number of cells having no data items within them. Second, neighboring grid cells tend to have similar densities, and densities change more gradually from one cell to another. This is usually the case for datasets with points uniformly

distributed in the spatial extent. Even for skewed datasets, we can have regions within which the grid cells have similar densities.

These observations suggest that we can reduce the size of the density file by compressing the storage required to represent neighboring grid cells with similar densities. The scheme can be explained as follows. We initially start with the density file of the DH scheme. We compare the density of each set of 4 consecutive grid cells (recall that Hilbert ordering recursively breaks down the space into 4 regions). If they are similar (close enough), then they are represented by only one entry in the density file. The "similarity" check is done by comparing the coefficient of variation (standard deviation divided by mean of the density values for the four cells) with a certain threshold. If the value is less than the threshold, then the 4 cells are combined into one value, else they are left as four cells. This procedure is then recursively carried out for the next lower level of the Hilbert order, and so on. The recursion stops when either there are no more grid cells to be merged, or when the number of grid cells to be considered is just one.

This scheme is expected to lower the size of the density file compared to the DH scheme, albeit at a higher cost required to build the density file. Further, finding the offset in the file for a particular grid cell is no longer straightforward as in the DH scheme. A slightly higher price has to be paid during estimation. We use a simple index structure to improve the performance of the lookup operation on the compressed density file. Other than this, the selectivity and nodes accessed estimation algorithms are the same as for the DH scheme.

#### 2.3.2.3 The Node-based Density Histogram (NDH) Scheme

Another way of reducing the size of the density file is by keeping the information at a slightly coarser level. In the DH scheme, the information was maintained at a grid cell granularity, thus (potentially) giving a finer level of accuracy in estimating both selectivity and nodes accessed. Instead, we maintain the file on an R-tree leaf node basis (i.e. one entry in the density file for each leaf node of the tree, with each entry containing the Hilbert range of the cells covered by that node together with the number of data items within that range). The selectivity and nodes accessed are calculated as follows.

**Selectivity:** Convert the query window into a sequence of Hilbert ranges as before. Next, we find the leaf nodes that intersect these ranges from the density file. Experimentally we have found that using a binary search within the density file to find the leaf node that intersects with the start of the first range, and then a linear search from that point for subsequent ranges works rather well. For each intersecting leaf node, we approximate the number of data items that would be retrieved as $x\%$ of the items within that node, where $x$ is the percentage of the node's Hilbert range that intersects the query window. This approximation assumes that the data items within a leaf node are uniformly distributed within the Hilbert range of that node. By summing this number over all the intersecting leaf nodes, we get the required selectivity.

**Nodes Accessed:** The same algorithm used in the DH scheme, which calculates the internal nodes that are accessed after the leaf nodes have been identified, is used here as well.

It should be noted that the density file of NDH essentially maintains equi-depth histograms [81]. Each bucket corresponds to an R-tree node, with the fanout determining the depth

of the bucket. A similar scheme has been used as one of the options in a more recent study [1]. However, since the points are sorted by Hilbert order in our approach, there is not much (consecutive buckets may at most have one Hilbert value in common) overlap between the buckets.

#### 2.3.2.4 Comparing Point Dataset Schemes

Before we proceed to rectangular dataset schemes, we give a brief summary of the three schemes for points. We use four criteria in discussing each scheme. These include the size of the auxiliary data structure (density file) that they use, the time for creating/building this auxiliary data structure (which is a one-time cost incurred at the time of building the R-tree), the time taken for estimating the selectivity and nodes accessed, and the accuracy of the estimation.

DH requires a large density file ($4^h * S_e$), which grows exponentially with the Hilbert order. DHC compresses this file, based on similarity between successive grid cells. The degree of compression would depend on the variation of the number of data items that fall in successive grid cells, together with the threshold that is used to qualify this variation as significant to warrant extra storage. The density file size is independent of Hilbert order for the NDH scheme, clearly making it a winner in terms of the size criterion. The density file size in NDH ($\frac{N_s}{F} * S_e * 2$) is directly proportional to the number of leaf nodes in the corresponding R-tree, with each leaf node requiring two Hilbert values (to indicate a range).

To create the density file in the DH scheme, we need to histogram the points based on which Hilbert grid cell they fall on. While one could use this straightforward approach for creating the density file, the performance may be rather poor since successive data items may fall on different grid cells (if the dataset is not sorted) resulting in a lot of I/O operations (the file can get quite large). Instead, we first generate the Hilbert values for the points (which would

be done in any case to build the bottom-up R-tree), sort them based on these values, and then histogram them to avoid thrashing effects. The creation of the density file for DHC would be even more expensive since the density file has to be compressed based on coefficient of variation. NDH takes very little time to build the auxiliary structure. Since the points are already sorted based on Hilbert order (to build the R-tree), we can just examine the two extreme points of each leaf node to find out its range.

In all three schemes, the first step in estimation is to break up the given query window into a sequence of Hilbert ranges. This would itself take up time that is dependent on the Hilbert order. Next, selectivity is determined by examining the density file for these ranges. This is a straightforward lookup (using the Hilbert value as an offset) of the density file for DH. DHC requires extra time, since there is another index structure within this file to get to the information. NDH requires binary and some linear searches within this file to find intersecting leaf nodes, and is, perhaps, the more expensive of the three. Subsequently, estimation of nodes accessed uses the same algorithm in all three schemes.

Finally, the estimation accuracy is largely dependent on the Hilbert order that is used for linearizing the dataset and breaking down the query window for all three schemes. For the DHC scheme, it is also dependent on the threshold that is used to detect variation in densities. In the NDH scheme, the accuracy of the selectivity is dependent on the spatial distribution of the data points contained within each leaf node of the R-tree.

| | DH | DHC | NDH |
|---|---|---|---|
| Density File Size | $4^h * S_e$ | Smaller than DH Depends on $h$, threshold and dataset patterns | $\frac{F}{N_s} * S_e * 2$ |
| File Creation Time | Time to generate Hilbert values + Time to sort + Time to histogram | Larger than DH | Smaller than DH |
| Estimation Time | Time to convert query window to Hilbert ranges + Time to lookup density file for each range | Higher than DH | Higher than DH |
| Estimation Accuracy | Depends on $h$ | Depends on $h$ and threshold | Depends on $h$, uniformity in leaf nodes |

Table 2.2. Point Dataset Schemes at a glance

### 2.3.3 Techniques for Rectangle Datasets

Rectangle datasets pose a more difficult problem than point datasets since they contain size information in addition to position information. The point dataset schemes may not necessarily work for rectangle datasets, because of this extra dimension to the problem. One could think about extending two dimensional Hilbert space into three or more dimensions (as was mentioned in [68]), but that would need a high amount of computation. Further, it is not straightforward to convert a query window into three dimensional Hilbert ranges while still maintaining the spatial relationships, i.e. objects that fall into those ranges should spatially intersect the query window.

We propose two schemes below that use simple geometrical properties of rectangles to address this problem, while still providing non-overlapping buckets. There are a couple of differences from the previous schemes. In the point dataset schemes, Hilbert space and ordering was used to grid the spatial extent. In the following two schemes, we do not really care, because there is no need to get a linearization of the spatial extent. The spatial extent is, instead, gridded

into cells by just drawing a number of vertical (columns) and horizontal (rows) lines. A cell is then denoted by its row and column. The density file is looked up by using a 2-dimensional offset (a row and column number).

The second difference from the point schemes is in what each entry of the density file contains. We cannot keep just the center point information of the rectangular items (i.e. each cell contains the number of rectangles whose center points fall within that cell), since this would loose the size information. Neither can we record how many rectangles intersect each cell either, since we would end up double/multiple counting the rectangles in estimation. In the following two schemes, we illustrate what we need to maintain within each grid cell to avoid multiple counting of rectangles without sacrificing size information.

#### 2.3.3.1 The Incremental Density (ID) Scheme

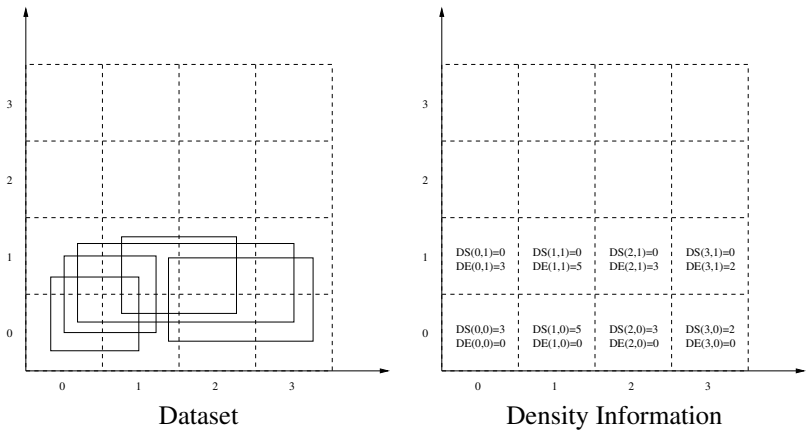



Fig. 2.1. Density Information in the ID Scheme

In this scheme, the density file keeps track of the information on an incremental basis. Specifically, for each grid cell we keep two values: (a) the number of rectangles whose bottom side/edge falls (intersects) on that cell ($DS(i,j)$); (b) the number of rectangles whose top side/edge falls (intersects) on that cell ($DE(i,j)$). This is shown pictorially on the right side of Figure 2.1 for the rectangular dataset on the left side.

**Selectivity:** We can find out how many rectangles ($N(qxl, qyl, qxh, qyh)$) intersect with a query window $(qxl, qyl, qxh, qyh)$ (the lower-left corner is $(qxl, qyl)$ and the upper-right corner is $(qxh, qyh)$) as follows. Let $S(xl, yl, xh, yh)$ denote the number of rectangles that start in region $(xl, yl, xh, yh)$ (i.e. whose lower edges intersect with this region), and let $E(xl, yl, xh, yh)$ denote the number of rectangles whose top edges intersect with this region. We can then use the following equation to calculate $N$:

$$N(qxl, qyl, qxh, qyh) = S(qxl, 0, qxh, qyh) - E(qxl, 0, qxh, qyl - 1) \tag{2.1}$$

Figure 2.2 illustrates this observation with an example. The query window covers $(1, 2, 2, 3)$, for the dataset with 11 rectangles numbered $a$ through $k$. For this example, $S(1, 0, 2, 3) = 11$ (i.e. all the 11 rectangles), and $E(1, 0, 2, 1) = 4$ (i.e. rectangles *a*,*b*,*c* and *d*). So $N(1, 2, 2, 3) = 11 - 4 = 7$.

We determine $S$ and $E$, from $DS$ and $DE$ respectively, as follows. Let $RS([xl, xh], j)$ represent the number of rectangles whose lower edges intersect with grid cells of row $j$ between

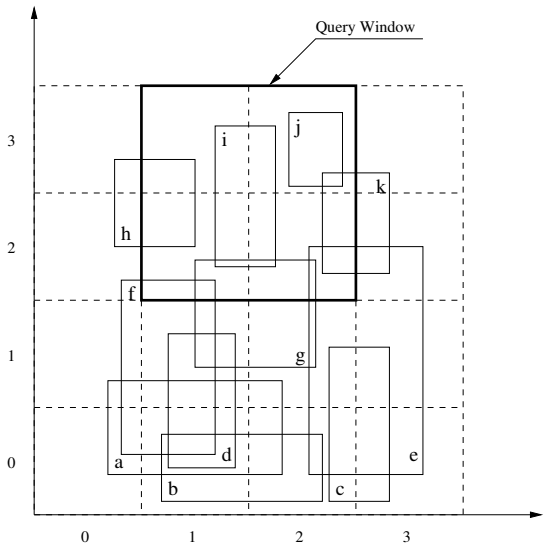



Fig. 2.2. Calculating Rectangles in a Query Window

columns $xl$ and $xh$.

$$RS([xl, xh], j) = DS(xl, j) + \sum_{i=xl+1}^{xh} MAX(DS(i, j) - DS(i-1, j), 0) \qquad (2.2)$$

For example, in Figure 2.1, $RS([0, 2], 0) = 3 + (5 - 3) + 0 = 5$, which means that there are five rectangles whose lower edges intersect with grid cells (0,0), (0,1) and (0,2). The following equation can then be used to calculate $S$,

$$S(xl, yl, xh, yh) = \sum_{j=yl}^{yh} RS([xl, xh], j) \qquad (2.3)$$

Similarly, let $RE([xl, xh], j)$ represent the number of rectangles whose top edges intersect with grid cells of row $j$ between columns $xl$ and $xh$. We can then calculate $E$ from $DE$ as

follows,

$$RE([xl, xh], j) = DE(xl, j) + \sum_{i=xl+1}^{xh} MAX(DE(i, j) - DE(i-1, j), 0) \quad (2.4)$$

$$E(xl, yl, xh, yh) = \sum_{j=yl}^{yh} RE([xl, xh], j) \quad (2.5)$$

$RS$ and $RE$, however, are not always accurate. For example, let us look at a situation where two neighboring rectangles have edges that fall on adjacent columns of a row as shown in Figure 2.3 (a). This scheme will not differentiate between this and a single large rectangle (shown in Figure 2.3 (b)). In both cases, this scheme will result in the same density information. The CD scheme, to be discussed next, does not suffer from this inaccuracy.

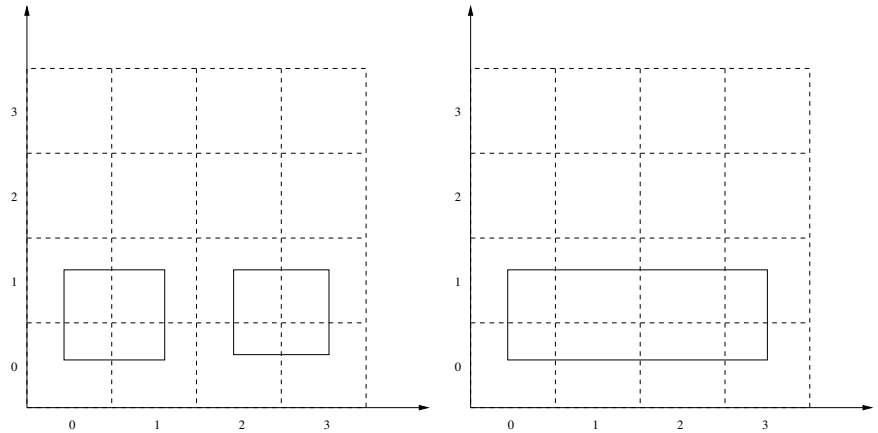



Fig. 2.3. Errors in Estimation for ID

**Nodes Accessed:** We cannot use selectivity information directly to estimate the nodes accessed (as in the point dataset schemes), because the density information is not maintained as Hilbert grids i.e. after the selectivity is obtained, we do not know the Hilbert values for the data items. Although it is possible to divide the universe using Hilbert order as in the point dataset schemes, we would need extra storage and longer computation times (the above equations for selectivity are based on simple geometric properties of rectangles, and there needs to be a level of translation before they can be used if we used Hilbert gridding). Instead, we use an alternate solution, using the property that each node in the R-tree can itself be represented by a rectangle in the spatial extent (the Minimum Bounding Rectangle covering its subtree). It is thus sufficient to examine if this MBR intersects the query window to find out if this node would be accessed. As a result, we maintain the MBRs of all the R-tree nodes (this doubles the space requirement if we use the same degree of gridding as with the selectivity), and use these MBRs themselves as the data for the above selectivity procedure. This would directly give us the number of MBRs (nodes) that intersect the query window.

It should be noted that since we are not using Hilbert grids, or making any other assumptions about the way the R-tree is built, the ID scheme is independent of the algorithm that is used to create the R-tree.

#### 2.3.3.2 The Cumulative Density (CD) Scheme

The main problem with the ID scheme is in the time it takes for estimation, and to a lesser extent the inaccuracy that was pointed out in Figure 2.3. We need to go through each row between 0 and $qyh$, and check the columns in the $[qxl, qxh]$ range to serve a query

$(qxl, qyl, qxh, qyh)$. This becomes expensive with a fine level of gridding (which would increase accuracy), or with large query windows. There is a similar problem with the point dataset schemes, and we have used a cumulative density information to alleviate this problem there. This technique can be used here as well, which gives us the CD scheme.

We grid the spatial extent as in the ID scheme, and we keep four values for each cell $(i, j)$:

- $BS'(i, j)$ (if $BS(i, j)$ is the number of rectangles whose lower-left corners lie in the range $(0, j)$ to $(i, j)$, $BS'(i, j) = \sum_{x=0}^{j} BS(i, x)$);
- $BE'(i, j)$ (if $BE(i, j)$ is the number of rectangles whose lower-right corners lie in the range $(0, j)$ to $(i, j)$, $BE'(i, j) = \sum_{x=0}^{j} BE(i, x)$);
- $US'(i, j)$ (if $US(i, j)$ is the number of rectangles whose upper-left corners lie in the range $(0, j)$ to $(i, j)$, $US'(i, j) = \sum_{x=0}^{j} US(i, x)$);
- $UE'(i, j)$ (if $UE(i, j)$ is the number of rectangles whose upper-right corners lie in the range $(0, j)$ to $(i, j)$, $UE'(i, j) = \sum_{x=0}^{j} UE(i, x)$).

The selectivity $N(qxl, qyl, qxh, qyh)$ can then be calculated as follows:

$$RS([xl, xh], j) = BS(xh, j) - BE(xl - 1, j) \quad (2.6)$$

$$S(xl, 0, xh, yh) = BS'(xh, yh) - BE'(xl - 1, yh) \quad (2.7)$$

$$RE([xl, xh], j) = US(xh, j) - UE(xl - 1, j) \quad (2.8)$$

$$E(xl, 0, xh, yh) = US'(xh, yh) - UE'(xl - 1, yh) \quad (2.9)$$

$$N(qxl, qyl, qxh, qyh) = S(qxl, 0, qxh, qyh) - E(qxl, 0, qxh, qyl - 1)$$

$$= BS'(qxh, qyh) - BE'(qxl - 1, qyh)$$
$$-[US'(qxh, qyl - 1) - UE'(qxl - l, qyl - 1)] \quad (2.10)$$

Instead of examining all density values in the range $[xl, xh]$, we need to access only two values for calculating S, and two for E. Thus, the estimation of selectivity and nodes accessed (a similar method of calculating selectivity with the MBRs of the R-tree nodes as explained with ID can be used to find out nodes accessed) require constant (4 disk accesses and 3 arithmetic operations) time. This time does not depend on the query window size nor the level of gridding (we can use a very fine level for higher accuracy without compromising on estimation time). Further, the CD scheme avoids the inaccuracies shown in Figure 2.3 of the ID scheme.

#### 2.3.3.3 Comparing Rectangle Dataset Schemes

A quick summary of the two rectangle dataset schemes is given in Table 2.3, comparing the size of the density file that is generated, the time taken for generating this information, the time taken for estimation and the accuracy of the estimation.

| | ID | CD |
|---|---|---|
| Density File Size | 2*2*$S_e$*$4^h$ | 2*4*$S_e$*$4^h$ |
| File Creation Time | Time to generate density of rectangles + Time to generate density of node MBRs | At least twice as ID build time |
| Estimation Time | Proportional to $h$ and window size | Constant |
| Estimation Accuracy | Higher $h$ gives higher accuracy | Depends on $h$, more accurate then ID |

Table 2.3. Rectangle Dataset Schemes at a glance

The size of the density file in both schemes (2*2*$S_e$*$4^h$ for ID and 2*4*$S_e$*$4^h$ for CD) depends on the number of grid cells ($4^h$). ID uses 2 variables for each cell, while CD requires 4. In addition to selectivity, the density information has to be maintained for the node MBRs for node access estimation, which doubles the size of the density file. The time taken for generating this density file is proportional to the number of grid cells (for the rectangles and for the MBRs of the R-tree nodes). The CD scheme takes at least twice the time taken by the ID scheme since it maintains twice the number of variables with each cell and its variables are cumulative variables.

Estimation is expensive for the ID scheme, since it is proportional to the size of the query window as well as the level of gridding. On the other hand, the CD scheme is independent of these factors, and takes a constant time for selectivity and node access estimation. The CD scheme is also expected to be more accurate than ID because cumulative information can lower estimation errors in some cases as was illustrated earlier.

It is also interesting to note that a point dataset can be viewed as a special class of rectangular data (of size 0). Consequently, *the rectangle dataset schemes can be used to estimate selectivity and nodes accessed of point datasets as well*. We have used the CD scheme for estimation of point datasets in the following evaluation studies, and compare it with the point dataset schemes.

### 2.3.4 Dynamically Updating Density File

While we have discussed the approach for creating the density/histogram information for all the above schemes, we would also like to point out that any updates (inserts/deletes) to the dataset can also be reflected in the density information.

In NDH, where the histogram directly corresponds to the leaf nodes of the R-tree, updating an entry in the leaf node only requires updating the corresponding bucket of the histogram. When the update results in changing the leaves of the index structure, then a similar redistribution has to be performed for the histogram information as well. The ID scheme requires touching only the grid cells that are affected by the update, and is thus fairly efficient as far as dynamic updates are concerned.

On the other hand, schemes such as DH, NDH and CD are a little more expensive since they cumulate the information. One way of alleviating this cost is to not reflect the changes to the density information immediately, but to let them accumulate. At some time when there are a large enough number of updates, to make a significant impact on the accuracy of the estimation, they can be carried out simultaneously to amortize the cost. In fact, this approach can be used for all the schemes to keep dynamic update costs fairly low.

## 2.4 Evaluating the Analysis Techniques

To evaluate the different schemes described in the previous section, we conduct extensive experiments with several point and rectangle datasets, and several query windows. These studies have been conducted on a 170 MHz SUN Ultra Enterprise 1 server (the choice of the machine is immaterial since we are comparing relative performance of schemes on the same system). In the following discussion, we briefly discuss the datasets considered, examine the metrics/criteria used for comparing the schemes, and present the results for specific query windows as well as in the average case. Our schemes are also compared with some of the previously proposed ones.

### 2.4.1 Datasets

We have considered a wide spectrum of point and rectangular datasets, that are either uniformly distributed in space or exhibit some kind of clustering (we use the terms clustered and skewed synonymously in this chapter). Some of them have been obtained from actual/real datasets (such as the Tiger [83] data), while others have been synthetically generated. The reader is referred to [65] for the results on all the datasets, and in this chapter, we present results for the following datasets:

- Four point datasets:
  - **SUP**: Synthetic Uniform Points, containing $240,000$ points;
  - **SCP**: Synthetic Clustered Points, with $240,000$ points clustered around one location,
  - **TOP**: Topological Point dataset taken from [94], with $478,786$ points following a rather regular pattern (points are arranged in regular rows with significant gaps between successive rows),
  - **CFD**: Computation Fluid Dynamics dataset taken from [78], with $208,688$ points that are clustered;
- Four rectangular datasets:
  - **SUR**: Synthetic Uniform Rectangles, with $500,000$ uniformly distributed rectangles,
  - **SCR**: Synthetic Clustered Rectangles, with $240,000$ rectangles that are clustered around the center,
  - **PAR**: a dataset containing the MBRs of rivers of Pennsylvania from the TIGER database [83], with $30,218$ rectangles,

– **CAR**: a dataset containing the MBRs of the streets of California from the TIGER database, with $248,643$ rectangles.

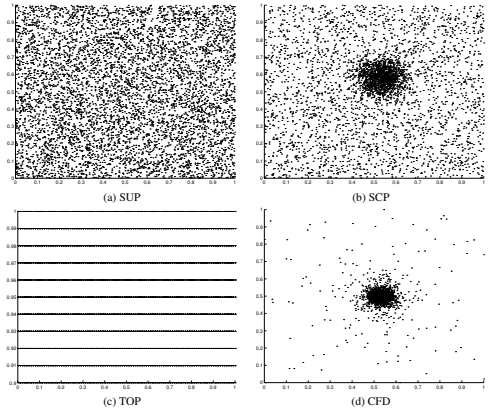



Fig. 2.4. Point Datasets

A pictorial view of these datasets is given in Figures 2.4 and 2.5.

We have considered various query windows which locations are randomly picked with the condition that they cover both sparse and clustered regions of the given spatial dataset. These query windows also have different aspect ratios and sizes(1%, 5%, 25%, 50% and 100%). The reader is referred to [65] for more detailed information about these queries. We believe that the chosen datasets and query windows capture sufficiently diverse workloads with interesting properties to stress the pros and cons of different schemes.

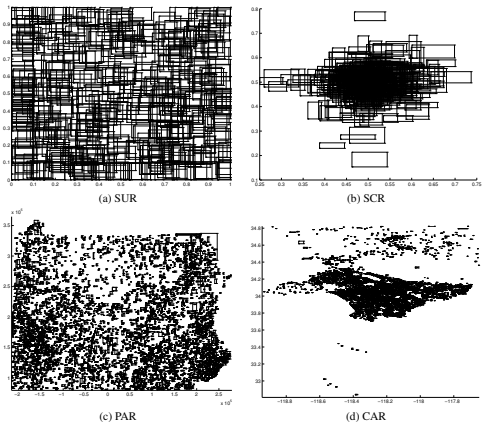



Fig. 2.5. Rectangular Datasets

### 2.4.2 Metrics/Criteria

We have developed a bulk-loaded packed R-tree based on Hilbert order [45, 64] for each of the above datasets, which we use for comparison. We use six criteria for discussing the pros and cons of each scheme:

- *Selectivity Estimation (s)*: This measures the accuracy of the scheme in estimating the number of data items retrieved for the specific query. It is expressed as a percentage error with respect to the number of items retrieved by the query on the actual R-tree.

- *Selectivity Estimation Time (st)*: This measures the time taken by a scheme to estimate selectivity for the specific query. It is expressed as a percentage of the time to execute the query on the actual R-tree.

- *Node Access Estimation (n)*: This measures the accuracy of the scheme in estimating the number of nodes accessed/touched in serving the specific query (which is an indication of

the I/O costs). It is expressed as a percentage error with respect to the number of nodes touched by the query on the actual R-tree.

- *Node Access Estimation Time (nt)*: This measures the time taken by a scheme to estimate the number of nodes accessed/touched by the specific query. It is expressed as a percentage of the time to execute the query on the actual R-tree.

- *Density File Size (d)*: This is a measure of the storage overhead (in bytes) to maintain the density information required by each scheme. It is expressed as a percentage of the storage taken by the actual R-tree.

- *Time for Building Density File (dt)*: This measures the time taken by a scheme to create the density file. It is expressed as a percentage of the time taken to build the actual R-tree.

The reader should note that a relatively small $s$ and $n$ is preferable with low $st$ and $nt$. Though one would like to have a low $dt$ and $d$ as well, it should be noted that $dt$ is a one-time cost, and $d$ may not be a big issue these days with ever increasing disk storage capacities (as long as density files are not larger or become a large fraction of the actual dataset/R-tree).

### 2.4.3 Point Dataset Results

Figure 2.6 shows a part of the results for the different schemes with the point datasets. We present representative results from four query windows for each dataset. We have obtained results for each scheme using different levels/orders ($h$=5,6,7,8,9) for the density file. Instead of presenting all those results, for each workload-scheme combination, we present only those for the lowest level/order (since this will have the lowest $d$ and $dt$) which gives a satisfiable degree (less than 5% error) of accuracy. In case, none of these levels gives an error lower than 5%, then

we give the results for $h$=9. Consequently, different schemes could have different orders/levels ($h$) for a particular workload (a scheme with a higher $h$ usually indicates that it does not do as well as a scheme with a lower $h$). The reader is referred to [65] for actual values if needed.

**Accuracy ($s$ and $n$):** As can be expected, the DH and NDH schemes give better accuracy with higher $h$. This is specifically the case for selectivity estimation since these schemes align the query window to the grid cells. In general, higher $h$ will reduce the change in query window that is needed, giving higher accuracy. There are certain situations where a finer level of gridding can result in a higher percentage change in area of query window compared to a coarser level of gridding (see [65] for some examples). The overall trend, however, indicates that there will be a higher level of accuracy with these schemes with higher $h$. In fact, one could theoretically hypothesize that at very high $h$, these schemes should come very close to the actual R-tree results. But we find that the estimated selectivities and nodes accessed are much lower. After a closer examination, we found that this is due to some points falling directly on cell boundaries. Such points are histogrammed into only one of the buckets to avoid double-counting (see [65]). Consequently, a query which is aligned to that grid cell and does not include the Hilbert value assigned to the point, will not count the point in its estimation. This also explains why the TOP dataset (where several points could lie on grid cell boundaries because of the nature of the dataset) shows unstable results. Despite this, we find that these inaccuracies are not a major problem from our experiments, and we can still get less than 5% error in most cases.

In terms of node access estimation, NDH uses the same technique as the DH scheme, and gives similar results. For selectivity estimation, NDH assumes that data items are uniformly distributed in space within each leaf node (or at least at the nodes which are at the boundaries of

a Hilbert range covered by the query). However, this assumption does not seem to hurt accuracy very much. This is, perhaps, because of two factors. The assumption is made only for the nodes that partially overlap a Hilbert range covered by a query, and this number is relatively low (as a percentage) compared to the total number of selected data items. Further, even clustered datasets, have some semblance of uniformity within small/isolated regions (such as those covered by a single leaf node of the R-tree) and can be approximated accordingly.

The performance of the DHC schemes with three different thresholds (0.6, 1.2 and 1.8) for the coefficient of deviation are given in Figure 2.6. It can be seen that accuracy for thresholds of 1.2 and 1.8, is not acceptable in several cases, and only 0.6 comes close to the other schemes.

As mentioned in the previous section, the rectangle schemes can be used for point datasets as well. In the tables, we have shown the results for the point datasets using CD. We can observe that CD gives similar accuracies as DH for selectivity estimation and higher precision for node access estimation. This is because it uses the actual R-tree information (MBRs of nodes) to estimate this information. So any approximation made in determining selectivities is not carried over to node estimation.

For most combinations of datasets and query windows, the DH scheme estimations were less than 10% error at order/level 6, and less than 5% error at order/level 8 [65]. A similar observations holds for the DHC, NDH and CD schemes as well. Hence, level 8 seems to be an appropriate operating point.

Apart from the nature of the dataset (uniform/clustered) and level of gridding, three other factors have an important effect on the accuracy of these schemes, namely, the location of query window, the size of the query window, and the size of the data set. The query window (0.10,0.61,0.96,0.90) covers a relatively sparse area of the CFD dataset. Consequently, the

number of nodes accessed for this window is quite low. This small value can result in a higher percentage error (even though the deviation from the value may not be much in absolute terms). Similarly, a window located on a dense area, may cause more aberrations when the window is aligned to the nearby grid cell boundaries, resulting in a larger number of data points being included/excluded than actual. A small query window can also give low selectivities and node accesses, and even a minor deviation from the actual number can mean a large percentage error. These factors, together with the impact of dataset size on estimation accuracy have been studied in [65].

**Estimation Time ($nt$ and $st$):** For some of the datasets, the estimation times for the DH and NDH schemes are quite expensive relative to the actual time taken for serving the query. All the benefits of estimating R-tree performance will be lost, if the estimation time is as high as $20 - 50\%$ of the query time itself. Higher the order (for better accuracy), the higher is this time since the query window needs to be broken into several finer Hilbert ranges, and these ranges need to be looked up in the density file. The estimation times for the DHC scheme are even worse, since it needs to look up index information to get to the appropriate densities. However, as the selectivity becomes larger, the estimation time as a percentage of the query time gets smaller, and these schemes may not be as bad in such cases. The CD scheme is clearly a winner for this criteria, because it takes a constant amount of time regardless of the dataset and order/level. As a result, the CD estimation time never exceeds even $1\%$ of the query execution time for any of the workloads.

**Density File Size and Creation Time ($d$ and $dt$):** The theoretical observations in the previous section about the density file size are well borne out by the experimental results. The size

quadruples in the DH scheme as we move to the next order/level. However, for the datasets considered, the size goes only as high as 10% of the space occupied by the actual R-tree (and that too in only a few cases). Given that space is not really a severe problem (as important as time), the DH scheme may not be a bad choice. Moving to the DHC scheme, we observe that the savings due to compression is not very significant. At low thresholds, there are not many nearby grid cells to merge, and the index that is necessitated for this file can offset any gains due to compression. There is some saving in DHC for certain clustered datasets (like CFD), where there are regions in space with little or no points that can be merged. The CD scheme requires nearly 8 times the size of the DH scheme. This is because there are 4 variables stored for each grid cell (DH requires only one), and we need to maintain the information for not only the data points, but also for the node MBRs. In terms of $d$, the NDH scheme is the winner. The space that it requires is directly proportional to the number of R-tree nodes, and this is much smaller than the actual space in bytes taken by the R-tree (much less than 1%). Further, this size is independent of the order/level of gridding.

The DH, DHC and NDH schemes require the Hilbert values be assigned to all the data points, and then externally sorted before they are histogrammed. DHC requires additional time for compression, which seems to be significantly higher from the results. The CD scheme does not require Hilbert values to be generated or sorted, but it requires additional time to calculate the cumulative information. These two factors more or less compensate each other, and the density file creation for CD takes roughly the same time as DH/NDH.

### 2.4.4 Rectangular Dataset Results

As in the point dataset results, Figure 2.7 shows the results of the ID and CD schemes on the four rectangular datasets with different query windows.

The ID scheme has significant errors for selectivity and node estimation even at level 9. In fact, we need to go higher than level 12 to get errors within 5%. This is because of approximation errors that were explained earlier using Figure 2.3. The estimation times are very high as well, and in many cases even exceed the actual query execution times on the R-tree. These results suggest that ID is not a feasible approach, and we do not discuss it further.

On the other hand, the CD scheme appears to give remarkably accurate results for both selectivity and node access estimation. We need to go only up to levels 8 or 9 to get the errors within 5%. It achieves this goal with a constant estimation time that is usually much less than 1% of the actual query execution time. The reader should note that the storage taken by the density file for CD (for DH and ID as well) is independent of the dataset size. It is purely a function of the level/order. So the storage (in bytes) taken by the density file at level 8 for CD in the PAR dataset is the same as the storage taken at level 8 in the SUR dataset. The reason why the percentages are quite high for PAR is because the dataset is itself quite small (the corresponding R-tree is smaller) compared to CAR. The same argument holds for the density file creation time. In summary, as the dataset size grows larger, the space overhead of the CD scheme gets smaller (as a percentage).

### 2.4.5 Comparison with Previous Research

In summary, the point dataset results clearly show that CD (and DH to a certain extent) does the best, giving fairly accurate results (less than 5% error for CD) in a short time (in CD it

takes much less than 1% of the query execution time for estimation). CD is the clear winner in the case of the rectangle datasets. For the datasets that were considered, we need to go only up to levels 8 or 9 for these schemes, and the density storage overheads are not overly demanding at these levels. As the datasets get larger (which is when analysis is really meant to be useful), the storage overhead as a percentage of the dataset size becomes smaller.

We have also compared node access estimation accuracy ($n$) of CD with three previously proposed analysis techniques for point datasets and rectangular datasets in Figures 2.8 and 2.9. The reader is referred to [65] for actual values and to [65] for details on the formulae/algorithms used in estimation for the previously proposed techniques. We have presented the comparisons over several query windows (the average case results, which have been used in previous studies, are presented in the next subsection) to bring out the quirks of the different techniques.

The [68] scheme gives reasonable accuracy for uniform point datasets. Since it does not consider the location of the query window, it gives higher errors (some even larger than 100%) for some windows on clustered datasets. The [32] scheme gives a little better accuracy for uniform datasets compared to [68]. But it is equally worse for the clustered datasets. If the query windows on such datasets were to be focussed on the clustered area (as an application may demand), these schemes would give gross approximations. The MinSkew scheme [1] is not very accurate for the SUP dataset, but is better than [68, 32] for the other point datasets.

For the rectangle datasets, the [68] and [96] schemes give similar accuracies. The [117] scheme gives a little better accuracy, coming closer to the actual R-tree results. This is because in our implementation of this scheme, we actually go through the dataset to find the density of the region covered by the query window (though this is not really practical) as explained in

[65] instead of assuming uniform densities. MinSkew does better than [68] for the clustered rectangular datasets (SCR and CAR) as expected.

The CD scheme gives the best accuracies in most cases. Our schemes take the query window size, location and dataset distribution into consideration in making the estimations. They neither assume any particular distribution (uniform, clustered, clustered following a certain rule etc.) for the dataset, nor do they make any assumptions about the size and location of query window.

### 2.4.6 Average Case Accuracy

One of the points we are trying to make is that it is important to examine individual estimation errors with a spectrum of different query windows as we have done in the previous experiments. Averaging the errors over a large number of windows as many previous studies have done [1, 68, 117, 96], could hide some of the gross inaccuracies for certain specific query windows (and these windows could be important for a particular application). To address any concerns that the reader may have with this approach (and any concerns that the reader may have about the importance/representativeness of the query windows considered until now), we have also run numerous query windows (1000) over the datasets, and present the average (mean) estimation error for the CD (level 9), KF93 [68] and MinSkew schemes in Table 2.4, together with the minimum, maximum and standard deviation of these errors. The reader should note that the maximum error captures the results for just one of the query windows, and this is usually for a window which has very low selectivity. A very low selectivity, can result in large percentage errors even if the absolute number is not significantly different from the actual value (for instance, an estimate of 2 for a selectivity of 1 will give 100% error). When many query windows have

low selectivities/node accesses in the workload, the corresponding mean errors are high as well. If $E_i$ is the estimation for the actual value $X_i$, then the mean error percentage ($\bar{d}$) is calculated as $\bar{d} = \frac{\sum_{i=1}^{N} R_i}{N}$ and the standard deviation of error percentage ($\sigma_d$) is calculated as $\sigma_d = \sqrt{\frac{\sum_{i=1}^{N} (R_i - \bar{d})^2}{N}}$ where $R_i = \frac{|E_i - X_i|}{X_i} \times 100$. Those with a $X_i = 0$ are excluded from these calculations.

For the query windows, we use two workloads (*uni* and *skew*). In both these workloads, the aspect ratio of the query window is uniformly varied between 0.33 to 3.0, and the size is varied between 0.1% to 25% of the spatial extent. In *uni*, the center point of the query window is uniformly distributed within the spatial extent.

Our *skew* workload is based on the hypothesis that the region with more data will draw more queries. We first obtain a spatial Cumulative Distribution Function (CDF) of the points (in point datasets) or center-points (in rectangle datasets) of the target dataset. The workload picks the center points of the query windows from the probability density function determined by this CDF.

Many previous studies have used workloads similar to *uni* to conduct average case experiments, and we feel that *skew* may be a better approximation to actual workloads (we can only hypothesize at this point, and we plan to investigate workload characterization in our future work). Regardless of the workload used for average case estimation, we still find that CD gives very good estimates (typically less than 5% error) over all the datasets as in the previous individual query estimations. Its results are significantly more accurate than the corresponding estimations by KF93 [68] and MinSkew in most cases.

| Workload | Scheme | Selectivity | | | | Node Access | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | Min(%) | Max(%) | $\vec{d}$ | $\sigma_d$(%) | Min(%) | Max(%) | $\vec{d}$ | $\sigma_d$ |
| TOP - Uni | CD | 0 | 72.73 | **2.34** | 8.74 | 0 | 44.85 | **1.70** | 6.63 |
| | KF93 | 0 | 86.10 | **11.28** | 11.87 | 0.04 | 135.68 | **12.79** | 12.19 |
| | MinSkew | 0 | 73.40 | **1.25** | 4.96 | 0 | 100 | **25.65** | 16.56 |
| TOP - Skew | CD | 0 | 158.70 | **5.52** | 17.87 | 0 | 65.58 | **3.61** | 10.69 |
| | KF93 | 0 | 118.87 | **19.86** | 22.28 | 0.1 | 84.82 | **15.49** | 11.68 |
| | MinSkew | 0 | 219.21 | **1.70** | 8.25 | 0 | 84.93 | **19.36** | 15.11 |
| CFD - Uni | CD | 0 | 54.55 | **1.93** | 3.16 | 0 | 31.28 | **1.09** | 3.68 |
| | KF93 | 0.15 | 13467.63 | **3230** | 2596.01 | 1.08 | 2754.6 | **697.92** | 594.86 |
| | MinSkew | 0.34 | 70400 | **2918.67** | 5394.05 | 0 | 3030.76 | **399.75** | 613.20 |
| CFD - Skew | CD | 0 | 14.58 | **0.12** | 0.69 | 0 | 13.58 | **0.13** | 0.77 |
| | KF93 | 57.99 | 7553.67 | **138.47** | 474.69 | 30.39 | 1601.29 | **95.12** | 96.61 |
| | MinSkew | 8.97 | 20702.08 | **143.93** | 1006.22 | 0 | 1888.24 | **52.05** | 118.47 |
| PAR - Uni | CD | 0 | 53.85 | **1.23** | 2.56 | 0 | 35.71 | **1.05** | 2.53 |
| | KF93 | 0.47 | 32160.24 | **420.51** | 1789.09 | 0.06 | 6153.75 | **251.62** | 441 |
| | MinSkew | 0 | 2784.62 | **36.19** | 159.04 | 0 | 350 | **37.90** | 22.09 |
| PAR - Skew | CD | 0 | 25.10 | **0.64** | 1.14 | 0 | 20.00 | **0.95** | 1.71 |
| | KF93 | 0.07 | 272.60 | **36.70** | 22.82 | 0.25 | 233.66 | **33.22** | 19.37 |
| | MinSkew | 0 | 118 | **9.25** | 10.45 | 0 | 100 | **46.16** | 18.33 |
| CAR - Uni | CD | 0 | 50.00 | **1.50** | 3.24 | 0 | 100.00 | **1.24** | 4.08 |
| | KF93 | 0.1 | 2616680 | **9839.30** | 117907 | 0.09 | 27059 | **883.42** | 2396.45 |
| | MinSkew | 0 | 85685.71 | **271.21** | 2939.75 | 0 | 1000 | **109.52** | 126.12 |
| CAR - Skew | CD | 0 | 8.28 | **0.71** | 0.85 | 0 | 21.05 | **0.92** | 1.59 |
| | KF93 | 0 | 305 | **69.51** | 22.11 | 0.1 | 225.08 | **66.75** | 18.75 |
| | MinSkew | 0.07 | 367.60 | **37.26** | 26.24 | 0 | 378.13 | **37.51** | 46.86 |

Table 2.4. Estimation Errors in the Average Case for CD, [KF93] and MinSkew

### 2.4.7 What Level/Order should we use?

There is an issue with regard to the proposed schemes that has not yet been investigated, and which is important to all histogram/density based techniques. This is regarding how we can decide on the level/order for the density file (the query window parameters are not known at the time the density file is created). A larger level/order usually improves the accuracy of estimation, albeit at a higher storage cost. Previous histogram-based studies [1] have used an experimental approach with average case studies of different data sets to find a good operating point. We use a similar approach, and the average errors used with numerous (1000) query windows randomly located at different regions of the spatial extent for each of three different sizes (0.1%, 1% and 10% of the spatial extent) is given in Figure 2.10 for the CD scheme on four real datasets. As expected, the average error decreases with increasing levels, and the percentage error is lower with larger query windows. Though the inferences that can be drawn are dataset dependent, we find that over all the datasets that we have considered (including the ones presented in [65], the errors become reasonably small (typically lower than 5-10%) beyond levels 9 or 10. Knowledge of the dataset and/or possible query windows for the given dataset can help decide on a level in a better fashion than using a workload with average case behavior. It should again be noted that the estimation time for the CD scheme is relatively independent of the level of gridding (unlike other schemes [1]), allowing us to use a reasonably large gridding level as long as storage for the density file does not become a big issue.

## 2.5 Chapter Summary

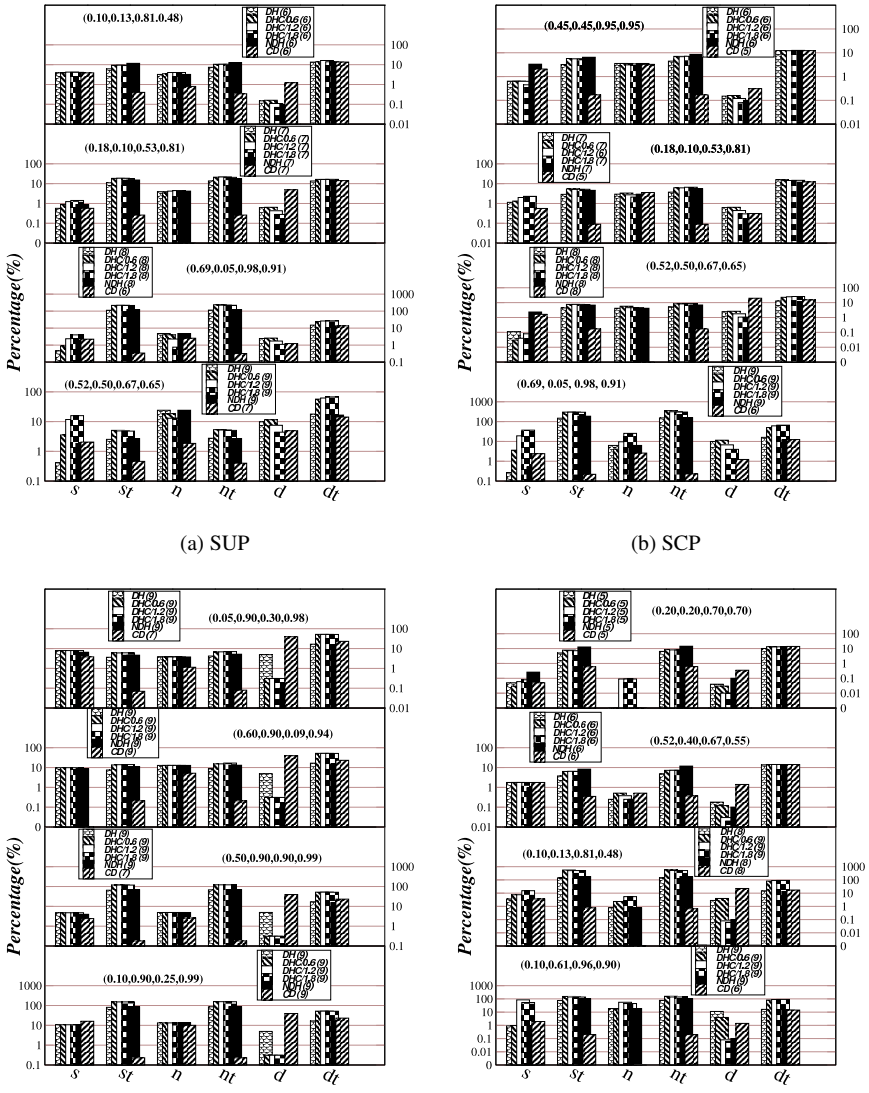
This chapter has presented a novel set of schemes to analyze spatial selection performance on spatial data. Three of these schemes can analyze point datasets, and the other two can be used for both point and rectangular datasets. These schemes make very little assumptions about the dataset, and use an auxiliary data structure (histograms) called a density file which can be constructed when the index structure is created (one-time cost). When a query is given, the density file is "quickly" looked up to get sufficient information about the dataset which is then used to calculate the selectivity. We have also illustrated how this information can be used to estimate the number of nodes that would be accessed in the index structure using the packed R-tree as a case-study.

With a diverse suite of real and synthetic datasets, which fall under both uniform and skewed classifications, this chapter has shown that one of the schemes, called *Cumulative Density* (CD), gives very accurate results, with errors that are much lower (usually less than 5% errors) than many previously proposed analysis techniques. This accuracy is obtained over a spectrum of query window sizes, locations and aspect ratios, and not just in the average case. This makes the CD scheme more universally applicable. The estimation with the CD scheme takes a constant amount of time (at most 4 disk accesses and 3 arithmetic operations), regardless of dataset or query window parameters. This time tends to be typically lower than 1% of the time that it would take to actually execute the query using an R-tree. As a result, the CD scheme is very practical and would be extremely useful in a query optimizer.

The CD scheme can be used to estimate the selectivity of both point and rectangular datasets. It can be used for other spatial objects, provided their bounding boxes are available (in

which case it can be used to measure the cost of a filtering step). In addition, the CD scheme can be used to estimate the number of nodes accessed by a spatial selection in any index structure (not just an R-tree), provided the bounding boxes of the nodes of the structure are made available. We can use these bounding boxes as input to the selectivity procedure of CD to find out the number of nodes that would be accessed in the index structure.

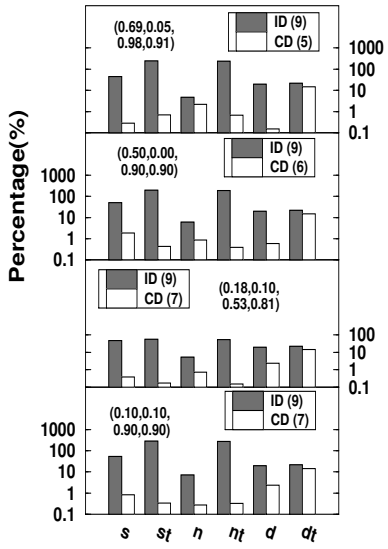
This chapter has opened interesting directions for future research. While the techniques presented here could be used for multidimensional datasets, there is a concern of the space required to maintain the histograms. However, the CD technique is still promising since its estimation costs remain constant. We plan to characterize the workload (data sets, query window parameters etc.) of different spatial databases for a better choice of the gridding level, which could also help us address the storage issue. We are trying to find out if the proposed schemes can be extended (or new ones can be developed along the lines of what has been presented here) to more complicated queries, such as spatial joins. Finally we would like to incorporate these schemes into a spatial database query optimizer.



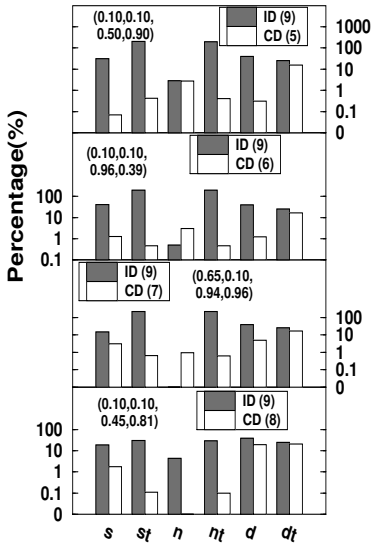

(a) SUP (b) SCP

(c) TOP (d) CFD

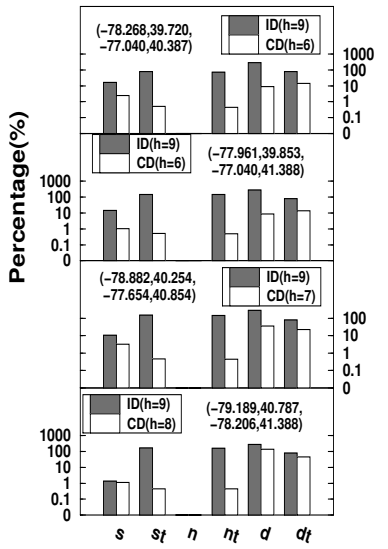
Fig. 2.6. Point Datasets



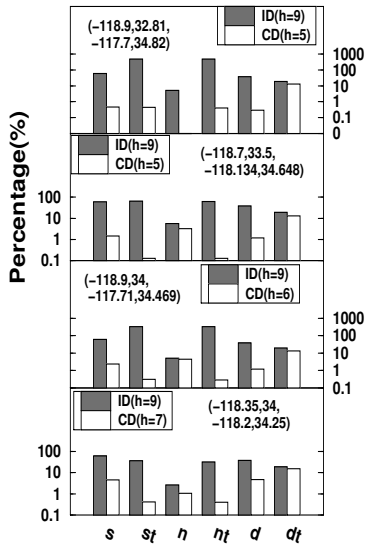

SUR (universe=[0,0,1,1])




SCR (universe=[0,0,1,1])

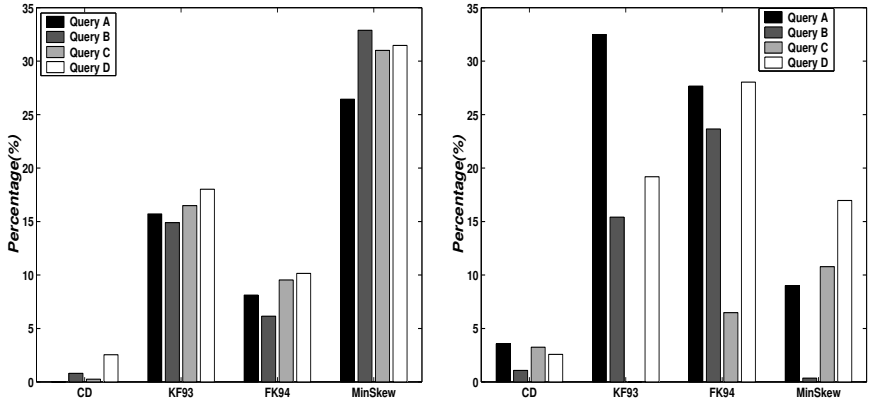



PAR (universe=[-79.901,39.7198, -76.8602,41.6181])




CAR (universe=[-118.9440,32.8062, -117.6480,34.8232])

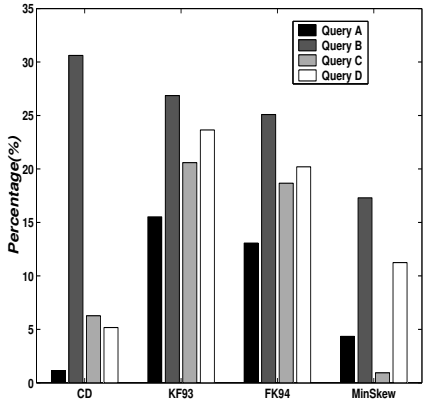
Fig. 2.7. Rectangle Datasets



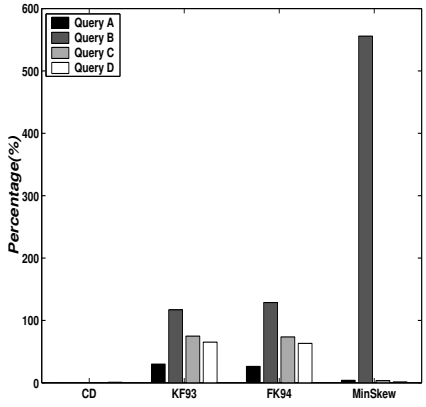

(a) SUP ( universe=[0,0,1,1] )
QueryA = [0.18,0.10,0.53,0.81]
QueryB = [0.10,0.13,0.81,0.48]
QueryC = [0.45,0.45,0.95,0.95]
QueryD = [0.69,0.05,0.98,0.91]




(b) SCP ( universe=[0,0,1,1] )
QueryA = [0.18,0.10,0.53,0.81]
QueryB = [0.10,0.13,0.81,0.48]
QueryC = [0.45,0.45,0.95,0.95]
QueryD = [0.69,0.05,0.98,0.91]

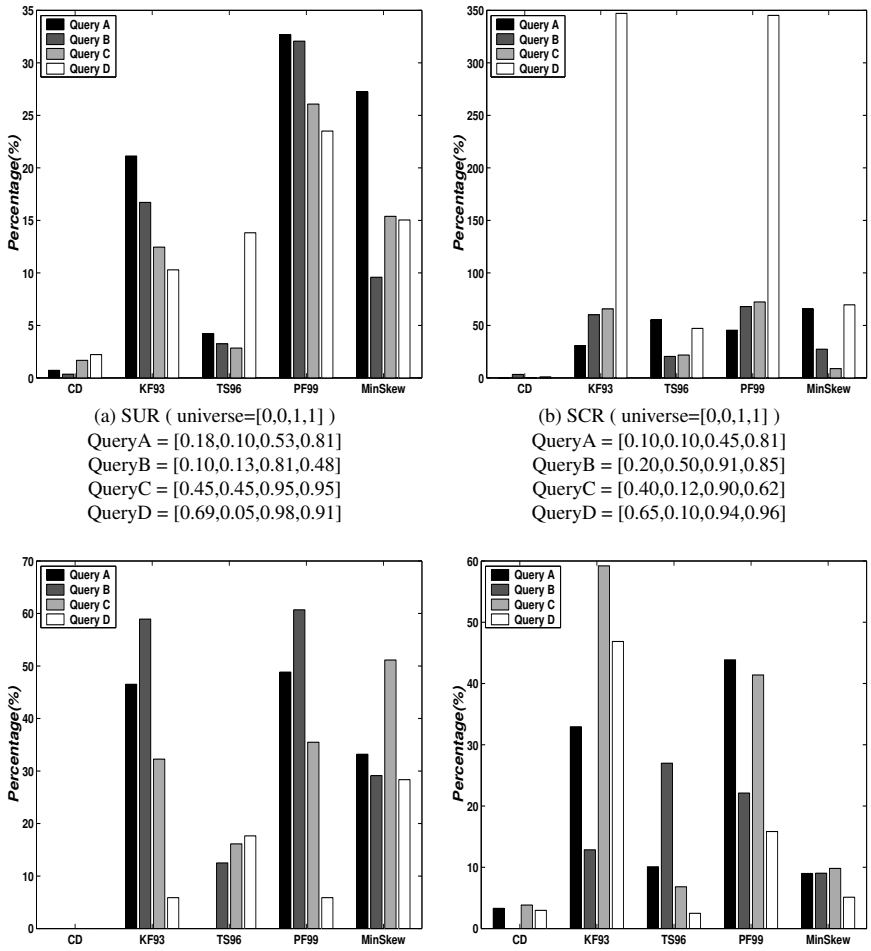



(c) TOP ( universe=[0,0.9,1,1] )
QueryA = [0.05,0.90,0.30,0.98]
QueryB = [0.10,0.96,0.90,0.99]
QueryC = [0.20,0.92,0.60,0.98]
QueryD = [0.60,0.90,0.90,0.94]




(d) CFD ( universe=[0,0,1,1] )
QueryA = [0.18,0.10,0.53,0.81]
QueryB = [0.10,0.13,0.81,0.48]
QueryC = [0.20,0.20,0.70,0.70]
QueryD = [0.52,0.10,0.81,0.96]

Fig. 2.8. Comparing Accuracy of Node Access Estimation for Point Databases with Previous Research




(a) SUR ( universe=[0,0,1,1] )
QueryA = [0.18,0.10,0.53,0.81]
QueryB = [0.10,0.13,0.81,0.48]
QueryC = [0.45,0.45,0.95,0.95]
QueryD = [0.69,0.05,0.98,0.91]

(b) SCR ( universe=[0,0,1,1] )
QueryA = [0.10,0.10,0.45,0.81]
QueryB = [0.20,0.50,0.91,0.85]
QueryC = [0.40,0.12,0.90,0.62]
QueryD = [0.65,0.10,0.94,0.96]

(c) PAR ( universe=[-79.901, 39.7198, -76.8602, 41.6181] )
QueryA = [-79.901,39.7198, -79.189,40.854]
QueryB = [-78.268,39.7198,-77.040,40.387]
QueryC = [-78.882,40.254,-77.654,40.854]
QueryD = [-79.189,40.788,-78.206,41.388]

(d) CAR ( universe=[-118.9440,32.8062, -117.6480,34.8232] )
QueryA = [-118.7,33.5,-118.134,34.648]
QueryB = [-118.8,33.4,-117.652,33.966]
QueryC = [-118.7,34,-117.891,34.809]
QueryD = [-118.2,33.5,-117.731,34.82]

Fig. 2.9. Comparing Accuracy of Node Access Estimation for Rectangle Databases with Previous Research

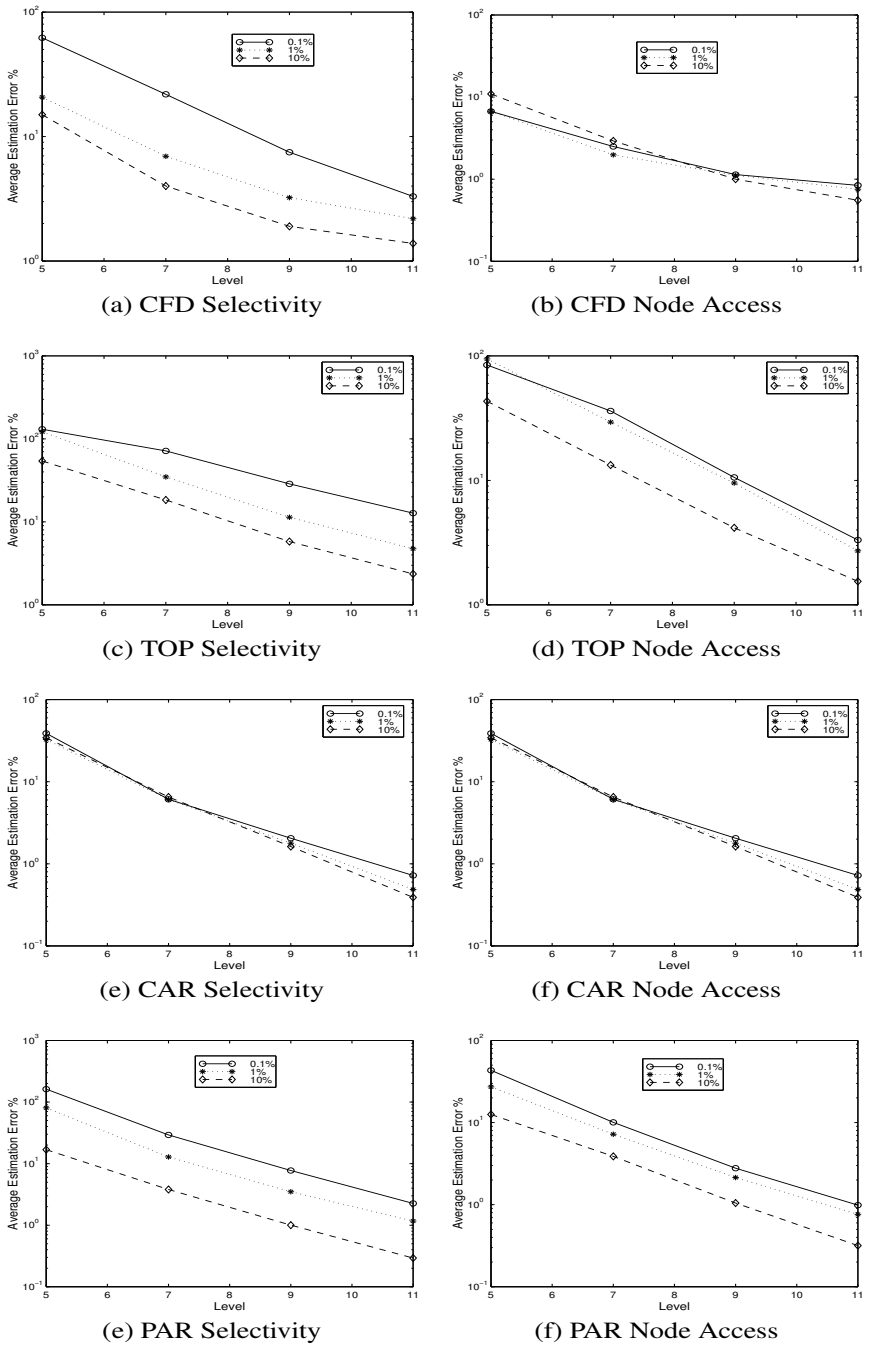



Fig. 2.10. Impact of Gridding Level on Accuracy of Estimation using CD. The average error is shown for three query window sizes (0.1%, 1% and 10% of spatial extent)

# Chapter 3

# Selectivity Estimation for Spatial Joins

## 3.1 Introduction

Spatial Database Management Systems (SDBMS) [108] have found widespread adoption in numerous domains including Geographical Information Systems (GIS), Image Processing, Military Planning and Logistics, Computer Aided Design (CAD), Multimedia Systems, and Medical Database Systems. These applications require mechanisms for efficient storage, retrieval and processing of spatial data. To meet these goals, a SDBMS needs to provide a range of specialized and optimized spatial operations, such as spatial selection, nearest neighbor query and spatial join. Of these operations, spatial joins are particularly important because they are not only important queries in their own right, but can also serve as building blocks for more complex spatial predicates. Spatial joins also present interesting challenges because of their high CPU and I/O costs.

A spatial join is commonly used to answer spatial queries involving more than one dataset. The intention is to find pairs of objects (one from each dataset) that meet a given spatial predicate, such as intersection/overlap, containment, etc. For example, the query "find all the major highways in Pennsylvania that cross a major river" can be answered by performing a spatial join on the highway and river datasets of Pennsylvania. In SDBMS, a spatial data object is typically abstracted/represented by its Minimum Bounding Rectangle (MBR), which is the smallest axis-parallel rectangle that fully contains this spatial object. Using MBRs, spatial joins

are performed in two steps [85]: the filter step and the refinement step. The filter step retrieves all Minimum Bounding Rectangles (MBRs) that satisfy the given spatial predicate. The refinement step then examines the exact geometry of the pairs produced by the filter step to discard any false hits. Although the refinement step is an important issue, most prior research (as is this chapter) has focused on the filter step. This includes techniques [22, 57] to perform spatial joins using R-trees [48], techniques [75, 77] to perform the join when only one of the datasets is indexed by an R-tree, and techniques [91, 14] when no index is available for either input dataset.

In all, a good deal of research has been done on optimizing spatial join processing. However, there is another important problem related to spatial joins: *How do we predict the performance (selectivity in particular) of spatial joins?* The spatial join selectivity of two datasets is the ratio of the resultant size of the spatial join to the size of the Cartesian product of both participants. As most prior research, this work considers only the filter step of the spatial join, and we thus deal only with two sets of axis-parallel rectangles (in a 2-D space, though the arguments automatically extend to higher dimensions). The spatial predicate for the join in this chapter extracts pairs of intersecting MBRs from the two datasets. Even with this simplication, accurately estimating the spatial join selectivity poses problems because (a) data items are located in a multidimensional space (instead of a single dimension in the traditional RDBMS), and (b) size of the spatial objects can vary significantly.

Being able to get a good selectivity estimation for spatial joins can help optimize the join implementation itself. It can help select the best index structure, and can also help fine-tune the structure within the purview of its definition. More importantly, selectivity estimation is crucial in a query optimizer for choosing a good execution plan for a given query. Selectivity estimates of spatial joins can themselves be used as responses to specialized user queries that are

seeking approximate figures. For instance, finding the approximate number of bridges in a given spatial extent may simply be satisfied by doing a join selectivity estimation between the streets and rivers datasets for that extent (and may not necessitate performing the actual join). Finally, spatial join selectivity can also be used for evaluating the correlation between datasets [34].

The utility of selectivity estimation for spatial operations is widely recognized [108]. While there have been a large number of forays into this topic in the context of range queries [35, 116, 32, 16, 96, 117, 119, 87, 68, 66], the problem of selectivity estimation for spatial joins has been little explored. There are two prior studies, [56] and [118], that have extended prior analytical models for range query costs, to estimate the I/O performance of joins using R-trees. To our knowledge, there have been very few attempts [13, 17, 34] at selectivity estimation for spatial joins. Taking one dataset as the source of query windows, and the other as the underlying data, [13] simply applies the technique proposed in [68] for range query estimation, and sums these results to get a convenient closed form formula. Alternatively, [17] uses fractal concepts to estimate the selectivity of spatial self-join for point datasets. Along the same line, [34] uses a power law to model the distribution of pair-wise distance between two real multidimensional point datasets. Using this law, a fairly accurate selectivity estimation is derived for the spatial join of two point datasets.

Selectivity estimation techniques can be broadly categorized into three classes: parametric, sampling and histograms. Parametric techniques typically make some assumptions about the dataset to present convenient closed-form formulae for estimation, at little cost. For instance, [13] assumes that the data items are uniformly distributed in the two datasets to be joined, while

[17] and [34] assume that the data items exhibit fractal behavior or obey a power law respectively. However, these assumptions restrict their applicability since real datasets may not necessarily adhere to such properties. Further, [17] and [34] can work only with point datasets. The other two classes of estimation techniques, sampling and histogram-based, try to draw sufficient information from the given dataset to predict query selectivity. As a result, they are applicable to a larger class of datasets than their parametric counterparts. Sampling techniques actually perform the query on a much smaller version of the dataset, called the sample (which is supposed to capture/represent the behavior of the entire dataset), and use the results to project the selectivity on the entire dataset. The space and time overheads are lower with a smaller sample, albeit at a potential loss in accuracy (in fact, it is difficult to say whether a larger sample would necessarily give higher accuracy making these techniques somewhat unstable). The difficulty in picking a representative sample with low overheads makes sampling somewhat undesirable. Histogram-based techniques, on the other hand, keep certain information for different regions of the spatial extent in an auxiliary data structure (histograms), and quickly consult this structure to find the selectivity when the query is given. The trick with histograms is in finding out what information to maintain and at what granularity, so that duplication across buckets of the histogram or the lack of information within each bucket does not significantly impact accuracy.

This chapter intends to fill a crucial void in selectivity estimation of spatial joins by proposing and evaluating different sampling and histogram based techniques [11]. While sampling techniques [49, 50, 51, 15] have been used in estimation for conventional databases, their applicability has not been studied for spatial joins. In particular, this chapter studies three different sampling techniques based on the way the samples are picked. In addition, two novel histogram based techniques are proposed. Using a diverse spectrum of real and synthetic datasets,

that exhibit wide spatial distributions/patterns, these techniques are examined in terms of the estimation error and the estimation costs (both time and space), compared to performing the actual join.

It is shown that in most cases, picking samples randomly, with a sample size of 5-10% of the dataset, gives less than 10% errors at a overhead that is around 10% of the join time when the R-trees for the two datasets are not available. However, this is not a worthwhile option if the R-trees are available since the join itself is not as expensive. It could then be argued that one could also have the sample picked beforehand (just as the R-tree is constructed beforehand), in which case sampling is again a possible option. One of the undesirable properties of sampling is that the results are unstable i.e. it is highly dataset and sample dependent, and it is difficult to draw concrete conclusions.

On the other hand, one of the histogram based techniques that we propose in this chapter, called the Geometric Histogram (GH) scheme, is shown to bring errors down to less than 5% with little overheads. This scheme uses extensive adjustments within and across buckets to avoid multiple and/or false counting of pairs in the join estimation. It is shown that both of our proposed histogram schemes can give much more accurate (and stable) results than the only known prior parametric technique for join selectivity estimation that has been discussed in [13].

The rest of this chapter proceeds as follows. Sections 3.2 and 3.3 present the sampling and histogram based techniques for estimating spatial join selectivity. These techniques are then experimentally compared in section 3.4 using a wide range of datasets. Finally, Section 3.5 summarizes the contributions of this chapter, and offers suggestions for future work.

## 3.2 Sampling Techniques

While sampling techniques have been used [49, 49, 50, 51, 15] to estimate the selectivity of equi-join, which is the counterpart of the spatial join in the relational DBMS, there has been no prior investigation, to our knowledge, of the applicability of these techniques to spatial data. In this study, we pick samples from both input datasets to be joined, and an R-tree [48] is then constructed for each of these samples. While one could try to directly perform a plane sweep algorithm [95] on the two samples, we have found that constructing an R-tree for the samples, then performing an R-tree join [22] is a better alternative, since even a small percentage of the datasets (which can be large) can result in a large number of data items to be joined. Suppose the sample sizes are $\alpha\%$ and $\beta\%$ of the the original datasets respectively, the estimated join selectivity is given by $\frac{R}{\alpha\% \times \beta\%}$, where $R$ is the selectivity of the join on the samples. We consider the following three techniques to pick samples from the two datasets:

1. Regular Sampling (RS): If the sample size is $n$ and the dataset size is $N$, RS generates a sample by taking every $k$th data item ($k = \lceil \frac{N}{n} \rceil$).
2. Random Sampling With Replacement (RSWR): Every data item of the given dataset has an equal probability of being selected, with a chosen data item potentially being picked more than once.
3. Sorted Sampling (SS): This follows the same procedure as RS, except that the input dataset is first sorted based on the Hilbert values [33] of the data items.

## 3.3 Histogram Based Techniques

The following subsections present two histogram based techniques to estimate spatial join selectivity. The common theme between these techniques is that an auxiliary data structure, histogram file, is constructed from the original dataset beforehand. The spatial extent is first gridded into equi-sized cells with a number of vertical ($2^h$) and horizontal ($2^h$) lines, where $h$ denotes the level of gridding. The histogram file stores the necessary information for each of the resulting $4^h$ cells. Later, when estimating a spatial join selectivity, these files for the two datasets (to be joined) are consulted. The following techniques differ in what information is kept in each cell.

### 3.3.1 Parametric Histogram (PH) Scheme

In this subsection, we first describe one prior parametric scheme [13], and see how it estimates the spatial join selectivity. A simple and straightforward extension is then proposed to overcome its shortcoming.

#### 3.3.1.1 Prior Approach

Assuming that both range queries and data are uniformly distributed over the entire spatial extent, Kamel and Faloutsos [68] developed an analytical formula to evaluate the average response time for a range query. This was later extended to estimate the selectivity of spatial joins [13]. The basic idea is to consider one data set as the underlying database and the other as a source for query windows. The sum of the estimated range query selectivities would then give an estimation of the spatial join selectivity.

Suppose we have the following parameters for dataset $DS_k$:

- $A$: the area of the entire given spatial extent.
- $N_k$: the number of all data items in the dataset $DS_k$ .
- $C_k$: the data coverage, i.e. the ratio of the sum of the areas of each data item in the dataset $DS_k$ to $A$.
- $W_k$: the average width of all data items in the dataset $DS_k$.
- $H_k$: the average height of all data items in the dataset $DS_k$.

Then, the selectivity of the spatial join between datasets $DS_1$ and $DS_2$ is estimated in [13] as:

$$Size_{1\_2} = N_1 \times C_2 + C_1 \times N_2 + N_1 \times N_2 \times \frac{W_1 \times H_2 + W_2 \times H_1}{A} \quad (3.1)$$

$$Selectivity_{1\_2} = \frac{Size_{1\_2}}{N_1 \times N_2} \quad (3.2)$$

#### 3.3.1.2 Proposed Extension (PH)

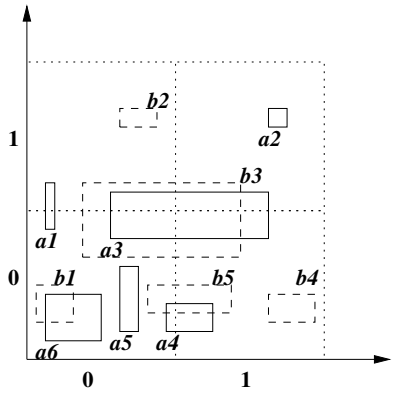



Fig. 3.1. Extending PH

While the parametric technique discussed in [13] incurs negligible time and space overheads (only requires computing Equation 3.1), the underlying assumption is that the data items are uniformly distributed in the spatial extent. Deviations from this assumption can result in significant errors in estimation as will be shown later in this chapter. One way of fixing this problem is to grid the spatial extent into cells, with the hope that the uniformity assumption holds better within each cell. This leads us to propose a technique called Parametric Histogram (PH), wherein we maintain the necessary information (the parameters in Equation 3.1), for each grid cell in the histogram file. Selectivity estimation is then a sum of the

selectivity over all the grid cells. While PH may appear straightforward, it has a drawback of multiple counting the intersections. For instance, in Figure3.1, MBRs $a3$ and $b3$ that span more than one cell actually intersect only once, but could be counted four times (in the four cells). Finer the gridding level (to better approximate uniformity within a cell), worse is the multiple counting problem. To alleviate this problem, our proposed PH scheme categories the MBRs from dataset $DS_k$ that intersect with a cell(i,j) into two groups: $Cont_k(i,j)$ which contains MBRs that are fully contained within cell$(i,j)$; and $Isect_k(i,j)$ which contains MBRs that intersect with cell$(i,j)$, but are not fully contained within it. For dataset $a$ in Figure 3.1, $Cont_a(0,0)$ includes MBRs $a5$ and $a6$ while $Isect_a(0,0)$ includes MBRs $a1$, $a3$ and $a4$.

For given datasets $DS_1$ and $DS_2$, the selectivity estimation for each cell$(i,j)$ now needs to handle four cases: (a) intersection of $Cont_1$ and $Cont_2$; (b) intersection of $Cont_1$ and $Isect_2$; (c) intersection of $Isect_1$ and $Cont_2$; and (d) intersection of $Isect_1$ and $Isect_2$.

| Parameters | Description |
|---|---|
| $AvgSpan_k$ | average number of cells spanned by MBRs spanning cell boundaries |
| $Area_{cell}$ | area of a cell. |
| $Num_k(i,j)$ | number of MBRs that are fully contained in this cell (i.e. $Cont_k(i,j)$). |
| $Cov_k(i,j)$ | ratio of the sum of areas of MBRs in $Cont_k(i,j)$ to $Area_{cell}$. |
| $Xavg_k$ | average width of MBRs in $Cont_k(i,j)$. |
| $Yavg_k$ | average height of MBRs in $Cont_k(i,j)$. |
| $Num'_k(i,j)$ | number of MBRs that intersect this cell and cross cell boundaries (i.e. $Isect_k(i,j)$). |
| $Cov'_k(i,j)$ | ratio of the sum of intersecting areas of MBRs in $Isect_k(i,j)$ with cell$(i,j)$, to $Area_{cell}$. |
| $Xavg'_k(i,j)$ | average width of **intersections** of MBRs in $Isect_k(i,j)$ with cell$(i,j)$. |
| $Yavg'_k(i,j)$ | average height of **intersections** of MBRs in $Isect_k(i,j)$ with cell$(i,j)$. |

Table 3.1. PH Parameters

Table 3.3.1.2 summarizes the parameters that are used to implement the PH technique for a given dataset $DS_k$. Note that except for the first two (which are for the entire dataset), the

other parameters are maintained for each cell. The estimation for the above four cases ($S_a$, $S_b$, $S_c$, $S_d$) can then be calculated using these parameters as follows (directly drawn from Equation 3.1):

$$\begin{aligned} S_a(i,j) &= Num_1(i,j) \times Cov_2(i,j) + Cov_1(i,j) \times Num_2(i,j) + Num_1(i,j) \times \\ & \quad Num_2(i,j) \times \frac{Xavg_1(i,j) \times Yavg_2(i,j) + Yavg_1(i,j) \times Xavg_2(i,j)}{Area_{cell}} \quad (3.3) \\ S_b(i,j) &= Num_1(i,j) \times Cov_2'(i,j) + Cov_1(i,j) \times Num_2'(i,j) + Num_1(i,j) \times \\ & \quad Num_2'(i,j) \times \frac{Xavg_1(i,j) \times Yavg_2'(i,j) + Yavg_1(i,j) \times Xavg_2'(i,j)}{Area_{cell}} \quad (3.4) \\ S_c(i,j) &= Num_1'(i,j) \times Cov_2(i,j) + Cov_1'(i,j) \times Num_2(i,j) + Num_1'(i,j) \times \\ & \quad Num_2(i,j) \times \frac{Xavg_1'(i,j) \times Yavg_2(i,j) + Yavg_1'(i,j) \times Xavg_2(i,j)}{Area_{cell}} \quad (3.5) \\ S_d(i,j) &= Num_1'(i,j) \times Cov_2'(i,j) + Cov_1'(i,j) \times Num_2'(i,j) + Num_1'(i,j) \times \\ & \quad Num_2'(i,j) \times \frac{Xavg_1'(i,j) \times Yavg_2'(i,j) + Yavg_1'(i,j) \times Xavg_2'(i,j)}{Area_{cell}} \quad (3.6) \end{aligned}$$

The basic idea behind these formulations is to break up rectangles spanning multiple cells into smaller ones (at cell boundaries), and handle the resulting rectangles in their appropriate cells. Of the above four cases, only $S_d(i,j)$ may cause multiple counting when we sum up the values from all the cells (only this case deals with rectangles that intersect in multiple cells). To adjust for this multiple counting, we can divide $S_d(i,j)$ by the mean of $AvgSpan_1$ and $AvgSpan_2$ i.e. the number of cells in which a rectangle in one dataset is likely to intersect with one rectangle in the other dataset. It should be noted that this is only an approximation to lessen

the impact of multiple counting of intersections, and is not exact. Finally, PH uses the following formula to estimate the required spatial join selectivity.

$$Size_{1\_2} = \sum S_a(i,j) + \sum S_b(i,j) + \sum S_c(i,j) + \frac{\sum S_d(i,j)}{\frac{AvgSpan_1 + AvgSpan_2}{2}} \tag{3.7}$$

### 3.3.2 Geometric Histogram(GH) Scheme

This is a completely novel approach to spatial join selectivity estimation that is proposed in this chapter. From Figure 3.2, one can observe that whenever two MBRs (rectangles) intersect with each other, the intersection is always another rectangle with four corners (let us call them *intersecting points*).

Each intersecting point could be the result of one of the following two situations: (a) A corner point of one MBR falls inside another MBR (in Figure3.2, there are two such points in cases 1 through 4, two points in cases 7 through 10, and four points in cases 11 through 12); (b) A horizontal line of one MBR intersects with a vertical line of another MBR (in Figure3.2, there are two such points in cases 1 through 4, four points in cases 5 through 6, and two points in cases 7 through 10). If we can accurately estimate how many intersecting points exist between the two datasets, simply dividing

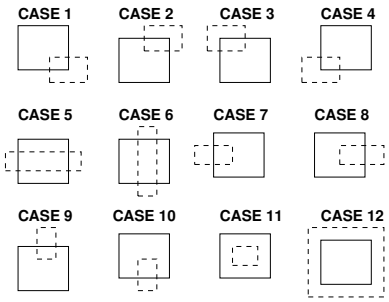



Fig. 3.2. All Possible Intersections of Two Rectangles

this estimate by four will provide us the desired spatial join selectivity. To estimate the number of intersecting points between the two datasets, we propose a novel approach called the Geometric Histogram (GH) Scheme.

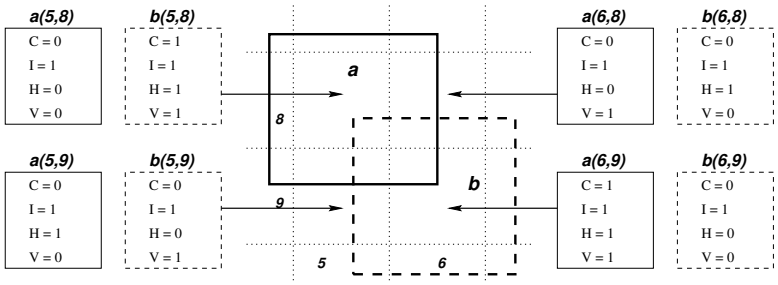



Fig. 3.3. Example for Basic GH

#### 3.3.2.1 Basic GH

GH builds a histogram file for each dataset by gridding the spatial extent into cells (buckets) as discussed for PH. For an intuitive explanation of how GH works, let us say we record the following information for each grid cell $(i, j)$: (a) how many vertical edges of MBRs pass through it ($V_k(i, j)$); (b) how many horizontal edges of MBRs pass through it ($H_k(i, j)$); (b) how many MBRs intersect it ($I_k(i, j)$); and (c) how many corner points of MBRs lie inside it ($C_k(i, j)$). Then an estimate for the number of intersection points between datasets $a$ and $b$ can be made as follows:

$$N_{a\_b} = \sum (C_a(i,j) \times I_b(i,j) + I_a(i,j) \times C_b(i,j) + V_a(i,j) \times H_b(i,j) + H_a(i,j) \times V_b(i,j)) \quad (3.8)$$

One can better understand this equation by examining the 16 cases of intersection in Figure 3.2 assuming that the gridding is done to such a fine granularity that the intersecting points between the two MBRs fall in different grid cells. In all these 16 cases, the above equation will correctly estimate four intersecting points (the first two terms calculate intersecting points corresponding to the sides of the two MBRs crossing each other, and the last two terms calculate intersecting points corresponding to a corner of one MBR falling within the other MBR). As an example, the number of intersecting points over the four grid cells that is shown in Figure 3.3 for MBRs $a$ and $b$ is calculated using the above equation at follows.

$$\begin{aligned} N_{a\_b} &= C_a(5,8) \times I_b(5,8) + I_a(5,8) \times C_b(5,8) + V_a(5,8) \times H_b(5,8) + H_a(5,8) \times V_b(5,8) + \\ &\quad C_a(6,8) \times I_b(6,8) + I_a(6,8) \times C_b(6,8) + V_a(6,8) \times H_b(6,8) + H_a(6,8) \times V_b(6,8) + \\ &\quad C_a(5,9) \times I_b(5,9) + I_a(5,8) \times C_b(5,9) + V_a(5,9) \times H_b(5,9) + H_a(5,9) \times V_b(5,9) + \\ &\quad C_a(6,9) \times I_b(6,9) + I_a(6,9) \times C_b(6,9) + V_a(6,9) \times H_b(6,9) + H_a(6,9) \times V_b(6,9) \\ &= 4 \end{aligned} \tag{3.9}$$

We then divide the number of intersecting points by 4 to get the desired spatial join selectivity (1 in this case).

#### 3.3.2.2 Revised GH

Equation 3.8 is based on the assumption that within a given cell, (a) every corner of the MBRs of one dataset falls inside all the MBRs of the other dataset which intersect this cell; and (b) every horizontal edge of the MBRs of one dataset intersecting this cell will intersect

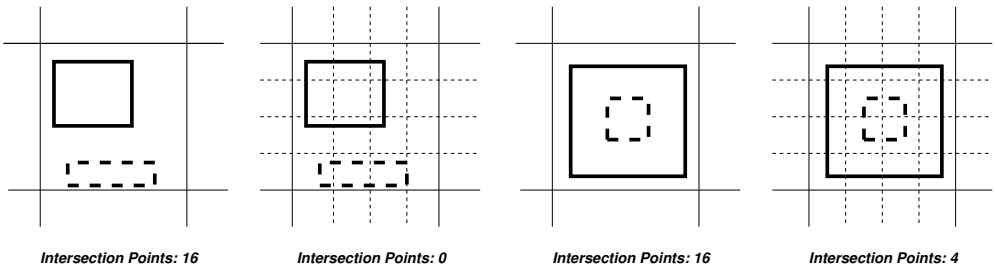



Fig. 3.4. Inaccuracies in Estimating Intersection Points with Basic GH

all the vertical edges of the MBRs of the other dataset intersecting this cell. This can lead to errors that are illustrated in Figure 3.4 due to the granularity of gridding. As we go for a very fine level of gridding, these errors would diminish, making the basic GH scheme more accurate. This is illustrated in Figure 3.4 which shows that the inaccuracies go away with a higher level of gridding. However, with a high level of gridding (number of grid cells grows exponentially), comes the high storage and processing costs, making it impractical. Instead, we propose to fix these inaccuracies by refining the basic GH scheme (with little additional overhead) as discussed below. The refinement is based on the assumption that data items are uniformly distributed within each grid cell.

To facilitate our discussion, we use the notations in table 3.3.2.2 representing the information GH will be needing for dataset $DS_k$ in each grid cell $(i, j)$.

Suppose we want to estimate the selectivity of spatial join between dataset $DS_1$ and $DS_2$. We will use MBRs a and b shown in Figure 3.5, which are from $DS_1$ and $DS_2$ respectively, to explain the basic idea of our approach when the estimation is done for the cell with

| Parameters | Description |
|---|---|
| $C_k(i,j)$ | number of corner points that fall within cell $(i,j)$. |
| $O_k(i,j)$ | sum of the ratios of the intersection area (with cell $(i,j)$ of MBRs to the cell area |
| $H_k(i,j)$ | sum of the ratios of horizontal intersections (with cell $(i,j)$) of MBRs to the cell width |
| $V_k(i,j)$ | sum of the ratios of vertical intersections (with cell $(i,j)$) of MBRs to the cell height |

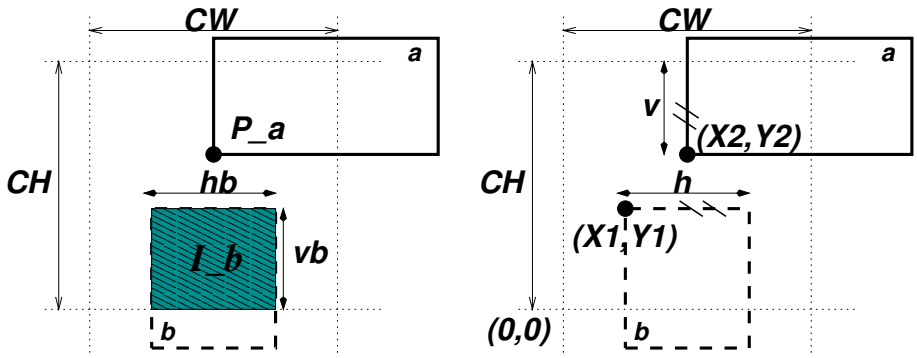
Table 3.2. GH Parameters

width $CW$ and height $CH$. The estimation of the interesting points within a given cell is done as follows:

- Estimating corner intersecting points (such as $P\_a$ falling within $b$ in Figure 3.5(a)):

  The shaded area $I\_b$ represents the intersection of MBR $b$ with the given cell, with the width and height of $I\_b$ being $hb$ and $vb$ respectively. Following the uniform distribution assumption, the probability of $P\_a$ falling in $I\_b$ is given by the ratio of the area of $I\_b$ (shaded area) to the area of the underlying cell, i.e. $\frac{hbvb}{CWCH}$. If $DS_1$ has $N$ corner points inside this cell, statistically $N \times \frac{hbvb}{CWCH}$ of these points are likely to intersect $I\_b$. Similarly estimating the intersections with the other MBRs of $DS_2$, gives $O_2(i,j) \times C_1(i,j)$ intersecting corner points of $DS_1$. Symmetrically there are $O_1(i,j) \times C_2(i,j)$ intersecting corner points of $DS_2$. Summing these two gives the total number of corner intersecting points in cell $(i,j)$.

- Estimating vertical and horizontal line intersection (such as $v$ intersecting $h$ in Figure 3.5(b)):




(a) Adjustment for Corner Intersection Points

(b) Adjustment for Vertical and Horizontal Intersection Points

Fig. 3.5. GH Adjustments

As shown in Figure 3.5(b), point $(X_1, Y_1)$ is the left endpoint of line h while point $(X_2, Y_2)$ is the bottom endpoint of line v. We thus have the following conditions.

$$X_1 \in [0, CW - h] \qquad Y_1 \in [0, CH] \qquad X_2 \in [0, CW] \qquad Y_2 \in [0, CH - v]$$

Suppose we have a variable $Z$ defined like this: $Z = \begin{cases} 1: & \textbf{if line h intersects line v} \\ 0: & \textbf{otherwise} \end{cases}$

Then, we can calculate the probaility that line h intersects line v using the following equation:

$$P(Z = 1) = \int_{y=0}^{CH} \int_{x=0}^{CW-h} P(Z = 1 | X_1 = x, Y_1 = y) f_{X_1,Y_1}(x, y) \, \mathrm{d}x\mathrm{d}y$$

Since $X_1$ and $Y_1$ are independent and distributed uniformly in their ranges respectively, we can get

$$\begin{aligned} f_{X_1,Y_1}(x,y) &= f_{X_1}(x) \cdot f_{Y_1}(y) \\ &= \frac{1}{CW-h} \cdot \frac{1}{CH} \end{aligned}$$

Hence,

$$\begin{aligned} & P(Z=1) \\ &= \int_{y=0}^{CH} \int_{x=0}^{CW-h} P(Z=1|X_1=x, Y_1=y) \cdot \frac{1}{CW-h} \cdot \frac{1}{CH} \, \mathrm{d}x\mathrm{d}y \\ &= \int_{y=0}^{CH} \int_{x=0}^{CW-h} P[x \le X_2 \le x+h, max(y-v,0) \le Y_2 \le min(y, CH-v) \\ & \quad |X_1=x, Y_1=y] \cdot \frac{1}{CW-h} \cdot \frac{1}{CH} \, \mathrm{d}x\mathrm{d}y \\ & \quad (Z=1 \quad iff \quad X_1 \le X_2 \le X_1+h \quad and \\ & \qquad max(Y_1-v,0) \le Y_2 \le min(Y_1, CH-v)) \\ &= \int_{y=0}^{CH} \int_{x=0}^{CW-h} P(x \le X_2 \le x+h) \cdot P[max(y-v,0) \le Y_2 \\ & \quad \le min(y, CH-v)] \cdot \frac{1}{CW-h} \cdot \frac{1}{CH} \, \mathrm{d}x\mathrm{d}y \\ &= \int_{y=0}^{CH} \int_{x=0}^{CW-h} \frac{h}{CW} \cdot \frac{min(y, CH-v) - max(y-v,0)}{CH-v} \cdot \frac{1}{CW-h} \\ & \quad \cdot \frac{1}{CH} \, \mathrm{d}x\mathrm{d}y \end{aligned}$$

**(following uniform distribution assumption)**

$$
\begin{aligned}
&= \int_{y=0}^{CH} \frac{h}{CW} \cdot \frac{min(y, CH-v) - max(y-v, 0)}{CH-v} \cdot \frac{1}{CW-h} \cdot \frac{1}{CH} \,\mathrm{d}y \\
&\quad \cdot \int_{x=0}^{CW-h} \mathrm{d}x \\
&= \frac{h}{CW} \cdot \frac{1}{CW-h} \cdot \frac{1}{CH} \cdot \int_{y=0}^{CH} \frac{min(y, CH-v) - max(y-v, 0)}{CH-v} \,\mathrm{d}y \\
&\quad \cdot (CW-h) \\
&= \frac{h}{CWCH \cdot (CH-v)} \cdot \int_{y=0}^{CH} [min(y, CH-v) - max(y-v, 0)] \,\mathrm{d}y
\end{aligned}
$$

Case i): $(CH-v) \geq v$, i.e. $CH \geq 2v$

$$
\begin{aligned}
&P(Z=1) \\
&= \frac{h}{CWCH \cdot (CH-v)} \cdot \int_{y=0}^{CH} [min(y, CH-v) - max(y-v, 0)] \,\mathrm{d}y \\
&= \frac{h}{CWCH \cdot (CH-v)} \cdot [\int_{y=0}^{v} y \,\mathrm{d}y \\
&\quad + \int_{y=v}^{CH-v} y - (y-v) \,\mathrm{d}y + \int_{y=CH-v}^{CH} (CH-v) - (y-v) \,\mathrm{d}y] \\
&= \frac{h}{CWCH \cdot (CH-v)} \cdot [\frac{v^2}{2} + v \cdot (CH-2v) + CH^2 - CH \cdot (CH-v) \\
&\quad - \frac{1}{2} \cdot (CH^2 - (CH-v)^2)] \\
&= \frac{h}{CWCH \cdot (CH-v)} \cdot [v \cdot (CH-v)] \\
&= \frac{hv}{CWCH}
\end{aligned}
$$

Case ii): $(CH-v) < v$, i.e. $CH < 2v$

$$
\begin{aligned}
&P(Z=1) \\
&= \frac{h}{CWCH \cdot (CH-v)} \cdot \int_{y=0}^{CH} [min(y, CH-v) - max(y-v, 0)] \,\mathrm{d}y
\end{aligned}
$$

$$
\begin{aligned}
&= \frac{h}{CWCH \cdot (CH - v)} \cdot [\int_{y=0}^{CH-v} y \, \mathrm{d}y + \int_{y=CH-v}^{v} (CH - v) \, \mathrm{d}y \\
&\quad + \int_{y=v}^{CH} (CH - v) - (y - v) \, \mathrm{d}y] \\
&= \frac{h}{CWCH \cdot (CH - v)} \cdot [\frac{(CH - v)^2}{2} + (v - CH + v) \cdot (CH - v) \\
&\quad + CH^2 - CH \cdot v - \frac{CH^2 - v^2}{2}] \\
&= \frac{h}{CWCH \cdot (CH - v)} \cdot [v \cdot (CH - v)] \\
&= \frac{hv}{CWCH}
\end{aligned}
$$

From case i) and ii), we get that the probability that a vertical line of size $v$ intersects with a horizontal line of size $h$ inside a 2-dimensional space of $CW \times CH$ is $\frac{hv}{CWCH}$. Adding this probability for all the vertical lines of $DS_1$ and horizontal lines of $DS_2$, we are likely to have $H_2(1, j) \times V_1(i, j)$ such intersecting points. Intuitively, we can get to this reasoning by going back to Equation 3.1 which estimates the number of intersecting rectangles in a 2-D space. If we simply set the areas $C_1$ and $C_2$ to zero, since we are dealing here with lines instead of rectangles, equation 3.1 degenerates to the formula used here. Symmetrically, we are likely to have $H_1(i, j) \times V_2(i, j)$ horizontal lines of $DS_1$ intersecting with vertical lines of $DS_2$ in cell $(i, j)$.

Putting these arguments together, we estimate the number of intersecting points ($IP$) using the following equation:

$$
\begin{aligned}
IP &= \sum (C_1(i, j) \times O_2(i, j) + C_2(i, j) \times O_1(i, j) \\
&\quad + H_1(i, j) \times V_2(i, j) + H_2(i, j) \times V_1(i, j)) \qquad (3.10)
\end{aligned}
$$

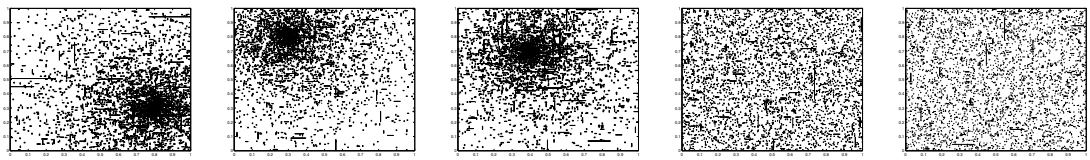
This number is then divided by 4 to get the desired selectivity estimation.

## 3.4 Evaluating the Analysis Techniques

In this section, we evaluate the accuracy and costs of the different sampling and histogram based techniques in estimating spatial join selectivity using a spectrum of real and synthetic datasets.

### 3.4.1 Datasets

To stress the pros and cons of the different schemes and their universal applicability, we have used a wide spectrum of real and synthetic datasets on which we conduct the spatial join operation. The selected datasets are quite diverse, and include both uniform and skewed spatial distributions. While the real datasets contain points, polylines and polygons, these are abstracted by their bounding boxes (MBRs) in our experiments, and the spatial join predicate is to find intersecting MBRs across the two datasets. In addition to pictorial views of these datasets in Figures 3.6 and 3.7, a brief description follows:

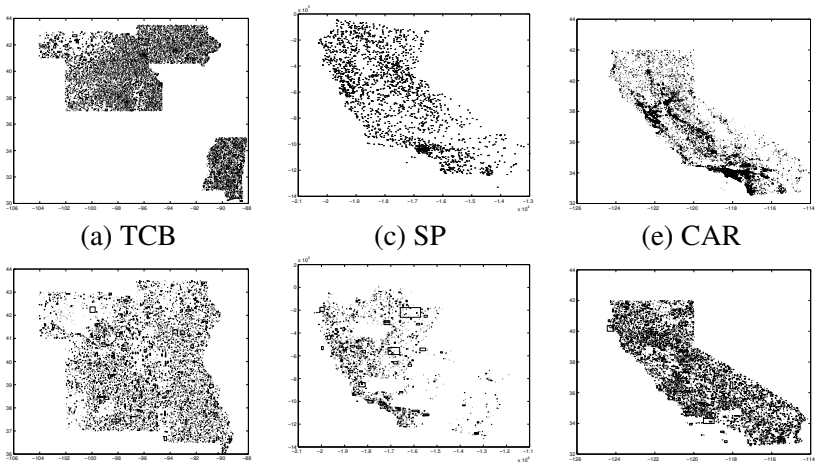


(a) SCRA (b) SCRB (c) SCRC (d) SURA (e) SURB

Fig. 3.6. Synthetic Datasets

- Synthetic Datasets:
  - SCRA: Synthetic Clustered Rectangles-A, containing 100,000 rectangles clustered around (0.8, 0.3) in a 1×1 space. The size (area) of these rectangles follow a normal distribution with mean $3 \times 10^{-5}$ and deviation $1 \times 10^{-6}$;
  - SCRB: Synthetic Clustered Rectangles-B, containing 100,000 rectangles clustered around (0.3, 0.8) in a 1×1 space. The size of these rectangles follow a normal distribution with mean $2 \times 10^{-5}$ and deviation $1 \times 10^{-6}$;
  - SCRC: Synthetic Clustered Rectangles-C, containing 100,000 rectangles clustered around (0.4, 0.7) in a 1×1 space. The size of these rectangles follow a normal distribution with mean $3 \times 10^{-5}$ and deviation $1 \times 10^{-6}$;
  - SURA: Synthetic Uniformly distributed Rectangles-A, containing 100,000 rectangles that are uniformly distributed in a 1×1 space. The size of these rectangles follow a normal distribution with mean $1 \times 10^{-5}$ and deviation $1 \times 10^{-6}$;
  - SURB: Synthetic Uniformly distributed Rectangles-B, containing 100,000 rectangles uniformly distributed in a 1×1 space. The size of these rectangles follow a normal distribution with mean $4 \times 10^{-6}$ and deviation $1 \times 10^{-6}$.
- Real Datasets:
  - TCB: Tiger Census Blocks, containing the MBRs of 556,696 census blocks (polygons) of Iowa, Kansa, Missouri and Nebraska from the TIGER/Line(R) datasets [83];
  - TS: Tiger Streams, containing the MBRs of 194,971 streams (polylines) of Iowa, Kansas, Missouri and Nebraska from the TIGER/Line(R) datasets [83];

- SPG: Sequoia PolyGons, containing the MBRs of 79,607 polygons taken from the Sequoia benchmark [115];
- SP: Sequoia Points, containing the MBRs of 62,555 points taken from the Sequoia benchmark [115];
- CAR: California Roads, containing the MBRs of 2,249,727 streets (polylines) of California from the TIGER/Line(R) datasets [83];
- CAS: California Streams, containing the MBRs of 98,451 streams (polylines) of California from the TIGER/Line(R) datasets [83];



(a) TCB (c) SP (e) CAR

(b) TS (d) SPG (f) CAS

Fig. 3.7. Real Datasets

Using these datasets we consider different combinations of spatial joins that capture interesting and diverse facets as follows:

- joins between datasets of different types such as TCB with TS (census blocks with streams), and SPG with SP (points with polygons);
- joins between datasets of roughly the same number of data items such as SCRA with SCRB, and SPG with SP;
- joins between datasets with different numbers of data items such as CAR with CAS;
- joins between datasets that have similar spatial skews such as SCRB with SCRC (skewed around the same point), and SURA with SURB (both uniformly distributed).
- joins between datasets that have different spatial skews such as SCRA with SCRB (skewed at different spatial locations), and SCRA with SURA (skewed with uniform).

### 3.4.2 Metrics of Interest

To evaluate the pros and cons of the different techniques, we consider the following metrics:

- *Estimation Error* to evaluate the accuracy of the techniques, that is expressed as the difference between that predicted by the techniques and the actual join selectivity normalized as a percentage with respect to the actual join selectivity.
- *Estimation Time*, which is the time to conduct the estimation relative to the time to perform the actual join using R-tree indices for the datasets (incidentally, for the sampling schemes, this overhead is given for both with and without the availability of the R-trees for the datasets, with the latter obviously resulting in a smaller estimation overhead).
- *Space Cost*, which is the overhead in bytes for storing the required information for each technique, expressed as a percentage of the space required to maintain the R-trees for the actual datasets.

- *Building Time*, which is the time taken to construct the necessary information (histogram file for the histogram-based schemes, and samples for the sampling schemes), expressed as a percentage of the time taken to build the R-trees for the actual datasets.

| Join | Dataset Sizes (Num. of Items) | Rtree Sizes (Kilobytes) | Rtree Building Times (Seconds) |
|---|---|---|---|
| SCRA ⋈ SCRB | 100,000 ⋈ 100,000 | 6,939 ⋈ 6,905 | 90.152 ⋈ 87.856 |
| SCRB ⋈ SCRC | 100,000 ⋈ 100,000 | 6,905 ⋈ 6,971 | 87.856 ⋈ 86.331 |
| SCRC ⋈ SURA | 100,000 ⋈ 100,000 | 6,971 ⋈ 6,824 | 86.331 ⋈ 87.335 |
| SURA ⋈ SURB | 100,000 ⋈ 100,000 | 6,824 ⋈ 6,873 | 87.335 ⋈ 85.305 |
| TS ⋈ TCB | 194,971 ⋈ 556,696 | 13,279 ⋈ 37,535 | 157.588 ⋈ 510.97 |
| CAS ⋈ CAR | 98,451 ⋈ 2,249,727 | 6,455 ⋈ 145,031 | 89.935 ⋈ 2582.13 |
| SP ⋈ SPG | 62,555 ⋈ 79,607 | 4,145 ⋈ 5,628 | 58.329 ⋈ 79.461 |

| Join | Rtree Join Time (Seconds) | Join Result Size (Num. of Pairs) |
|---|---|---|
| SCRA ⋈ SCRB | 17.739 | 595,649 |
| SCRB ⋈ SCRC | 35.959 | 2,784,717 |
| SCRC ⋈ SURA | 17.589 | 956,199 |
| SURA ⋈ SURB | 12.395 | 348,080 |
| TS ⋈ TCB | 44.748 | 1,186,887 |
| CAS ⋈ CAR | 151.487 | 3,092,847 |
| SP ⋈ SPG | 2.416 | 138,832 |

Table 3.3. Statistics on Datasets and the actual Spatial Join

A relatively low estimation error (we believe approximately 10% error would be a good target for query optimization), and estimation time will be preferred. While building time is important if the target of the estimation is intermediary result(s) of a long query, space cost is less important given the large amount of storage availability these days (as long as the storage requirements do not become comparable or exceed the dataset size itself). The statistics on the actual join of these datasets, together with the details on their R-trees, are given in Table 3.3 for the reader's reference.

### 3.4.3 Results for Sampling Techniques

Figures 3.8 and 3.9 show the results for the estimation of the spatial join selectivity with the sampling schemes for the synthetic and real datasets respectively. All the bar graphs in these figures follow the same convention. The x-axis represents different sample size combinations. The first three sets of bars in these figures use samples (of sizes 0.1%, 1% and 10% of the datasets) from both datasets for the estimation. The fifth to ninth sets of bars use a sample from only one of the datasets, with the entire other dataset (shown as 100) being used. The individual bars within each set show the performance for the three ways of selecting a sample that were discussed in Section 3.2.

All these graphs show the estimation error as was defined in the previous subsection. The time cost is shown in two forms: *Est. Time 1* is the time overhead in selecting samples, building the R-trees from the samples and then performing the join, as a percentage of the time to do the actual join assuming the R-trees on the datasets are not available (i.e. they are built before the join is performed); and *Est. Time 2* is the same overhead assuming that R-trees are available, in which case the R-trees need not to be constructed for the original datasets. Obviously, Est. Time 1 is lower (as a relative percentage) compared to Est. Time 2. The building time for constructing the samples is already included in these time costs. The space overheads are not explicitly shown here since they are apparent from the size (in percentage) of the samples that are chosen.

One can intuitively hypothesize that larger the sample, the more accurate the estimation. While this is an overall trend, we do find exceptions in some cases (such as RS for SCRB with SCRC when we go from 1/1 to 10/10, RSWR for SCRC with SURA when we go from 1/1 to 10/10, etc.). This is because the sampling idea is based on statistical arguments, and it is

impossible to definitely say that a larger sample will necessarily give a more accurate estimate. However, it is fairly obvious that larger samples incur higher time and space costs, as is apparent from the graphs.

We uniformly find in Figures 3.8 and 3.9 that using all of one dataset and picking samples from only the other dataset does not pay off. The accuracy of this approach is not significantly better than picking a 10% sample from both datasets, and is in fact worse in many cases. Further, the time overheads are much worse than taking samples from both datasets if the R-trees on the two datasets are not available.

The other important consideration is the impact of the dataset size (or rather, the difference between the sizes of the two inputs to be joined) on the effectiveness of sampling. In general, we find that taking a smaller fraction from the larger dataset results in better estimation accuracy than taking the same fraction from the smaller dataset. This results in a much better statistical approximation of the two datasets. This also makes sense from the time cost viewpoint, since a larger fraction of the larger dataset incurs higher estimation overhead.

Between the three ways of picking samples, we find that Sorted Sampling (SS) is not really a good choice. While its accuracy is not significantly better than the other two (in fact, it is worse in some cases), the sorting significantly adds to the time costs compared to the other two strategies. Regular (RS) and Random (RSWR) are more or less comparable, particularly for the synthetic datasets, where the data items are anyway generated randomly. With the real datasets, their relative performance really depends on the vagaries of the dataset (RSWR does better in two cases, and is comparable in the third). Hence, it is suggested that samples be generated randomly (RSWR) from the datasets.

In general, we find that if the R-trees are not available for the datasets, we can get the estimation error within 10% with sample sizes of 10% (i.e. 10/10), with time overheads that are also within 10% for random sampling (RSWR). This suggests that random sampling may be a viable option for spatial join estimation for intermediate steps/results (where the dataset is not previously available) of a long/complex query execution. When the R-trees are already available for the datasets, the results show that the estimation time costs (Est. Time 2) are much higher to get reasonable accuracy. However, one could argue that if R-trees are already available, then the samples for a dataset (and the R-trees on these samples) could also be made available beforehand. To find out if such an approach has any merit, we show in Figure 3.10, the Est. Time 2 incurred by RSWR with 10/10 samples for the different joins, that is given when the R-trees are already available for the two datasets. As can be seen, with the availability of the R-trees on the samples, RSWR becomes once again a possible option with the estimation time cost being less than 10%.

### 3.4.4 Results for Histogram Based Techniques

We next consider the two histogram-based techniques proposed in this thesis. Figures 3.11 and figure 3.12 show the performance of the PH and GH schemes for the synthetic datasets and real datasets respectively. In all these graphs, the x-axis depicts the level of gridding ($h$, where $4^h$ is the resulting number of grid cells into which the spatial extent is histogrammed). The results are shown in terms of the estimation error, estimation time, building time (for constructing the histograms), and the space overhead that have been described earlier.

We focus first on the results for PH. It should be noted that the PH results for $h = 0$ (the left most point in the curves) denotes the parametric model that has been originally proposed in

[13], where the universe is assumed to be uniformly distributed and a simple formula is used to estimate spatial join selectivity based on this assumption. The other levels divide the space into equi-sized regions (cells), and use the uniformity assumption within each such region. There are two factors affecting the accuracy of the estimation as the number of levels is increased. We can expect better accuracy since a finer level of gridding will help better adhere to the uniformity assumption within each grid cell. However, finer gridding can result in data items spanning several grid cells, causing the estimation to multiple count (in several cells) the intersections (leading to an overestimation). Consequently, we expect the accuracy curves to first trend downward (the former factor is more significant) and then trending upward (the latter factor becomes more significant at higher $h$). This can be observed for the SCRA with SCRB, SCRB with SCRC and TCB with TS joins. Since the datasets for these joins are clustered, the uniformity assumption hurts at lower levels. In all these cases, we find that we do not want to go beyond levels 4 or 5, since the multiple counting starts hurting accuracy. In the joins for CAR with CAS and SPG with SP, the errors keep dropping even upto level 9. As can be observed pictorially, these datasets are highly skewed making the uniformity assumption a severe restriction at lower levels. In the joins for SCRC with SURA and SURA with SURB, the uniformity assumption holds (SURA and SURB have been generated that way) causing the multiple counting factor to become more significant even at level 1. With increasing levels, the time and space costs go up as well. However, even at level 9, the estimation time takes less than 10% of the cost of performing the actual join, and the time for building the histogram file is also a rather small percentage of the time to build the R-trees. The sudden spike in building times at high levels is because the histogram file gets too large to fit in memory. It should be noted that the histogram file size is purely dependent on the level of gridding and not on the dataset itself. In summary, the PH scheme gives acceptable

(10% errors) accuracy at level 5, with the time and space costs being negligible at this level of gridding.

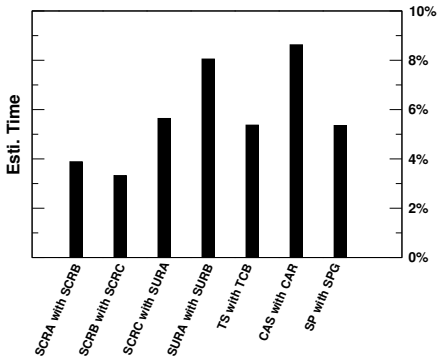



Fig. 3.10. Estimation Time for RSWR with respect to the time to do the actual join when the R-trees on the datasets are available assuming Sample R-trees (for 10/10) are available

Moving on to the GH scheme, we find the estimation errors monotonically decrease with the level of gridding. One can recall that this scheme attempts to avoid the double counting problem. As a result, it does not have the drawback that PH had with higher grid cells. Increasing the gridding level makes the cells small enough so that the information within the cell is more accurately captured (false intersections are discounted). Consequently, the errors only decrease with gridding level. This is a nice property of GH which makes it somewhat more attractive than PH or any of the sampling schemes that are more unpredictable. The estimation time for GH is even lower than for PH. In fact, GH is very accurate (less than 5% errors) in all the seven joins that are shown here, at level 7 (where the estimation time is around 1% or less). The space overhead for storing the histogram is typically 10% or lower for GH at this level.

In summary, GH is much more desirable than PH. Not only is the accuracy better for GH, but the results are much more stable as we increase the gridding level (PH requires us to find a good sweet spot for the gridding level). GH requires less space than PH (compare the information stored for the two schemes in Tables 3.3.1.2 and 3.3.2.2), and is also slightly less

time consuming for each grid cell (compare Equations 3.7 and 3.10). These factors make GH a much better option than PH.

### 3.4.5 GH: Impact of Dataset Size

It is interesting to find out how the dataset size affects the performance of GH. To investigate this, we use the two synthetic datasets - SCRA and SCRB - and crank up the number of data items (250K, 500K, 750K, 1000K, 1500K) in each using the underlying distribution of the dataset, and the results are shown in Figure 3.13. As the dataset gets larger, we find that the estimation error as a percentage drops, which is good news. Further, the estimation overheads, both time and space, also decrease with a larger dataset size. These results further emphasize the benefits of GH as we progress to larger datasets in the future.

### 3.4.6 Estimating Self-Join Using GH

Self-join is a specific instance of spatial join, which by itself can be widely used on a single dataset. For instance, one may be interested in finding out the number of crossroads in a dataset of streets, which is essentially a self-join. We evaluate how well GH is able to handle self-joins with four datasets, and the results are shown in Figure 3.14. The predicate for the join is again to find pairs of intersecting MBRs, though the MBRs are within the same dataset. As can be seen, GH again does a good job giving fairly good estimates at very reasonable costs.

## 3.5 Chapter Summary

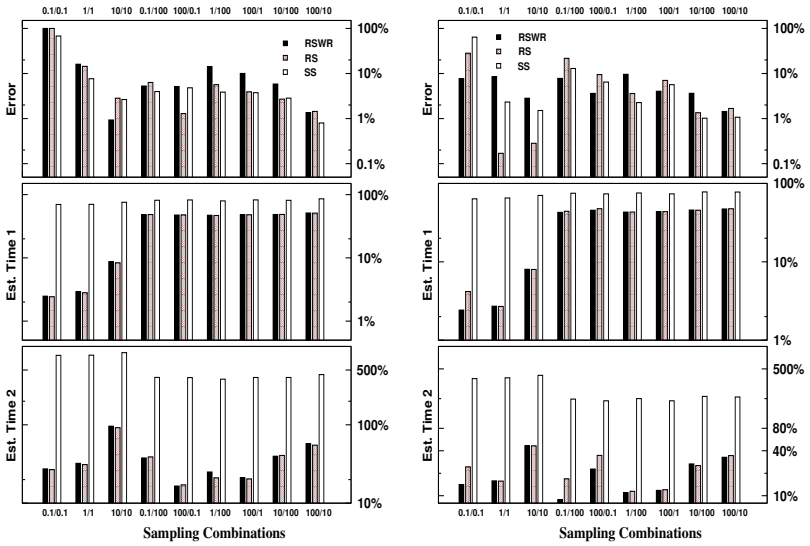
A recent article [108] identifies analysis of common spatial operations to be a crucial and daunting open problem for the success of SDBMS. In the last ICDE [66], we attacked the selectivity estimation of range queries and proposed a histogram-based technique that accurately

estimates the selectivity of this operation with less than 5% error at little cost. We believe that in this chapter, we have taken a similar step towards spatial join operations.

While the main contribution of this chapter is a novel histogram-based technique for estimating spatial join selectivity, this chapter has also for the very first time explored the suitability of well-known sampling techniques on this problem. Considering three ways of picking samples, and using a diverse range of both synthetic and real datasets, we have investigated the accuracy, as well as the time and space costs of sampling based analysis of spatial joins. Sampling techniques are more attractive when the original datasets do not have an index structure associated with them, in which case we can get around 10% accurate estimates within reasonable overheads. Even with the availability of index structures, this option become viable when the samples have been picked beforehand. However, the problem with sampling is that the results are unstable, and it is not clear what would be an ideal sample size.

On the other hand, the histogram based technique that we have proposed (GH), gives very accurate results (within 5% error), often taking less than 5% of the time that it would take to perform the actual join. It turns out to be a better option than an adaptation of a prior [13] parametric technique for histograms. GH becomes even more attractive as we move to larger datasets in the future, and is also fairly accurate for self-join.

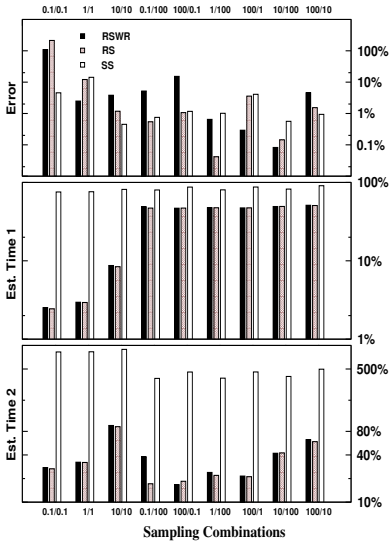
In the future, we would like to develop analysis techniques for estimating selectivity and I/O costs for other spatial database operations, in addition to developing a SDBMS incorporating query optimizations based on these analysis techniques.



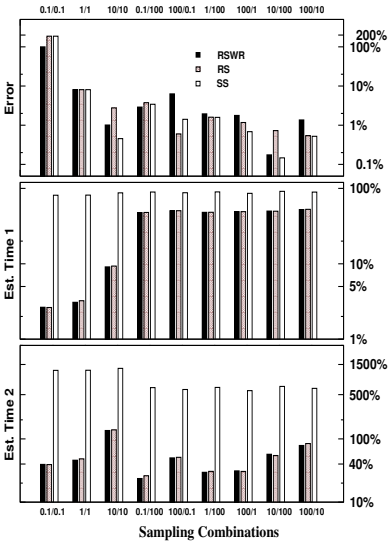

(a) SCRA with SCRB




(b) SCRB with SCRC

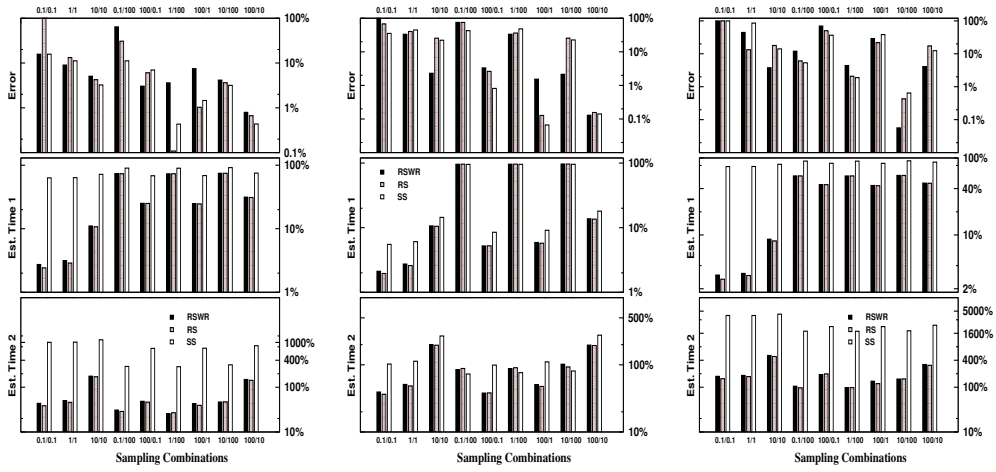



(c) SCRC with SURA



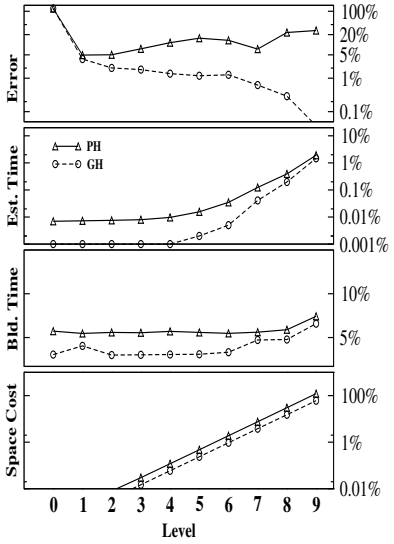

(d) SURA with SURB

Fig. 3.8. Sampling Techniques on Synthetic Datasets

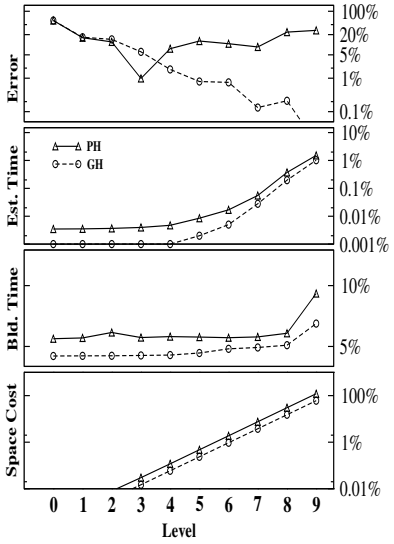



(a) TS with TCB (b) CAS with CAR (c) SP with SPG

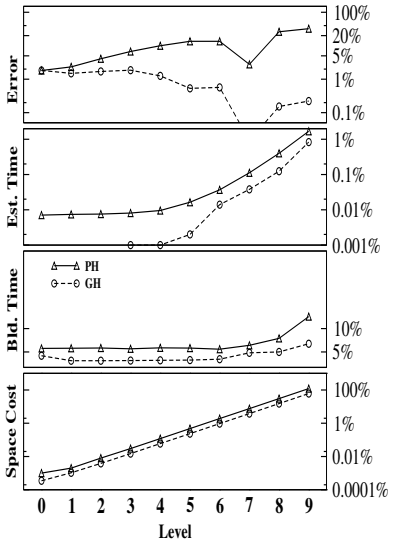
Fig. 3.9. Sampling Techniques on Real datasets



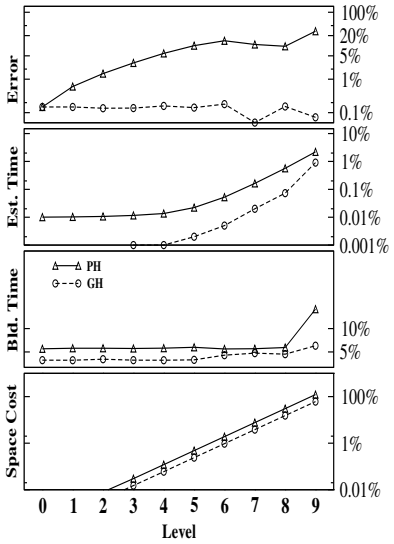

(a) SCRA with SCRB




(b) SCRB with SCRC

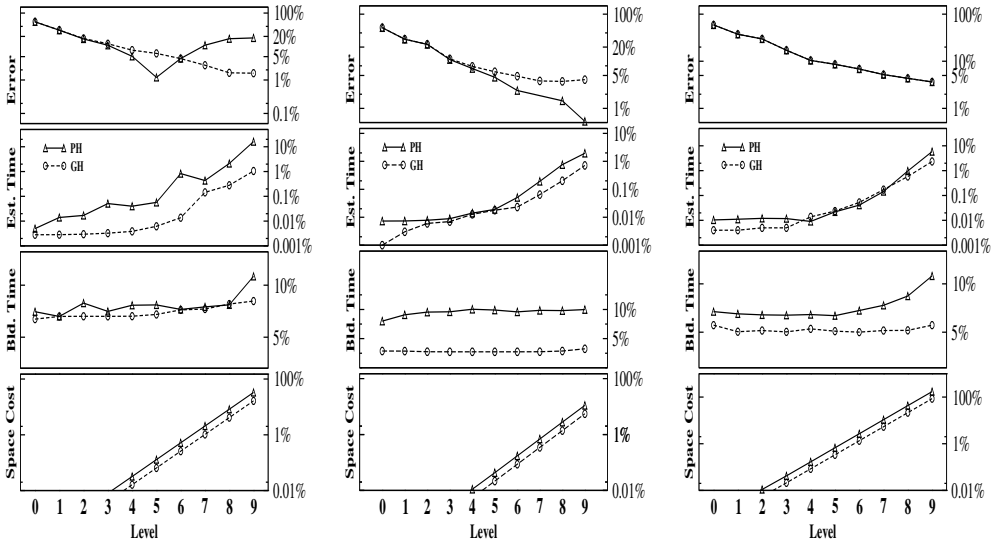



(c) SCRC with SURA




(d) SURA with SURB

Fig. 3.11. Applying Histogram-based Techniques on synthetic datasets




(a) TCB with TS (b) CAR with CAS (c) SPG with SP

Fig. 3.12. Applying Histogram-based Techniques on real datasets

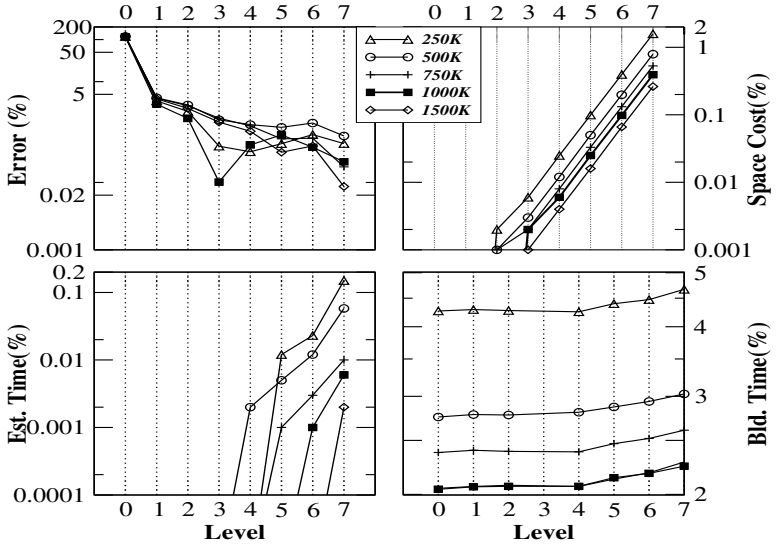



Fig. 3.13. Impact of Dataset Size for GH

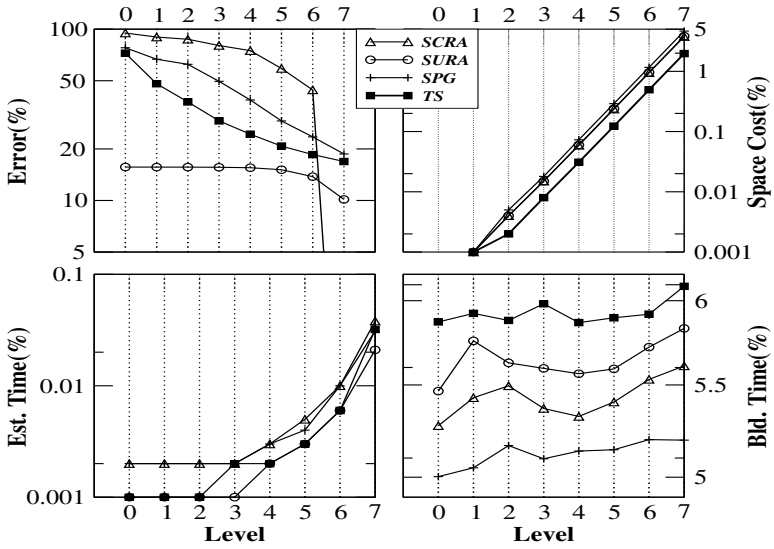



Fig. 3.14. Using GH to Estimate Selectivity of Self-Join

# Chapter 4

# Analyzing Energy Behavior of Spatial Access Methods for MemoryResident Data

## 4.1 Introduction

Computing is becoming a pervasive and ubiquitous part of everyday life. The traditional modus-operandi of sitting at a desk to interact with a computer system is gradually going out of style, with users demanding access to computational resources and information whenever and wherever (even when they are on the move) they choose. These needs have opened the door to several interesting and crucial topics for research in the broad domain of mobile and resource-constrained computing. Focusing specifically on spatial databases (an important and useful class of mobile applications), this chapter explores the energy (a scarce and valuable resource in mobile devices) consumption and performance trade-offs of different storage organizations for spatial data on resource-constrained mobile devices.

Programs running on mobile devices (PDAs, laptops, etc.) can be subject to very different operating conditions compared to their desktop/server counterparts. This includes limited computational resources, storage capacity, battery energy, and connectivity, that are a consequence of design considerations such as small form factor, weight, cost and diverse operating conditions. It is widely recognized that battery energy is, perhaps, one of the the most challenging limitations, with many other factors (such as computational speed) directly or indirectly related to energy availability. Mobility precludes the use of a wall socket to power the device, and at the same

time one does not wish to carry a heavy battery along for its operation. The growing mismatch between energy capacity of batteries and the energy consumption of mobile devices makes it all that much more critical to employ algorithmic, software and architectural techniques for energy savings. It is hypothesized [63] that high level optimizations in algorithms and data structures can give much more energy savings than micro-managing the energy consuming resources at a very low level. Such optimizations can even amplify the savings obtained from well-known low level energy saving techniques [120], and are thus the motivation for this work.

Database applications are expected to be one of the dominant workloads running on the mobile devices [4]. Apart from the prevalent personal organizer database applications (address book, calendar, etc.), there are numerous productivity-enhancing commercial, entertainment and convenience-based database applications envisioned for such devices. Here, we specifically focus on spatial databases, an important class of applications for the mobile devices. In general, Spatial Database Management Systems (SDBMS) [108] have found widespread adoption in numerous areas including Geographical Information Systems (GIS), Image Processing, Military Planning and Logistics, Computer Aided Design (CAD), Multimedia Systems, and Medical Database Systems. SDBMS are important for mobile computing, with several possible applications in this domain. Already, mobile applications for spatial navigation and querying using a street atlas are available for many PDAs [80, 41]. In addition, traditional data input and querying for conventional SDBMS can be supplanted by mobile operations for better productivity and convenience.

Even in a resource-rich environment, SDBMS design and implementation is a difficult problem [108], because the system has to deal with multidimensional data as opposed to conventional databases, where single dimensional indices may suffice. Data objects have varying sizes

associated with them, and spatial operations are in general much more complex than standard relational operators. Further, the efficiency of the storage organization is highly dependent on the nature and idiosyncrasies of the spatial dataset. Moving the target to a mobile device makes the design and implementation of a SDBMS even more challenging. Resource constraints such as limited energy, computational power and memory add to the complexity of the problem. Performance is not necessarily the only goal for optimization. Sometimes a user may be willing to sacrifice some amount of performance if that will enable a device to run longer on battery. Further, power dissipation of different system components may also be an important issue for thermal considerations.

There are several important and interesting issues in designing a SDBMS for a resource-constrained mobile device, and a few of them include *connectivity* (communication), *data storage and access methods*, *query processing*, and *dynamic adaptation* which are detailed below.

With limited resources, there is the important question of where should the operations (queries) be performed. Does it make sense to ship an operation to a resource-rich server (which may, perhaps, have access to the data) and simply ask a mobile device to act as an intermediary to display the end results to the inquiring user, or should the device itself perform the operation? There are several practical considerations that may force the latter choice. The first reason is connectivity. It may not always be possible to be able to connect to a resource-rich machine (e.g. the connection could be bad, a user may be in an unreachable location, or a user may not have subscribed to a service for communication). Communication is also energy consuming in the wireless components, and it is essential to trade-off the computation energy saved with the additional communication cost [58]. Second, even if the user is able to connect, there is the issue of privacy (to avoid giving out query parameters or user location). Finally, several (not all) spatial

databases are static with information rarely updated. For example, once a road atlas is download to a mobile device, the user may not want any more updates till the device is resynchronized explicitly later on. All these reasons may warrant the storage of the spatial data on the mobile device itself, and with the queries directly performed on it. Work partitioning between a mobile device and a server is also briefly explored in this chapter. A more detailed study of this topic is given in [47], and for most of the initial discussions in this chapter we consider the dataset to be fully resident on the mobile device.

If the dataset needs to be stored in a mobile device, how should it be organized for good performance? Earlier work has focussed mainly on optimizing the retrieval and processing of large disk-resident spatial datasets on server environments. It is imperative to revisit this issue for resource-constrained devices with limited memory and without the presence of a disk (while laptops are equipped with small disks, few other mobile devices enjoy this luxury) not only from the performance viewpoint, but from the energy consumption angle as well.

Query processing and optimization is always a key determinant to performance [62, 88]. Decomposing the high level user request into the fundamental database operations, and deriving a query execution path should be based on both performance and energy consumption. Dynamic adaptation based on changing resource constraints (such as energy, connectivity, etc.) is another important consideration. Modulation of the storage structures, query execution and optimizations is needed when the operating conditions are changing.

These are just a few of the important issues for consideration when designing a SDBMS for a mobile device. Examining all these issues is overly ambitious, and is well beyond the scope of our study. Instead, we specifically focus on the second problem listed above: *what are the performance and energy implications of storing and processing memory-resident spatial*

*data on a resource-constrained device?* In particular, we address the issue of storing spatial data in main memory and performing certain basic spatial operations on this data including point queries, range queries and nearest-neighbor queries. We assume that all of the dataset is resident in the memory of a mobile device, there is no necessity for communication with a server (no dynamic updates), and complex queries (and their optimizations), such as spatial joins, are not considered. This is a largely unexplored area, with most previous work on spatial databases examining storage organizations on disks of resource-rich environments. Memory resident spatial data organization has not been extensively studied from the performance angle, let alone the energy viewpoint.

The first step to the development of energy and performance efficient storage organizations for memory-resident spatial data is a rigorous examination of the pros and cons of the already existing solutions [39] that have been proposed for resource-rich environments. Such a study can not only identify energy-performance trade-offs between the existing solutions, but can suggest enhancements, or can even suggest entirely new storage organizations (though this issue is not extensively explored in this chapter). At the same time, performance and energy profiles can suggest architecture/hardware enhancements to improve the performance and energy savings of resource-constrained systems. This is similar to the motivation behind a recent study [3] that has examined the execution profile of commercial relational DBMSs, except that our focus here is on SDBMS and energy profiling (together with performance-energy trade-offs) that has not been explored before. Our study takes the first step to the development of energy-efficient SDBMS by attempting to answer the following important questions:

- How do the previously proposed alternatives for spatial data organization such as Quadtrees [54, 55], R-trees [48, 68] and Buddy trees [105, 104], compare for memory resident

datasets in terms of performance? What are the energy consumptions of these different structures when answering queries?

- During the processing of a query, how much energy and time are expended in traversing the index structures to identify candidates that are potential solutions for the query (filtering step)? Subsequently, how much energy and time are expended in performing the geometric operations on the actual candidate data items to find the exact solutions (refinement step)? Such software profiles are very useful to find hotspots for potential optimization (code restructuring), and to study the pros and cons of the structures in detail.

- For each phase of query processing, how much energy is consumed by the different hardware components of a mobile device - processor core, processor clock, cache, memory and buses? Such a hardware profile can also help us structure the code and suggest architectural enhancements to fix hardware hotspots, potentially without extending the execution time.

- How does the nature of the queries affect the performance and energy profiles? Spatial proximity can translate to improved locality in the data access patterns of the processor, thus reducing the cache and memory energy consumption. At the same time, queries resulting in the selection of several data items can cause capacity and conflict misses in the cache, thereby increasing the energy consumption of the memory hierarchy.

- Traditionally node sizes of the hierarchical index structure are governed by performance related issues such as disk access costs, tree spans, etc. With memory-resident structures, how important a role does node size play in performance for spatial data? Are there any additional insights that an energy perspective can give to the choice of a good node size?

- What is the impact of technological trends on the relative performance of the schemes? In particular, with the observed hardware and software profiles, what energy and performance enhancing architectural features would be beneficial?

- Is there anything to gain in terms of energy and/or performance by offloading the query to a resource-rich server, where energy is not a concern, or should it be done only on the client because of the communication overheads?

To explore these issues, we use a detailed energy and performance estimation execution-driven simulator, called SimplePower [120], that is available in the public domain. A wireless network interface model has been added to this simulator. Four different storage organizations have been implemented on this simulator, and they have been used to evaluate three kinds of spatial queries on four different datasets. Detailed hardware and software profiles are used to answer the questions listed above.

To our knowledge, this is the first study to present the energy and performance profiles/trade-offs for storage organizations for main memory spatial datasets. By presenting the first set of results on this topic, this study [10, 6] not only fills an important void in this area, but also sets the tone for future research on energy optimized storage organizations. The rest of this chapter is organized as follows. The next section puts this study in perspective with the current state-of-the-art. Section 4.3 gives a quick overview of previously proposed index structures that are used in this evaluation. Section 4.4 explains the experimental setup and workloads. The results are presented in Section 4.5 and their implications are given in Section 4.6. Section 4.7 summarizes the contributions of this chapter.

## 4.2 Related Work

A great deal of prior work has been done in the area of storage organizations for spatial (multidimensional) data [102, 39]. This has led to the development of numerous index structures such as Grid Files [53], Quad-trees [54, 55], $k-d$-trees [18], $k-d$-B-trees [97], Cell-trees [46], BANG files [37], $h$B-trees [76], Buddy-Trees [105], R-Trees [48] and its variations $R$+-Tree [106] and $R^*$-Tree [82]. The book [102] and survey articles [39] are testament to the indepth research that has been done in this area. In fact, there is a figure in [39] which shows the pictorial evolution of spatial data structures in which over 56 multidimensional access methods are depicted. All these structures have been proposed and evaluated from the performance, scalability, space overheads, simplicity, and concurrency management viewpoints. There has been no prior study examining their energy consumption behavior. Further, many of these are meant to be secondary storage index structures, and there has not been a detailed comparison of their suitability for memory resident datasets.

On the normal relational databases front, there have been investigations on memory resident datasets [73]. This has included new index structures and alternate algorithms for selection/projection/join to accommodate main memory processing. There has also been a recent study [3] that has profiled the execution time of a relational DBMS to understand the processor and memory implications on memory resident datasets. Again, none of these studies have explored the issues from the energy angle.

To our knowledge, most prior work on energy efficient indexing is for broadcast data [58]. The problem there is to be able to power down wireless network interfaces intelligently so that they do not need to unnecessarily consume energy listening in on conversations between a

base station and other mobile devices. With intelligent indexing, the device can turn on exactly when there is data of interest to it, thus saving energy. However, this problem is quite different from the one under investigation in this chapter where we are specifically looking at storage organization and query processing. Further, we are interested in queries that may be different for each user, rather than look at information that is needed by all users (broadcast data).

With the recent popularity of numerous mobile devices (especially PDAs), there has been a sudden plethora of applications that have been thrust on these devices. A popular one that several vendors seem to offer [41, 80] is a version of a road atlas (a simple spatial database application), allowing the user to get driving directions (shortest path problem), examine detailed map information (range queries), give details on a landmark/restaurant that the user points to (point queries), and so on. However, many of these applications have been developed in an ad hoc manner, and there is no published result so far on the energy consumption of these applications or the energy optimizations that have been performed. Further, several such offerings are very primitive in terms of the querying support that they support (for instance, proximity queries are not well supported). On the other hand, a detailed investigation of the energy consumption of storage organizations and querying can provide a much more systematic way of designing and implementing an energy-efficient SDBMS for resource-constrained environments. This can also suggest hardware enhancements to improve the performance and energy savings of such systems.

## 4.3 Spatial Structures Under Consideration

As was mentioned in the previous section, there are numerous spatial data organizations that have been proposed [102, 39] and exploring the energy behavior of all these structures is

well beyond the scope of this study. Rather, we select three previously proposed structures - PMR Quadtrees [54, 55], Packed R-Trees [68], and Buddy Trees [105] - that have been argued to perform relatively well for a range of datasets [39]. These structures are also representative examples from the design space of storage structures for spatial data. In Quadtrees, the index nodes at the same level have non-overlapping spatial extents, while R-trees and Buddy-Trees allow overlaps. Quadtrees are improvements over spatial partitioning techniques such as Grid Files, while R-trees are extensions of B-trees for spatial data. Buddy trees are representative of hashing based schemes using a tree structured directory. R-trees give more balanced structures than Quadtrees or Buddy trees.

As for the datasets, we consider line segments in a two dimensional space in this study. We believe that this does not significantly impact the main contributions of this work. Line segments represent an important class of datasets, especially in the road atlas applications for the mobile devices. Line segments (or polylines) can be used to represent streets, rivers, etc. Other related studies have also used line segment datasets [54, 55]. In all the structures, the line segments are sorted based on the Hilbert-order [45] of their centroids and kept in an array. The leaf nodes of the structures have pointers (index into the array) to the actual data items. As was mentioned in Section 4.1, we do not consider dynamic structures in this study, and assume that all the data items are pre-loaded into the memory-resident database (and do not change).

We consider three kinds of queries that have been identified [54, 55] as important operations for line segment databases:

- **Point Queries:** In these queries, the user is interested in finding out all line segments that intersect a given point. For instance, such an operation could be used to find out which streets meet at a given intersection.

- **Range Queries:** These are used to select all line segments intersecting with a specified rectangular window. Very often, the user wants to magnify a portion of the atlas for a closer examination, and this query can serve such a request.

- **Nearest Neighbor Queries:** These are proximity queries where the user is interested in finding the nearest line segment (street) from a given point (e.g. what is the closest street to a given landmark, subway station, etc.). This is the perpendicular distance to the line segment if the perpendicular intersects the segment, and is the distance to one of the end points (closest one) otherwise.

Range and Point queries are typically implemented using a *filtering step* where the possible candidates are first identified using their minimum bounding rectangles (MBRs). Each index node of the hierarchical spatial structures represents a rectangular region of the spatial extent that it covers, and is represented by the MBR of this region. The filtering step, that traverses the index structure, uses these MBRs to identify possible candidates. Subsequently, a *refinement step* is needed to perform the actual geometric operations on each short-listed data item to find the exact answers to the query. In structures (Quadtrees) that do not allow overlapping ranges between the index nodes at the same level, a line segment that spans more than one range needs to be replicated in all those ranges (we do not consider clipping based approaches that break a segment into multiple parts for each region that it falls in, and recombine/reconcile them in the refinement step). This does not need to be done for structures that allow overlapping ranges such as R-trees. As a result, part of the refinement step for Quadtrees involves duplicate elimination as well.

The Nearest Neighbor query is a little more complicated to implement for index structures, with different previous suggestions [54, 90, 98, 89]. For instance, [90] uses a progressively expanding (in size) range query centered around the query point till the first data item is found. Another possibility [54] is to actually go to that region of the index structure, and examine around this region in the structure instead of composing the searches as separate range queries. A more interesting, and perhaps more efficient, approach is studied in [98] that is the strategy used in this study. The search starts at the root node and examines the MBRs of its children. It orders these MBRs in terms of distances from the query point, and uses these distances to determine the recursive search order. In addition, it also uses these distances to prune the search when noticing that certain MBRs will definitely contain data items that are closer than those for the other children. The process is then recursively carried out for the candidate child nodes. This is a general technique that can be used for any of the considered hierarchical spatial access methods. The nearest neighbor query does not have separate filtering and refinement steps in our implementation.

### 4.3.1 PMR Quadtrees

The PMR Quadtree proposed in [54, 55] is a member of a family of data structures that adaptively sort the line segments into buckets of varying size. Line segments are inserted one-by-one into an initially empty block. When the block reaches its capacity, it is split into four blocks of equal size. A line segment needs to be inserted in all the blocks (at the leaf level) that intersect it, and each of those blocks need to be again checked for capacity and possibly split into four again. One of the properties of this structure is that a block is split only once, and never again (it is no longer at the leaf level).

In the end, all internal nodes have pointers (their MBRs are implicit) to four children, while the leaf node has pointers to the actual data items that fall within its region. A consequence of this structure is that a line segment could be contained in more than one block/subtree (all those that intersect it), which makes it necessary to check for duplicates in the final refinement. Also, the structure may not necessarily be balanced. However, there is the potential of searching fewer subtrees when responding to the query because their spatial coverages are disjoint. Depth-first search is used to traverse this hierarchical structure in our implementation.

### 4.3.2 Packed R-trees

The R-tree [48] has been proposed as an extension to the B-tree structure for handling multidimensional data. Many variants of the R-tree, such as $R+$-tree [106] and $R^*$-tree [82] have been studied. They differ in the algorithm that is used for insertion, specifically in splitting a node of the tree when its subtree is filled. They attempt to give better balanced (and efficient) trees by dynamically adapting to the insertion pattern/sequence. However, these structures can become inefficient when the database of spatial items is static (and known *a priori*). In such cases, one should use bulk-loading techniques rather than insert item by item to build the data structure. Roussopoulos and Leifker [99] use packed R-trees for such static databases to lower response times. Further, Kamel and Faloutsos [68] suggest using Hilbert value (a linearization technique for multidimensional space [45]) for sorting the data items before constructing the bulk-loaded R-tree. This is the structure that is evaluated in this chapter. Typically, such R-trees are built in a bottom-up fashion, level by level. After the line segments are sorted, for each line segment, starting from the first and going one after another, a pointer and its MBR are entered into an index node. When the number of pointers exceeds the node capacity, a new index node is

created. After all the lines are assigned pointers and MBRs, index nodes for the next higher level are created to point to the index nodes at the lower level, and the process continues recursively till we get a single root index node.

Along with each pointer, we keep track of the MBR of the area that is covered by that subtree. The R-tree structure allows MBRs of pointers to overlap, which actually helps keep it more balanced than a Quadtree. However, the downside to this is that a search has to traverse more possible paths in the hierarchical structure. Depth-first traversal is used to implement the queries.

### 4.3.3 Buddy-Trees

The Buddy-Tree [105] can be regarded as a compromise between the R-tree [48] and Grid File [53]. It is different from Grid Files in that it does not partition spatial regions that do not contain any data items. It is different from R-trees in that the spatial partition into which a data item can fall are pre-determined (similar to Quadtrees). The Buddy-Tree was originally proposed for points [105](which fall in exactly one region), and later extended [104] to work for other objects allowing for overlapping buckets/index nodes. In this study, the overlapping regions strategy proposed in [104] is used to handle line segments.

The construction of the Buddy-Tree can be briefly explained as follows. Data items are inserted into an initially empty bucket (that corresponds to the spatial extent of the dataset), till its capacity is reached. Subsequently, it cuts the space into two (along one of the dimensions), creates an index node with two pointers, each pointing to a different leaf node (containing the data items in its half of the split region). In case the split results in one of the regions being empty, then the algorithm splits the other region once again in the other dimension. These

actions are recursively repeated as leaf nodes get filled to their capacity. The dimension of the split is alternated at each level. Essentially, the centroid of the line segment is used to determine where the line segment falls in the tree structure. The MBRs for this bucket and the nodes going all the way up to the root are adjusted to account for the size/shape of this line segment (which can result in MBRs of sibling index nodes to overlap). Depth-first traversal is used to implement the queries.

## 4.4 Experimental Setup

All the above index structures and queries have been implemented and evaluated using the energy estimation framework and workloads discussed below.

### 4.4.1 Energy Estimation Framework

Energy consumption is the integral of the power consumed over operating time. There are *three sources* for power consumption in CMOS circuits that are widely used in current computing devices [101]. The first source is the *logic transitions* (e.g., going from logic value of zero to one) that occur at the internal nodes of a circuit. This causes power to be drawn for charging the capacitance associated with these nodes, and is called *switching power*. The switching power is also influenced by the difference in the actual supply voltages used to represent logic values one and zero. The smaller the difference, the smaller the switching power. *Short-circuit currents* that flow directly from the supply to the ground when the inputs to a circuit transition are the second cause of power consumption. Both switching and short-circuit components of power consumption are dependent on the transition activity of the inputs, and together constitute the

*dynamic power consumption*. The third source of consumption is the *leakage current* that flows even when the inputs do not change, and is called *static power consumption.*

While the static power consumption of a mobile device could be a dominant factor when it is in a standby or quiescent mode, its dynamic power consumption is much more significant when it is active [25]. Since this study is investigating the energy behaviors of different spatial index structures on a mobile device and they are relevant only when the mobile device is active, the dynamic power consumption naturally becomes the main focus of our investigation.

The dynamic energy consumed in a mobile device can be expressed as the sum of the energies consumed in different components such as the *processor datapath*, *caches*, *clock distribution network*, *buses*, and *main memory* (we are not considering disks since we are dealing with memory resident datasets). Other peripherals, such as the display, are not under consideration here since our focus is mainly on energy expended in query execution (it has also been shown that around 52% of the energy is expended by the components under consideration on some mobile devices [100]). The activity, and consequently the energy consumed by these components, is determined by the program executing on the system. The program can modify the number of transitions in the components (note that this affects switching power) by altering the input patterns, or reduce effective capacitance by reducing the absolute number of accesses to high-capacitance components (e.g. large off-chip memories). Note that transition frequency and effective switch capacitance are key contributors to the dynamic energy consumption, together with supply voltage.

Our energy estimation framework uses *SimplePower*, an architectural-level, cycle-accurate execution-driven energy simulator that is available in the public domain [120]. The architecture of the simulated system includes a single-issue five-stage pipelined integer datapath (instruction

fetch (IF), instruction decode/operand fetch (ID), execution (EXE), memory access (MEM), and write-back (WB) stages) , on-chip instruction (I) and data (D) caches, that is connected to an external (off-chip) memory. This architecture is a representative of the current commercial offerings in the PDA domain [120]. The instruction set architecture is a subset of the instruction set (the integer part) of *SimpleScalar*, which is a suite of publicly available tools to simulate modern microprocessors [23].

The major components of *SimplePower* are: SimplePower core, RTL power estimation interface, technology dependent switch capacitance tables, cache/bus simulator, and loader. The *SimplePower* core simulates the activities of all the functional units and calls the corresponding power estimation interfaces to find the switched capacitances. These interfaces can be configured to operate with energy tables based on different micron technologies. Transition sensitive, technology dependent switch capacitance tables are available for the different functional units such as adders, ALU, multipliers, shifter, register file, pipeline registers, and multiplexors. The *SimplePower* core continues the simulation until a predefined program *halt* instruction is fetched. Once the simulator fetches this instruction, it continues executing all the instructions left in the pipeline, and then dumps the energy statistics. SimplePower provides the total number of cycles in execution and the energy consumption in different system components (datapath, clock, cache, memory and buses) as explained below.

**Processor Datapath:** The energy consumed in the processor datapath is dependent on the number, types, and sequence of instructions executed. The instruction type determines the components in the datapath that are exercised while the number of instructions determines the duration of the activity. Whenever a component is exercised, there is a switching activity in that

particular component contributing to dynamic energy consumption. For example, when executing an integer addition instruction, energy is consumed in the instruction cache and instruction fetch logic in the first stage of the pipeline, in the register file when accessing the source operands in the decode stage, in the ALU when executing the operation, and, again, in the register file during write-back. It must be noted that energy is also consumed in the pipeline registers of the datapath, and that the components in the memory stage of the pipeline are not exercised by this instruction.

**Caches:** The cache simulator of *SimplePower* is interfaced with an analytical memory energy model based on [112]. The memory energy is divided into those consumed by the cache decoders, cache cell array, the buses between the cache and main memory, and the main memory. The components of the cache energy is computed using analytical energy formulations and is dependent on the number of cache accesses, number of misses, the cache configuration (e.g., associativity, capacity, line size), and the extent of utilizing energy-efficient implementation techniques (e.g., bitline segmentation, bitline isolation, pulsed wordline). Energy is expended in the row decoders when a particular cache line is selected for a read or write operation, in the lines that activate each cell (wordline) in a particular row of the cache line, in the bitlines when the values are written or read from the cells, in the sense amplifiers to amplify the values read, and finally, in the column decoders that select a part of the activated cache line. The energy consumed during reads and writes vary since the voltage swings in the bit lines vary during these operations (full swing for writes and partial swing for reads). Consequently, it is important to estimate the number of reads and writes individually. The energy consumed in the caches is

largely independent of the actual data accessed from the caches, and prior work has shown that the number of cache accesses is sufficient to model energy accurately [43].

**Memory:** The organization of the main memory is similar to that of the caches, with two main differences. First, the memory arrays have no tag comparison portion. Second, the basic cell for implementing memory storage (DRAM cells) is different from that used in on-chip caches (SRAM cells). Consequently, there is a difference in the energy consumed during read/write accesses. Typically, the energy cost of accessing memory is larger than that of on-chip caches because of the additional costs associated with off-chip packaging capacitances and also due to the energy consumed in refreshing the DRAM cells. The energy consumed in the memory can be again analytically modeled fairly accurately by capturing the number of accesses and the interval between the accesses obtained from a cycle accurate simulator [26].

**Buses:** The buses are used to communicate the addresses from the datapath to the caches and memories, and to transfer the data between these units. The energy consumption is quantified analytically based on the number of bus transactions, the width of the buses, the switching activity on these buses, and bus line capacitance. The switching activity here is assumed to have equal probability 50% since no exact information about it could be obtained.

**Clock Network:** The simulator also reports the on-chip clock energy using detailed models [31]. The components of the clock network that contribute to the energy consumption are the clock generation circuit (called PLL), the clock distribution buffers and wires, and the end-load on the clock network presented by the clocked components. The energy consumed in a single cycle depends on the parts of the clock network that are active. The PLL and the main clock

distribution circuitry are normally active every clock cycle during execution. However, the participation of the end-load varies based on the active components of the circuit as determined by the software executing on the system. For example, the clock to the caches are gated (disabled) when a cache miss is being serviced. Thus, actual execution and stall cycles can be used to calculate the clock energy. Thus the energy in the stall cycles do not manifest in the datapath component in our statistics, but appear in the clock network. If interested in further information on the energy models of the clock network, one can find it in [27].

The energy results from the simulator have been shown to be within 8% of those measured from a real system [120]. All results reported here are obtained using the parameters given in Table 4.1.

We also model a wireless network interface (that is capable of transmission bandwidths of up to 11 Mbps) which offers different operating modes - Sleep (0.02 watts), Idle (0.1 watts), Receive (0.165 watts) and Transmit (3.09 watts), based on its current functionality. This model has been drawn from [111]. We use this interface in one set of experiments in section 4.5.8, and is assumed to be absent otherwise.

### 4.4.2 Workloads

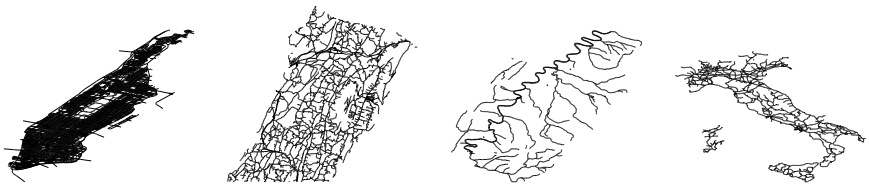
In our experiments, we use four line segment datasets as explained below: (a) **NYCS** contains 12355 streets of New York City, taking about 1.14 MB; (b) **PAFS** contains 16431 streets in Pennsylvania Fulton county, taking about 1MB; (c) **SVR** contains 5848 rivers from the Shenandoah valley, taking 106 KB; and (d) **IRR** contains 12338 railway tracks of Italy, taking 468KB. The first three datasets are taken from the Tiger Dataset [83] while the last is taken from the Digital Chart of the World [60].

| Parameter | Value |
|---|---|
| Supply Voltage | 3.3 V |
| Cache Sizes (each of I and D) | 8KB, 16KB, 32KB (32 bytes line size) |
| Associativity | Direct-Mapped (DM), 2-way, 4-way |
| Data Cache Hit Latency | 1 cycle |
| Memory Size | 8 MB |
| Memory Access Latency | 100 cycles |
| Per Access Energy for DM-Caches (in nJ) | 0.048 (8K), 0.082 (16K), 0.094 (32K) |
| Per Access Energy for Memory | 3.57 nJ |
| On-Chip Bus Transaction Energy | 0.069 nJ |
| Off-Chip Bus Transaction Energy | 6.9 nJ |
| Per Cycle Clock Energy | 0.18 nJ |
| Technology Parameter | 0.35 micron |

Table 4.1. Base configuration parameters used in the experiments.

The datasets are also representative of some of the SDBMS applications on mobile devices. NYCS and PAFS are typical of road atlas applications for navigation and locational information, the former is for a city and the latter for a rural county. SVR is a dataset that could be useful for hikers/environmentalists on the trails. Finally, IRR is from a different database and would be useful to find the nearest railway track, finding the identity of a station on a track, etc., with the queries that we are considering. A pictorial view of the four datasets is shown in Figure 4.1.

On these datasets, we use the results from 100 runs for each of the three kinds of queries (Point, Range and Nearest Neighbor). Each run uses a different set of query parameters. For the Point queries, we randomly pick one of the end points of line segments in the dataset to compose the query. For the Nearest Neighbor queries, we randomly place the point in the spatial extent in each of the runs. For the Range query, the size (between 0.01% and 1% of the spatial extent), aspect ratio (0.25 to 4) and location of the query windows is chosen randomly from the



(a) NYCS (b) PAFS (c) SVR (d) IRR

Fig. 4.1. Datasets

distribution of the dataset itself (i.e. a denser region is likely to have more query windows). The results presented are the sum total over all 100 runs.

| Index | Code Size | NYCS | PAFS | SVR | IRR |
|---|---|---|---|---|---|
| Quadtree | 39KB | 150KB | 183KB | 59KB | 133KB |
| R-tree | 35KB | 285KB | 378KB | 135KB | 285KB |
| BuddyTree | 38KB | 670KB | 989KB | 344KB | 732KB |

Table 4.2. Code Size and Storage Overheads for the Index Structures

The code sizes for the implementation of the index structures and the storage sizes of the index structures (not including the space taken by the dataset) are given in Table 4.2 for the chosen fan-outs (see Section 4.5.1). As far as the code size is concerned, the Quadtree code is a little larger because of the duplicate elimination code that is absent in the other two (the code for building the structures is not included in these sizes). Despite these minor differences, the code size is not very different across these structures. R-tree incurs more storage overheads because of its more balanced nature. Despite the packed R-tree algorithm that is used, some nodes could still be under-utilized. The property of the buddy tree which keeps index nodes that are not

entirely packed to capacity (could be much sparser than R-tree nodes), results in a much poorer space utilization compared to the other two structures.

As was mentioned earlier, we do not study the building costs for the structure since we are examining a static situation without dynamic insertions (and the storage structure is downloaded from a server on to the mobile device similar to how it is done in [80]). The chosen dataset sizes and their index overheads are also similar to some of the pocket atlas datasets (e.g. the New York City map that is available in the public domain for PocketStreets [80] running on Windows CE takes 865KB).

### 4.4.3 Metrics

We examine both the energy behavior as well as the performance profile for each execution. This helps us understand the trade-offs between the two if any, and also helps us explain the energy consumption based on the performance results.

For the energy behavior, we profile the consumption (in joules) by each of the hardware components - processor datapath, I-cache, D-cache, Memory, Buses (between cache and memory), and clock network. For the performance profile, we give the breakdown of the cycles spent by the processor performing useful work, and also when stalling on I-cache and D-cache misses.

These profiles are given for each of the query executions on each dataset using the different index structures, and compared with the Brute-Force approach. The profiles are also separately given for the Filtering and Refinement steps, to understand where the overheads are incurred from the software perspective. From the hardware perspective, the impact of different cache organizations on energy and performance behavior is also studied.

Energy consumption, execution cycles and the product of these two (denoted as energy*cycles) are the key metrics that are used for our comparison. Energy*cycles is actually a synonym to the metric energy*delay that has been traditionally used in the low power design community as it considers both the battery lifetime and the performance of a mobile device. Since the delay is defined as cycles*cycle time and the cycle time is kept constant in our study, energy*cycles and energy*delay can be used exchangeably here. For the consistency, however, we will use energy*cycles henceforth. We also present energy/cycles values capturing the average energy per unit time (power), which is important for packaging and thermal considerations.

Many of the results and trends are common across the datasets. As a result, *the graphs show the behavior averaged over all the datasets*. Whenever there is a dataset influence, the effects are explicitly mentioned in the discussion.

## 4.5 Experimental Results

### 4.5.1 Impact of Fan-Out

One of the important considerations for each index structure is the fan-out issue. For R-tree and Buddy-Tree, this corresponds to the number of (MBR, ptr) pairs at all levels of the hierarchical structure, with each such entry taking 20 bytes. In the Quadtree, the fan-out of the internal nodes is fixed at 4 entries (taking 80 bytes totally) as per the definition of the structure, and the only choice is for the number of pointers to maintain at the leaf level for the lines falling within this bucket (as suggested in [54]). Apart from the nature of the dataset itself, several factors govern the choice of a fanout. In a disk-based storage structure, the disk access times have a large influence on the choice of the fan-out, and it will be interesting to see how memory

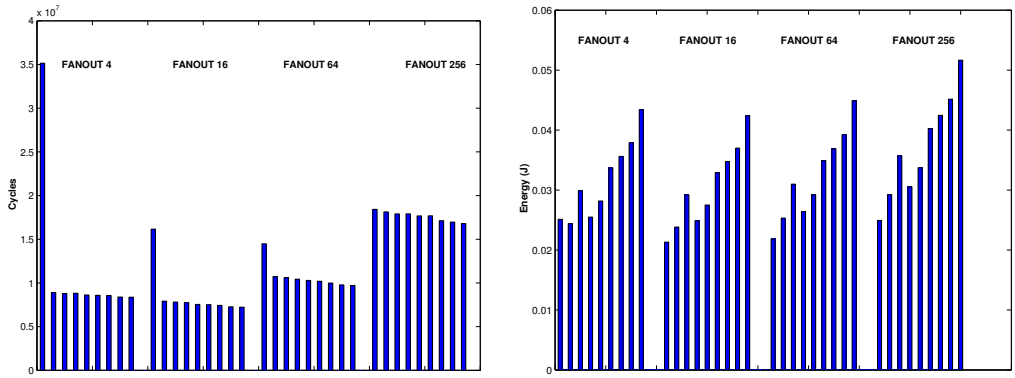



Fig. 4.2. Impact of Fan Out of R-tree on Total Cycles (left) and Energy (right) for Range Queries with PAFS. For each configuration, the nine bars from left to right correspond to cache configurations of (8K,DM), (8K,2way), (8K,4way), (16K,DM), (16K,2way), (16K,4way), (32K,DM), (32K,2way), (32K,4way).

resident datasets affect this issue. We have varied the fan-out of the different structures, and collected both performance and energy profiles for the different datasets. Figure 4.2 shows a representative result (results are similar for other datasets), illustrating the impact of fan-out with range queries for R-trees on PAFS. As fan-out increases, the depth of the tree goes down, thereby improving performance initially. On the other hand, the number of paths to be searched and the number of comparisons at each index node may increase, which worsens performance (in terms of CPU cycles). With these two contrasting factors, the best fan-out that we observe is at 16 for the R-trees as is shown in the figure (except for 8K direct mapped caches), which yields a node size of 320 bytes. We also observed a similar behavior for the fan-out of the leaf nodes of the Quadtree, where the ideal leaf node size again turned out to be 320 bytes (80 pointers). A fan-out of 16 was observed to give the best performance for the Buddy-Tree as well. These observations hold across cache sizes and associativities as can be seen in the graphs.

Another interesting observation is that the fan-out has a similar effect on energy consumption as performance, suggesting that using one of these metrics to optimize the fan-out may suffice in practice for the overall energy*cycles savings. It should be noted that each query can demand a different fan-out, and it is difficult to predetermine this value unless we have a good idea of the workload imposed on these structures. Since range queries are usually much more prevalent, we have chosen a fanout for each structure that is optimized for the range queries, and use this ideal fanout for all our experiments (regardless of the query).

### 4.5.2 Results from Brute Force Method

Before we get to the index structures, we present results for the Brute-Force method separately in Figure 4.3 since its performance is much worse (and including its results in the same graph as the index structures increases the scale significantly, making it difficult to see any noticeable differences between those structures).

For the range and nearest neighbor queries, both the cycles and energy consumption are an order of magnitude higher (if not more) than those with the index structures (compare with Figures 4.6 and 4.8). In these queries, the predicate checking for every data item is significantly higher than that incurred for a point query. Since the overall execution energy is a function of both the number of data items checked and the cost of the check, range and nearest neighbor queries are much more expensive in Brute Force compared to the point queries. Since the brute force approach is significantly worse from both performance and energy viewpoints, we do not consider it any further in this study.

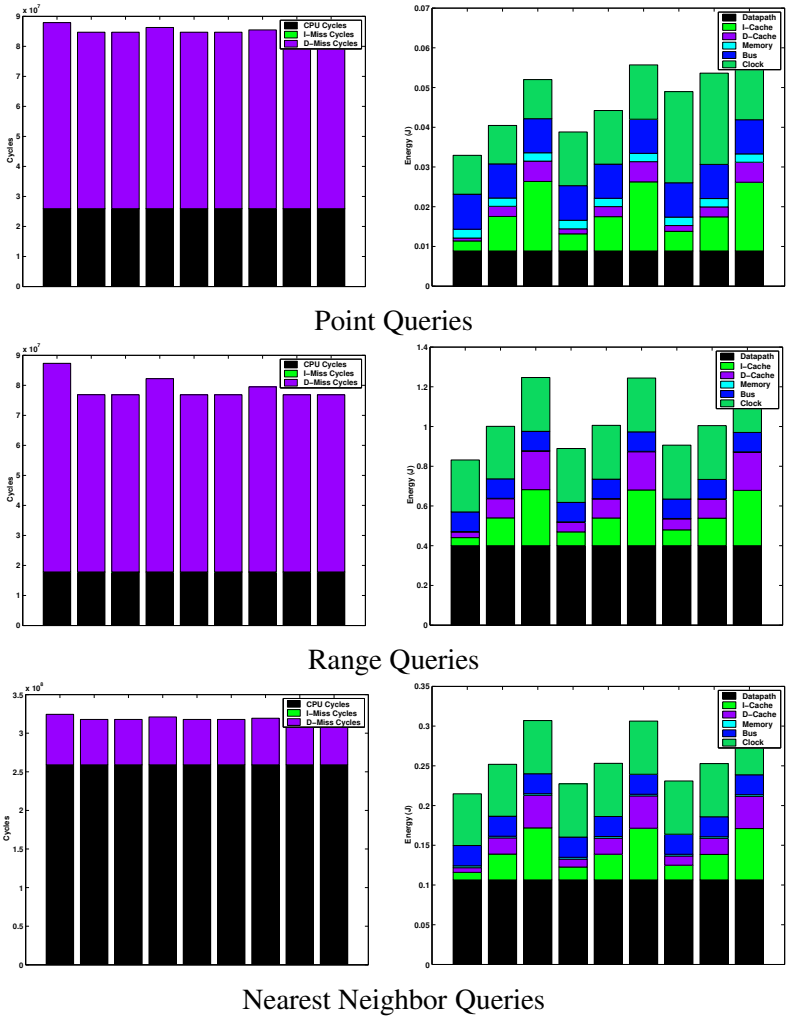
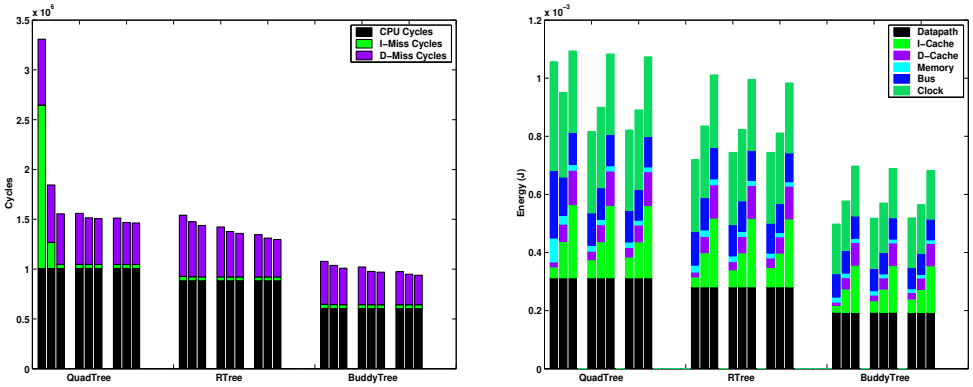
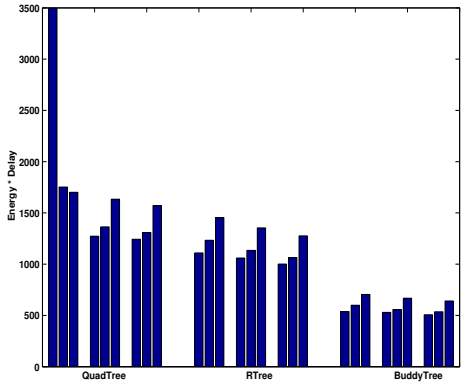



Fig. 4.3. Performance and Energy of Brute Force method for the three queries averaged over the four datasets. In each graph, the nine bars from left to right correspond to cache configurations of (8K,DM), (8K,2way), (8K,4way), (16K,DM), (16K,2way), (16K,4way), (32K,DM), (32K,2way), (32K,4way)




(a) Average Cycles (b) Average Energy




(c) Average Energy * Cycles

Fig. 4.4. Comparison of Index Structures for Point Queries. The nine bars from left to right for an index correspond to cache configurations (cache size, cache associativity) of (8K,DM), (8K,2way), (8K,4way),( 16K,DM), (16K,2way), (16K,4way), (32K,DM), (32K,2way), (32K,4way).

### 4.5.3 Results for Point Queries

Figure 4.4 shows the performance, energy and energy*cycles profiles for the point queries with the different schemes averaged over the four datasets.

Examining the execution cycles graph, we see that Quadtree has a higher processor cycle count compared to R-tree. Since the Quadtree does not allow overlaps (and a point query does not have a high probability of falling in more than one bucket), the filtering step on the Quadtree to find the candidate leaves is relatively fast. This is evidenced by the execution profile for the filtering step in Figure 4.5 which shows the cycles for Quadtree are significantly lower than those for the R-tree. However, the refinement step graph in the same figure shows that Quadtree is much more time consuming, placing the sum of these two steps slightly in favor of the R-tree. It should be noted that the overhead for Quadtree in the refinement step is mainly due to the larger number data items (even though the leaf nodes in the two structures have the same size - 320 bytes, this corresponds to 80 data items in the Quadtree which stores only pointers and to 16 data items in the R-tree which stores pointers and MBRs) that the Quadtree has to deal with in the refinement step. The reason we chose the 80 data items fanout in our implementation was because it gives good performance for range queries (as mentioned earlier). Though the graphs are not explicitly shown, we would like to point out that a 16 data item fanout (same as the R-tree leaf node), does indeed give better performance for point queries with Quadtrees, cutting down the overhead of the refinement step, thus making the Quadtree performance similar (or even slightly better in some cases) to the R-tree. Between the R-tree and Buddy-Tree, we find the latter giving better performance. This is mainly due to the splitting criteria for a node, where

the Buddy-Tree partitions based on spatial locations while the packed R-tree just uses Hilbert order groupings.

We find that Quadtree has better data locality than R-tree, which can be explained with the higher internal node fanout and overlapping buckets in the latter. With overlapping buckets, R-tree may entail searching more paths even with a point query (while the Quadtree is more focussed). Since the fanout of the internal nodes is higher, there is more scope for eviction of data items from the cache that may be needed again. This is evident by examining the D-misses in the filtering step in Figure 4.5. The Buddy-Tree locality falls between these two.

It is difficult to comment on the I-cache locality behavior of the different codes without clearly understanding where the called procedures fall within the code segment and how they reference each other (for conflicts). In general, we find that 8K caches direct-mapped caches are not a good idea for the I-cache (which is true in later queries as well). Most of the penalties are reduced with a 16K 2-way I-cache.

From the energy perspective, we find that the components' energies have a strong dependence on the time spent in the cache miss cycles (stalled cycles) vs. CPU cycles (non-stalled cycles). The datapath (and the resulting clock) contribution to the overall energy consumption, for instance, is considerable reflecting the importance of the CPU cycles spent in the instruction execution shown in the performance graph. Overall, the differences between the three indexes in terms of energy consumption reflect the same observations that were made between them from the performance perspective. As a result, for this set of experiments the energy and performance results go hand-in-hand to a large extent. The only exception to note is the I-cache and D-cache energy consumption changes as we change the cache configuration. With improved (larger size or better associativity), the miss rate is expected to go down, but at the same time

energy cost incurred per access goes up. These two factors can help us decide on a good energy-cycles conscious cache configuration (captured by the energy*cycles values in Figure 4.4(c)). With 16K and 32K (I and D) caches, most of the locality in instruction and data references is captured well by these configurations, and the energy increase with associativity is more significant. As a result, with these cache sizes, it would be better to have a direct-map structure from the energy*cycles perspective (see Figure 4.4(c)). With 8K I and D caches, we find that the performance penalties due to conflict misses are quite severe, preferring a higher associativity from the energy*cycles perspective. For the Quadtree (especially due to its high I-cache misses), an associativity of 4 is needed, while R-tree and Buddy-Tree give the best energy*cycles for a direct-mapped cache.

Despite the depiction of lower cycles, energy and energy*cycles for R-tree over Quadtree in Figure 4.4(c). we would like to reiterate that these differences are mainly due to the differences in the chosen fanouts. We believe that these two structures are more or less comparable in terms of these metrics if we fine-tune the fan-out values at the leaf level for the Quadtree to suit this query. In terms of all these metrics, we find that Buddy-Tree delivers the best results for point queries.

### 4.5.4 Results for Range Queries

Figure 4.6 shows the performance, energy and energy*cycles profiles for the range queries with different schemes averaged over the four datasets.

Rather than repeat all the observations that are similar to those for the point query, we would like to point out the differences. The first noticeable difference is that the Quadtree performs much worse than the R-tree and Buddy-Tree in terms of both performance and energy

(despite having chosen a fan-out that gives the best performance for the Quadtree). Compared to the point query, range queries have higher likelihood of covering spatial extents of more than one leaf node. As a result, the searches are not that focussed any more on a Quadtree, and more than one path may need to be searched. Second, since the region boundaries of a Quadtree's index nodes are pre-determined and are not adapted to a dataset's vagaries, there is the scope for traversing more paths in a Quadtree compared to the R-tree (this can be seen by the differences in their performance for the filtering step in Figure 4.7). Finally, non-overlapping boundaries of index nodes can result in a data item being replicated, and duplication elimination is time-consuming for the Quad-tree (as seen by the performance for the refinement step in Figure 4.7). In the filtering step, all candidates are inserted into a list. The refinement step for the Quadtree first sorts this list to remove duplicates, and then performs an item-by-item comparison. These overheads materialize in the refinement step performance, and is also the reason why Quadtree has slightly higher data misses for the refinement step compared to the R-tree and Buddy-Tree.

In general, we find that range queries are more processor intensive than point queries, with a smaller fraction of the time spent stalling on cache misses for both index structures. The significance of the refinement step which has good locality (due to sequentially searching a list for exact matches) in the overall performance picture is the main reason for this behavior (see Figure 4.7).

We find that performance is the main factor governing these schemes when we examine them from the energy and energy-cycles perspectives. Higher number of instruction executions imply a larger datapath energy and clock energy. At the same time, each instruction fetch references the I-cache, incurring an energy cost (even when it is a hit). For instance, we can see a

noticeable difference in the I-cache energy costs between the Quadtree and the other two in the refinement step which is a time-consuming fraction.

In the point queries, we could see both performance and energy impact of cache configurations playing significant roles when determining a good operating point in both R-trees and Quadtrees. In the range queries, we find that the energy*cycles metric obeys the performance (cycles) trend in nearly all cases (except for the Buddy-Tree with 8K 4-way caches). Both R-trees and Buddy-Trees do a good job for this query along all three perspectives - performance, energy, and energy-cycles, with Buddy-Trees having a slight edge.

### 4.5.5 Results for Nearest Neighbor Queries

Figure 4.8 shows the performance, energy, and energy*cycles profiles for the nearest neighbor queries with the different schems averaged over the four datasets.

The nearest neighbor query presents an entirely different picture from what we have observed in the previous two queries. Compared to the previous two, the results show that cache misses dominate the execution time, and processor cycles are a much smaller fraction in many of the datapoints. In the first place, there is no separate refinement step for this query as was mentioned earlier, with data items examined closely when they are first encountered. Even with the earlier queries we found that misses are more significant in the filtering step (tree traversals) than in the refinement step. Further, the working set sizes for implementing the nearest neighbor algorithm (explained in Section 4.3) are higher than for point/range queries. Specifically, when traversing a subtree, closest distances need to be calculated for all children and they need to be sorted and pruned, before recursively traversing them. Point/Range queries can examine children one at a time, moving to the next after traversing the subtree under the previous child.

These operations make the nearest neighbor query much more dependent on miss penalties. The number of D-misses is closely related to the number of children (fanout of internal nodes), which also explains why R-tree has higher D-cache misses compared to Quadtree and Buddy-Tree. The I-misses show a reverse behavior with R-trees having better code locality (except for the 8K DM case) than Quad-trees. Since R-tree has a larger fanout and lower depth, the sorting/pruning operations and overheads are amortized over a larger number of children at a time, while the Quadtree and Buddy-Tree may keep switching between traversal and pruning more often. Overall, from the performance viewpoint, we find that R-tree does the best except for the 8K DM cache. Of the other two, the Quadtree outperforms the Buddy-Tree in many cases.

While there was not a noticeable difference in the relative performance of the schemes across the datasets for the previous two queries, we would like to mentioned that there is a difference between the datasets for this query with the Buddy-Tree structure (there was not a significant effect on the other two indexes). In datasets that are much more clustered (NYCS and IRR), the Buddy-Tree incurred more processor cycles than the others since it does not do as good a job as the R-tree (or even the Quadtree) in balancing the hierarchical structure to reduce the number of levels. For the other two datasets, its performance becomes comparable to the R-tree.

The most interesting observation with this query (compare Figures 4.8(a) and (b)) is that **better performance does not necessarily imply better energy** (except in 8K DM). R-tree takes fewer cycles to service the query, while Quadtree takes lower energy (with Buddy-Tree energy falling in between). The reason for this behavior can be explained as follows. R-tree incurs much lower CPU cycles than the Quadtree, but incurs higher cache misses. Miss penalties

(which require crossing pin boundaries and bus to get to main memory) translate to much more overheads in terms of energy (*off-chip energy*) compared to performance. While the additional miss cycles are not significant enough to put R-tree overall cycles higher than Quadtree, the miss energy (in D-cache, Bus and memory) overhead compensates for any savings in the lower datapath energy (e.g. compare the datapath, D-cache, bus and memory energy components for the R-tree with that for the Quadtree for the 32K 2-way caches). The Buddy-Tree energy falls between that for R-tree and Quadtree in most cases.

A consequence of the differences between the energy and performance behavior for this query is the interesting observation in the resulting energy*cycles metric shown in Figure 4.8(c). This captures the facets of whether the improvement in performance warrants the additional energy that is expended. The results show that *even though R-tree is better in terms of performance, Quadtree (which is better in terms of energy consumption) may be a better alternative from the energy*cycles perspective* (i.e. the performance benefits for the R-tree come at a much higher energy cost that it may not be as attractive in energy-constrained environments) in most of the better cache configurations. The energy*cycles of Buddy-Tree falls in between these two in many cases.

### 4.5.6 Examining Datapath Power

It is important to also take note of the *power* consumption in specific system components so as to not exceed a specified limit. This is important for uniform heat dissipation across a chip, since increasing the power consumption in one particular component can create a thermal hot-spot in the chip [121]. Here we focus on the datapath power. Datapath power is presented as energy consumed per cycle (power in watts can be obtained by accounting for cycle time).

| Index | Query | Phase | Energy/Cycles |
|---|---|---|---|
| QuadTree | Range Query | Filtering Step | 31.88 |
| | | Refinement Step | 34.99 |
| QuadTree | Point Query | Filtering Step | 31.31 |
| | | Refinement Step | 30.81 |
| QuadTree | Nearest Neighbor Query | - | 37.82 |
| RTree | Range Query | Filtering Step | 32.22 |
| | | Refinement Step | 35.86 |
| RTree | Point Query | Filtering Step | 31.31 |
| | | Refinement Step | 31.33 |
| RTree | Nearest Neighbor | - | 38.36 |
| BuddyTree | Range Query | Filtering Step | 32.29 |
| | | Refinement Step | 35.71 |
| BuddyTree | Point Query | Filtering Step | 32.19 |
| | | Refinement Step | 31.97 |
| BuddyTree | Nearest Neighbor Query | - | 38.16 |

Table 4.3. Influence of Indexing Scheme and Query Type on Datapath Power Consumption (nJ/cycles) for PAFS

It should be noted that the energy consumption and number of execution cycles can exhibit totally different behavior. For instance, two runs can complete within the same number of cycles, but, depending on the hardware components exercised during these cycles, can result in very different energy consumptions. Power consumption (energy per cycle) shows the average complexity of computations performed within a cycle and Table 4.3 gives the datapath power for PAFS. We observe that the nearest neighbor query is by far the most power consuming query independent of index structure. This is mainly due to the complexity of the geometric operations (stressing energy expensive units such as the multiplier) that are performed when examining each data item.

The relative power consumption costs of filtering and refinement depend on the type of the query as well. For example, in the Buddytree, with range queries, the refinement step is 10% more power consuming than the filtering step. In contrast, with point queries, the refinement step is 1% less power consuming than the filtering step. Again, the geometric operations for testing whether a point is one of the end-points of a line segment (during refinement for the point query), requires only simple comparisons and is less power consuming than testing the same with a MBR in the internal nodes (during filtering). For the range query, testing whether a line intersects the rectangular window (refinement) is more complex than testing whether a MBR interests the window (filtering).

Further, we can observe that *optimizing for energy and optimizing for power may result in conflicting choices.* For instance, the refinement step using point queries for the R-trees is more power efficient than that of the Buddy-Tree. But the situation is exactly the opposite in terms of datapath energy consumption (63 $\mu$J compared to 61 $\mu$J for Buddy-Tree).

So far, we have not carefully examined the impact of the dataset itself, and power dissipation is one issue where the differences are brought out. Table 4.4 shows the impact of differences in dataset on power consumption. We observe that the datapath power consumption varies as much as 3.5% due to the variation in datasets. In comparison, the impact of indexing schemes themselves for the nearest neighbor query is as much as 2.6%.

| Index | PAFS | IRR | NYCS | SVR |
|---|---|---|---|---|
| BuddyTree | 38.16 | 38.83 | 37.98 | 38.71 |
| Quadtree | 37.82 | 38.60 | 37.28 | 37.79 |
| RTree | 38.36 | 38.77 | 37.95 | 38.34 |

Table 4.4. Impact of Datasets on Datapath Energy/Cycles (nJ/cycles) for Nearest Neighbor Query.

### 4.5.7 Impact of Architectural Innovations and Technological Trends

The previous experimental results have examined current architectures and technologies to evaluate the pros and cons of the indexing schemes. It is also important to consider the impact of technological trends and innovations on the spatial access methods, especially since the hardware capabilities of the mobile devices are constantly changing/improving. We specifically consider two approaches here that both focus on optimizations of the memory system.

#### 4.5.7.1 Energy-efficient Cache Architectures

Uniformly, we find in the previous results that the cache energy is a significant consumer of the overall system energy (even over 50% in the nearest neighbor queries). It is not only on misses that energy is expended, but on all accesses including hits (which are very frequent). Further, these SDBMS workloads are data intensive, and are thus motivating factors behind energy-efficient cache design. While this issue is well-beyond the scope of our study, we would like to briefly touch upon the possibilities and impact of energy-efficient cache design.

A common trend in energy-efficient hardware design is the partitioning of components into smaller units, and selectively activating the unit that is needed. This approach reduces the energy consumption per access, since the smaller unit takes less power during the activation. Such an optimization can be applied to the I and D caches as well, wherein a single monolithic cache can be partitioned into several smaller ones (subcaches) and selectively activating one of them. One simple way of storing data in a cache partitioned into two is based on whether the referenced data exhibits spatial or temporal locality [44]. The downside to this approach is to be able to selectively activate the subcache where the current data resides. For the I-cache, simple prediction strategies like Most Recently Used (MRU) subcache can give good performance, since instruction references usually have good locality. The data references on the other hand can be difficult to predict, especially for the database workloads considered here that have very dynamic reference patterns when traversing hierarchical structures. When the prediction fails, a performance (since other subcaches need to be searched subsequently) and energy penalty is incurred. There are also set associative cache organizations where non-accessed ways within a set can be disabled by predicting the way that is being referenced (called way prediction [59]).

To explore the feasibility and benefits of such energy-efficient subcache structures we conducted a simple experiment where we fed the D-cache references to: (a) a single monolithic 32 KB DM cache (DM); (b) a single 32 KB 2-way set-associative cache using way-prediction to determine the next way within the set for selective activation (*Way-Prediction*); and (c) two subcaches (each of 16 KB) holding spatial and temporal data respectively, using access history to predict the subcache to be accessed next (*Subcaching*). Parameters used in this experiment such as the way-predictor accuracy and energy per access to sub-caches are taken from a simulation study in [69]. Table 4.5 shows the resulting energy, access time penalty (cycles) and energy*cycles with these optimizations for the range query using Quadtrees on the PAFS dataset (values are normalized with respect to DM).

| | Energy | Latency | Energy*Cycles |
|---|---|---|---|
| Way-Prediction | 0.7 | 1.15 | 0.8 |
| Subcaching | 0.4 | 0.8 | 0.32 |

Table 4.5. Impact of Cache Optimizations on Memory system energy and latency for Range Query using Quadtrees with PAFS normalized with respect to DM for D-Cache

The reason for this experiment is not to evaluate the subcache designs in detail or propose new strategies. Rather, we are only examining whether such a strategy would work for these SDBMS workloads with dynamic reference patterns, and if so how it would affect the relative performance of the indexing mechanisms. We find that there is definitely an energy benefit from these optimizations, and in some cases the latency can also be improved. We find that the overall energy*cycles from such structures can give us as much as 68% energy*cycles savings

in the memory system energy stressing the importance of energy efficient caches. If partitioned caches are employed, then the resulting executions will favor those schemes with a large number of I-cache and D-cache hits. Specifically, this will help Quadtrees which has more memory references in general.

#### 4.5.7.2 On-chip Main Memory (eDRAM)

While the previous technology provides energy optimizations for hits, it is important to study the technological trends affecting energy and performance on cache misses. For instance, embedded DRAM(eDRAM) is one such technology [113] where a portion of the main memory can be moved on-chip. This can significantly reduce miss penalty apart from reducing energy consumed by off-chip buses (transactions going out of the chip are expensive). To study how the indexing mechanisms would compare with eDRAMs of the future, we conduct an additional experiment. Using the method discussed in [27], we derive the values of eDRAMs energy consumption by scaling down the DRAM values by a factor of 10, and this factor is obtained from the comparison between an actual eDRAM implementation - M32RD and a conventional DRAM [123]. The miss latency is also cut down to 10 cycles from 100. The results of this experiment are shown in Table 4.6.

We observe that the eDRAM significantly improves the cycles, energy and consequently the energy*cycles of all indexing structures. More importantly, the effect is more pronounced for executions that incur more misses, and for this query it is the R-tree. As a result, the benefits of the R-tree are amplified in terms of energy*cycles for the (8K,2-way) execution, where the R-tree was already doing better without a eDRAM architecture. In the (16K,2-way) and (32K,2-way) executions, the eDRAM architecture makes the R-tree a better alternative from the energy*cycles

viewpoint (though it still consumes higher energy than Quadtree), even though it was worse than the Quadtree for the same metric without the eDRAM. Since we find that R-tree in general incurs more misses across the queries, eDRAM technologies are expected to benefit R-trees more than the other two structures.

### 4.5.8 Offloading Work to the Server

Throughout this study, we have assumed that both the dataset and index are completely resident in memory and the entire task of query processing is performed locally at the mobile client. Sometimes, as pointed out earlier, a wireless network and a resource-rich server may be available, and in such cases it would be interesting to see whether it makes sense to offload the query to the server to save on energy and cycles. The mobile device (the client) communicates with the server via a wireless Network Interface Card (NIC). Previous studies have shown that the NIC is a significant power consumer [40] which can offset benefits provided by a server. The trade-offs depend on factors such as the query-type, computing power of the client and the server, network bandwidth etc. In this section, we briefly motivate how work partitioning can provide both energy and performance benefits. A more detailed evaluation is conducted in [47].

Since it would require much more work to offset communication costs, we consider a larger dataset here, specifically we have extended PAFS to include the streets of Franklin, Bedford, and Huntingdon counties of Pennsylvania, making the dataset approximately 10.06 MB in size (client memory availability is increased to 32 MB and with 16 KB I-cache and 8 KB D-cache, both 4-way set associative). We show results with the packed R-tree that takes about 3.56 MB for this dataset. The server is assumed to be a 4-issue superscalar processor clocked at 1 GHz, with adequate memory to hold all of the dataset and index considered in its memory,

and the client is clocked at one-eighth the rate of the server (i.e., 125 MHz) that is comparable to what is found today in commerical offerings such as the StrongARM SA-1110 [61]. We consider network bandwidths of 2, 4, 6, 8, 11 Mbps (which is in the range of what is available or expected to be soon available in commercial offerings [30]).

There are several ways of partitioning the task of query-processing between a client and server. The query can be executed fully on the client, without any involvement from the server. This strategy has been the main focus of our study. On the other hand, the query can be sent to the server, which in turn can execute it completely and send the results back to the client. Figure 4.9 presents the results comparing these two schemes. For each bandwidth, the bar on the left shows the result when the data is not available on the client whereas the one on the right is for the case when the data is available locally as well. In the latter case, the server does not have to send back the data items that satisfy the query; instead, only the ids of the items need to be returned, thereby reducing the transfer-size of the message. Reducing the transfer-size reduces the amount of time for which the NIC needs to be active, thereby reducing the network transfer-time and also the energy consumption. It is interesting to note that the *gains in the performance and energy show different operating-points over the range of chosen bandwidths*. When the data is kept locally, work partitioning outperforms the fully-local case even at 2 Mbps, though it takes over 6 Mbps before it gets more energy-efficient. This happens because the energy cost of using the NIC is much more than the performance cost, and higher bandwidths are required to offset this difference.

There are other options for partitioning the work between the client and server, such as positioning the filtering at one end and the refinement at the other, and the trade-offs between those options are studied in [47].

## 4.6 Discussion

Despite the relatively small size of the datasets (to fit in the main memory or resource-constrained devices) we find that it is imperative to provide an index-based spatial access method to answer the three considered queries. Performance penalties of brute-force approaches are so significant (despite not incurring storage overheads needed to maintain index structures), and have a direct consequence on energy costs as well. If storage space overhead is a major concern for the resource-constrained environments, Quadtree is a better alternative than the other two structures (see Table 4.2). Of the other two, the packed R-tree makes better utilization of the space taken by its index nodes (see Table 4.2).

Between the three index structures, we find no clear winner across all queries and criteria that have been studied. Table 4.7 summarizes some of the observations that have been made in the earlier sections. It ranks the schemes (from 1 to 3) based on their relative merits for the performance, energy, and energy-cycles criteria, and a list of observations follow:

- For the point queries, we find the buddy tree giving better performance while incurring a lower energy cost. Consequently, it has the lowest energy*cycles values of the three. Between the other two, the differences are not very prominent, especially if we can tune their fan-outs for this query.

- With range queries, both R-trees and Buddy trees are giving good performance, energy savings and energy*cycles values. Quadtree is worse than these with queries needing to process more data for refinement.

- While performance largely dictates energy costs for the point and range queries, this study has shown that these criteria do not always go hand-in-hand. There could be circumstances

when a scheme giving the performance can incur the highest energy cost. This was observed with the nearest neighbor query where R-tree was giving the best performance but incurs the highest energy. Quadtrees turn out to be better from the energy or energy*cycles perspective for this query.

- Power dissipation is also another important consideration to keep the packaging and cooling costs low. We observed how filtering and refinement steps dissipate different amounts of power, with the nature of the query playing an important role on which phase is more significant. While the energy and performance trends for the four datasets did not appear very different, on a closer examination we find that the dataset has as much as an impact on power dissipation as the index structure itself.

- The results show that index-based query executions on spatial databases exercise the memory system considerably. A similar result has been noted recently in [3] where misses have been found to constitute around 40% of the execution time for memory-resident relational databases. The energy perspective shows that it is not only important to optimize miss behavior (by lowering number of misses, or by reducing energy consumption during misses), it is crucial to optimize energy consumption of hits as well (or even reduce the number of memory references). Energy consumption of caches plays an even more dominant role than its performance impact. While improving the caches (in terms of size and associativity) can reduce the miss behavior, the access costs increase. Consequently, we find that 16K 2-way associative caches are a good compromise between performance and energy for these workloads.

- To investigate the influence of evolving architectural enhancements and technological trends that can help reduce cache hit and cache miss energy consumption, we examined the impact of partitioning caches and embedding main memory on-chip. Partitioned caches for energy efficiency are likely to benefit Quadtrees more than the others, since they tend to have better locality during filtering and refinement compared to R-trees or Buddy trees. On the other hand, embedding the main memory on-chip favors R-trees which in general has poorer locality than the other two.

- Off-loading the work to a resource-rich server using a wireless medium is a good option in certain demanding queries (range queries) when we can get reasonable communication bandwidths. Energy and performance, again have different implications on when you need to perform such off-loading. Energy savings requires a higher communication bandwidth than that for performance savings.

These observations can help a designer customize a SDBMS for a given target resource-constrained environment, fine-tune the implementation to dynamically adapt for changing energy and performance criteria, and to even provide guidelines on incorporating architectural enhancements that can help meet energy-performance criteria in a more effective manner.

## 4.7 Chapter Summary

The growth in mobile computing has made mobile databases one of the most prominent segments of embedded database market [84, 86]. The market for embedded databases is expected to grow about 12% annually to 705 million dollars in 2003. Many of these applications are targeted for automotive and handheld devices which are likely to hold, access and process

spatial data. With the resource-constraints imposed on these embedded environments, energy and limited memory designs, take center-stage together with performance. This chapter has presented the first indepth examination of memory-resident spatial access methods for three index structure (Quadtrees, R-trees and Buddy Trees) from the energy, performance and energy-cycles perspectives. By doing so, we have identified the key issues affecting both energy and performance, at the algorithmic and architectural levels. It has taught us several important lessons including the fact that optimizing performance does not necessarily optimize energy and could in fact aggravate power dissipation. Since the target environments may have different (storage) capacities, processing power, and resource-constraints, the results from this work will be helpful to select and tailor the spatial access methods for designing mobile applications operating in diverse conditions. This exploration can in turn provide insight on new problems for research on embedded and spatial databases, accelerating their deployment on numerous mobile devices.

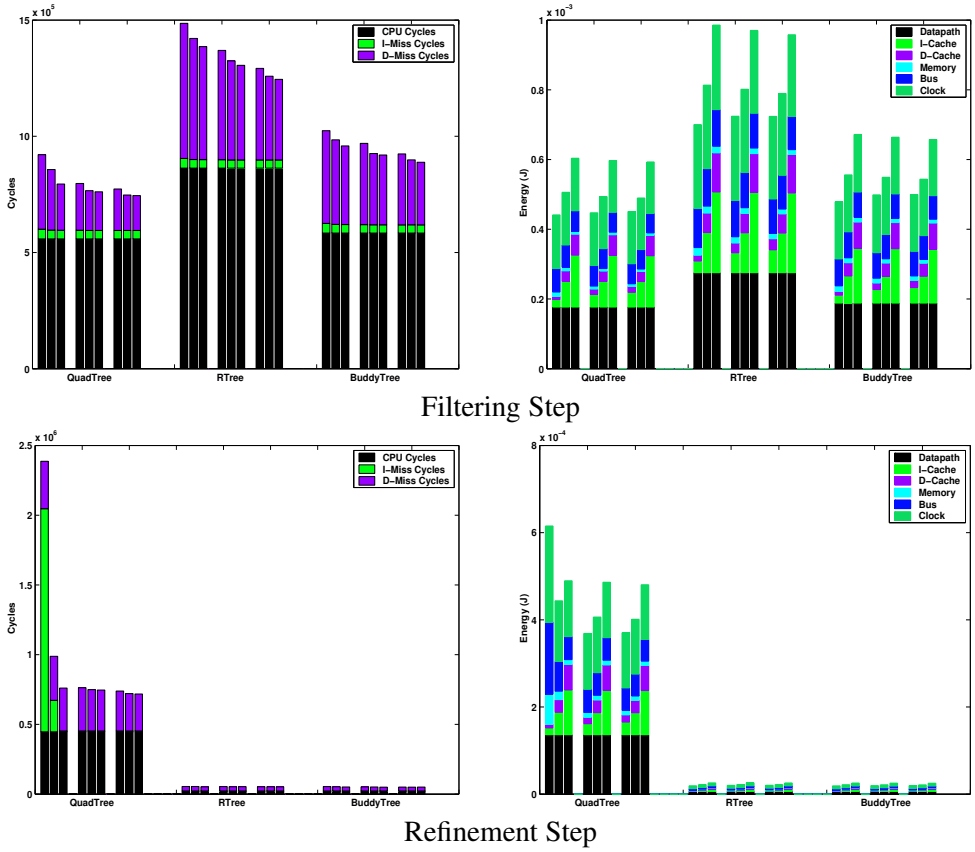
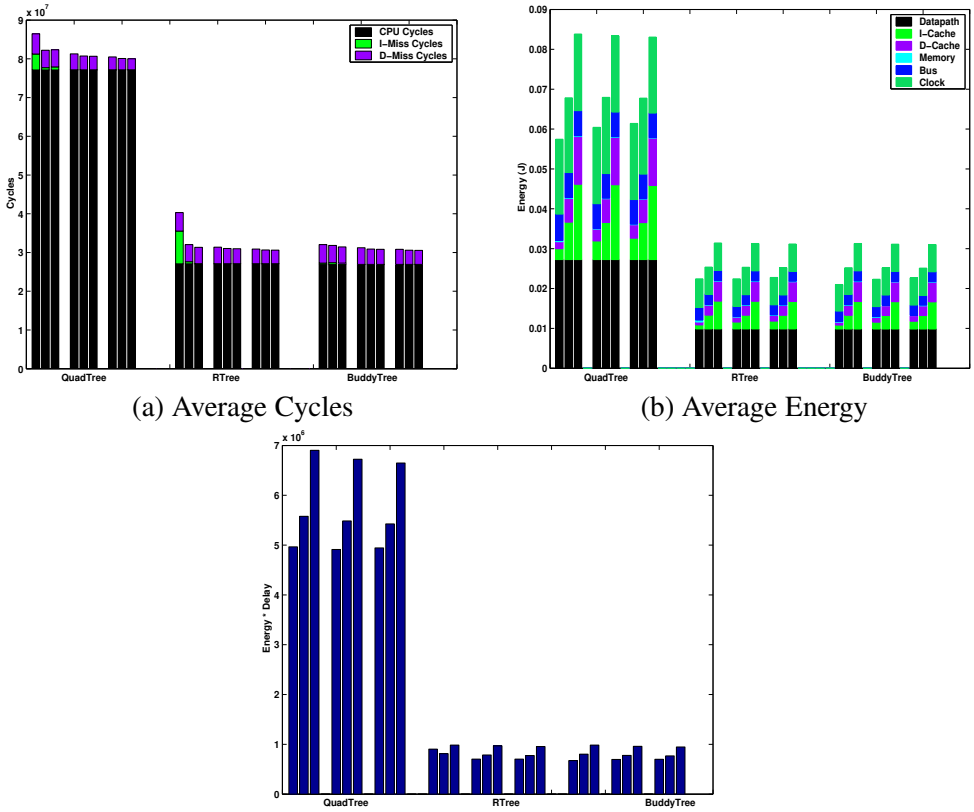



Fig. 4.5. Comparison of Index Structures for Point Queries: Average Cycles and Energy for Filtering and Refinement Steps. The nine bars from left to right for an index correspond to cache configurations of (8K,DM), (8K,2way), (8K,4way), (16K,DM), (16K,2way), (16K,4way), (32K,DM), (32K,2way), (32K,4way).




(a) Average Cycles

(b) Average Energy

(c) Average Energy * Cycles

Fig. 4.6. Comparison of Index Structures for Range Queries. The nine bars from left to right for an index correspond to cache configurations (cache size, cache associativity) of (8K,DM), (8K,2way), (8K,4way), (16K,DM),( 16K,2way), (16K,4way), (32K,DM), (32K,2way), (32K,4way).

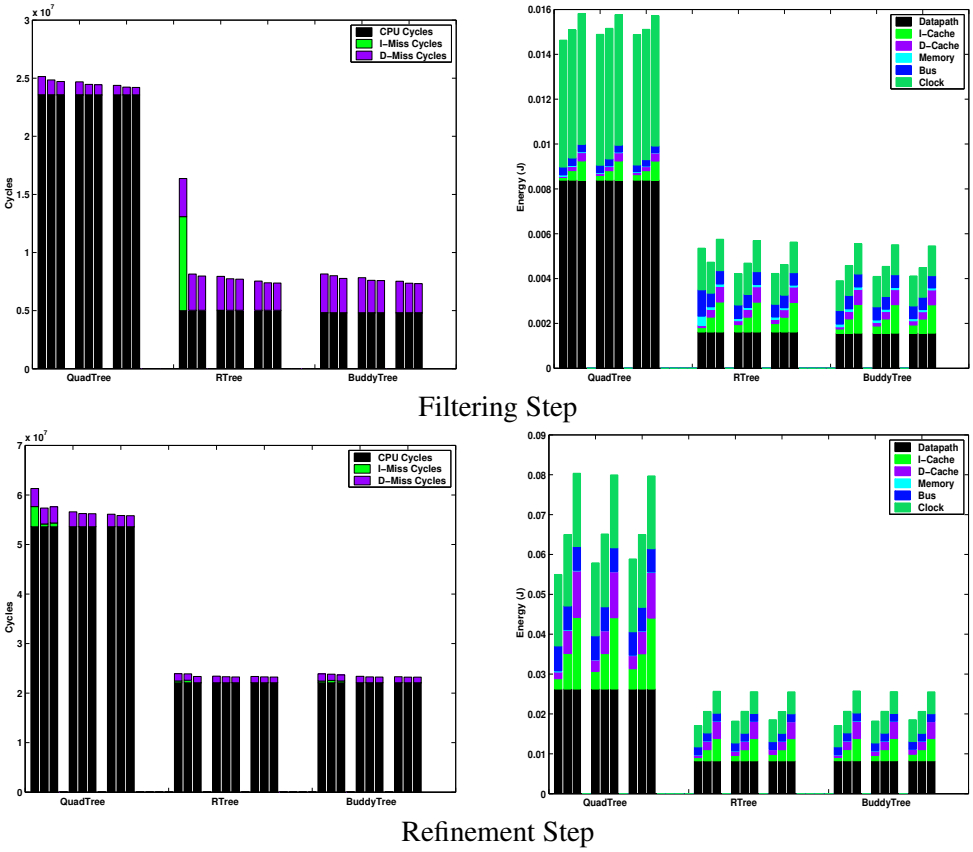
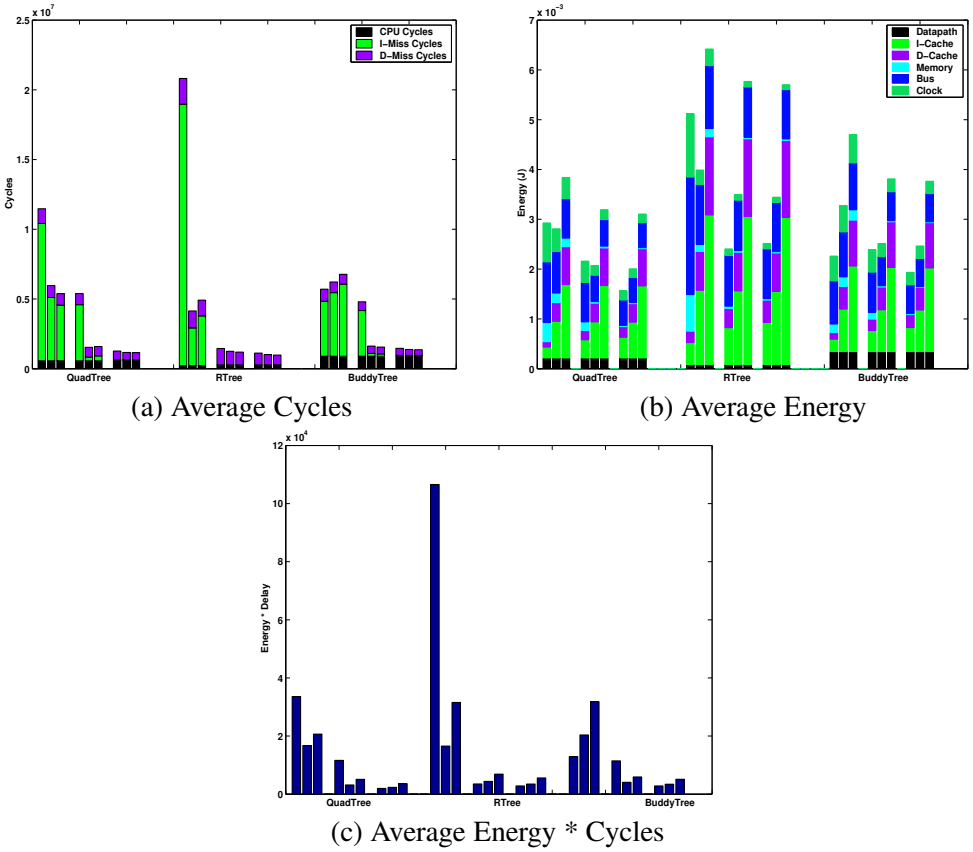



Fig. 4.7. Comparison of Index Structures for Range Queries: Average Cycles and Energy for Filtering and Refinement Steps. The nine bars from left to right for an index correspond to cache configurations of (8K,DM), (8K,2way), (8K,4way), (16K,DM), (16K,2way), (16K,4way), (32K,DM), (32K,2way), (32K,4way).




(a) Average Cycles

(b) Average Energy

(c) Average Energy * Cycles

Fig. 4.8. Comparison of Index Structures for Nearest Neighbor Queries. The nine bars from left to right for an index correspond to cache configurations (cache size, cache associativity) of (8K,DM), (8K,2way), (8K,4way), (16K,DM), (16K,2way), (16K,4way), (32K,DM), (32K,2way), (32K,4way).

| | Normal | | |
|---|---|---|---|
| | Cycles | Energy (J) | Energy*Cycles |
| Quadtree (8K,2-way) | 5959406 | 0.0028102 | 16747.1 |
| R-tree (8K,2-way) | 4145641 | 0.0039902 | 16541.9 |
| Buddy-Tree (8K,2-way) | 6218897 | 0.0032773 | 20381.2 |
| Quadtree (16K,2-way) | 1532206 | 0.0020735 | 3177.0 |
| R-tree (16K,2-way) | 1252116 | 0.0034984 | 4380.4 |
| Buddy-Tree (16K,2-way) | 1625547 | 0.0025138 | 4086.3 |
| Quadtree (32K,2-way) | 1169681 | 0.0020068 | 2347.3 |
| R-tree (32K,2-way) | 1012516 | 0.0034450 | 3488.1 |
| Buddy-Tree (32K,2-way) | 1383872 | 0.0024659 | 3412.5 |
| | With eDRAM | | |
| | Cycles | Energy (J) | Energy*Cycles (improvement) |
| Quadtree (8K,2-way) | 1111398 | 0.0020920 | 2325.0 (86.1%) |
| R-tree (8K,2-way) | 614581 | 0.0034671 | 2130.8 (87.1%) |
| Buddy-Tree (8K,2-way) | 1429412 | 0.0024918 | 3561.8 (82.5%) |
| Quadtree (16K,2-way) | 668678 | 0.0019456 | 1301.0 (59.0%) |
| R-tree (16K,2-way) | 325228 | 0.0033611 | 1093.1 (75.0%) |
| Buddy-Tree (16K,2-way) | 970077 | 0.0024064 | 2334.4 (42.9%) |
| Quadtree (32K,2-way) | 632425 | 0.0019272 | 1218.8 (48.1%) |
| R-tree (32K,2-way) | 301268 | 0.0033398 | 1006.2 (71.2%) |
| Buddy-Tree (32K,2-way) | 945909 | 0.0023939 | 2264.4 (33.6%) |

Table 4.6. Impact of eDRAM on Nearest Neighbor Queries averaged over all datasets

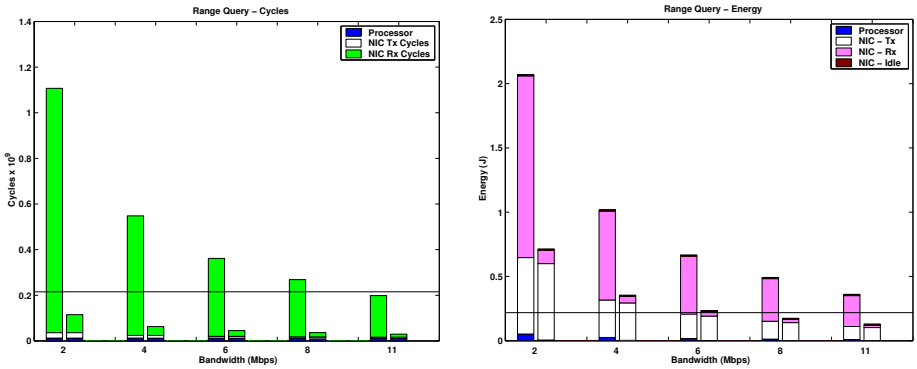



Fig. 4.9. Range Queries. Comparing the option of performing the query at the client (shown as the horizontal line, vs. the option of performing the query completely at the server (shown as bars). The left bar for each bandwidth, are for the case where data objects are not available at the mobile client and need to be shipped from server, while the right bars are for the case where data objects are already available on the client. The profile for energy and cycles is given in terms of what the mobile client incurs in the NIC (given separately for transmission, reception and idle) and all other hardware components that are bunched together as processor.

| | Cycles | Energy | Energy*Delay |
|---|---|---|---|
| Point Query | 1. BuddyTree<br>2. R-Tree,QuadTree | 1. BuddyTree<br>2. R-Tree,QuadTree | 1. BuddyTree<br>2. R-Tree,QuadTree |
| Range Query | 1. BuddyTree,R-Tree<br>2. QuadTree | 1. BuddyTree,R-Tree<br>2. QuadTree | 1. BuddyTree,R-Tree<br>2. QuadTree |
| Nearest Neighbor | 1. R-Tree<br>2. QuadTree<br>3. BuddyTree | 1. QuadTree<br>2. BuddyTree<br>3. R-Tree | 1. QuadTree<br>2. BuddyTree<br>3. R-Tree |

Table 4.7. Comparison of Index Structures for different queries and criteria using the results of 2-way 16K cache configuration. (1) denotes the best and (3) denotes the worst for each entry in this table

# Chapter 5

# Storing Spatial Data on a Network of Workstations

## 5.1 Introduction

A Spatial Database Management System (SDBMS) stores, retrieves and analyzes spatial information. Apart from obvious applications in geographic and cartographic domains, the ability to explore physical, logical and temporal properties of information in a SDBMS makes it popular in numerous other areas as well. Demography, epidemiology, terrain analysis, mining, military planning and logistics, computer-aided design, computer vision and robotics are just a few of the domains where such a system can find widespread applicability.

All these different applications are uniformly characterized by the vast amount of information that needs to be stored, retrieved and analyzed. In addition to just being able to handle these large data sets, a SDBMS should also be able to perform queries on this data efficiently to meet real-time constraints. Queries to a SDBMS are not necessarily limited to spatial searches or selections. In a combat terrain information system, the troops not only require current information about the battlefield, but are also interested in finding the shortest (or most strategic) routes to take under constantly changing conditions (such as a bridge getting blown up). Answering the query would require the latest information about the battlefield and may need to employ complex algorithms to process this data. In these situations, it is important to lower query response times.

To meet such stringent requirements, a SDBMS must employ a high performance computer system. Conventional platforms for SDBMS have used high performance Input/Output

(I/O) subsystems that are attached to a high performance workstation [67]. However, despite the I/O parallelism offered by some of these systems (such as RAID), the channel between the processing center and I/O system can itself become a bottleneck, limiting the speed of data transfer. Further, such an architecture does not provide any additional computational power for executing complicated queries beyond the raw processing power of the native workstation. This observation leads us to believe that a balanced high performance platform for a SDBMS should support parallelism in processing (CPUs), primary (memory) and secondary (disk) storage, as well as I/O channels.

Recent trends in computer architecture show that a Network of Workstations (NOW) - also referred to as cluster systems - is emerging as a cost-effective solution for high performance. It is feasible today to put together a cost-effective high performance platform for SDBMS with rapidly improving off-the-shelf workstations and network hardware. The multiple CPUs and their memories can provide processing and primary storage parallelism, while disks connected to individual workstations on this network can provide secondary storage parallelism for both data access and data transfer. However, there are several open research issues to be addressed in harnessing the full capabilities of such a platform to realize a high performance SDBMS, and this study takes a step towards this goal.

Several algorithmic, software and hardware design alternatives/parameters can significantly impact the performance of a SDBMS on a NOW platform. On the algorithmic side, the choice of data structures used to maintain the spatial information and algorithms for manipulating these data structures are largely influenced by the information that is being stored and the queries on this information. On the software side, the distribution of data between different workstations would dictate the communication overheads in the execution. Placing a majority

of the data locally would lower communication costs for a workstation, but it could result in a mismatch of workload between workstations resulting in loss of parallelism. We need to ensure that queries that are focussed on a few data items involve as few workstations as possible to minimize communication overheads (*minload*), while queries that involve large search windows are distributed uniformly across the workstations (*unispread*). It is thus important to keep both data placement and load balance in mind when designing a SDBMS on a NOW platform. There are also several software messaging alternatives ranging from traditional TCP/IP sockets, to RPCs and more recent (and efficient) user-level messaging layers [122] that would have a direct bearing on the communication and synchronization overheads in the execution. On the hardware side, the processing capabilities of the workstation CPUs, amount of physical memory, disk bandwidth, and the network used to connect the workstations are just a few of the important parameters likely to impact performance.

A comprehensive evaluation of all these alternatives is too ambitious and is beyond the scope of our study. However, this research takes the first step towards this goal by keeping constant a few of the alternatives, and varying others. First, we limit the study to spatial data structures, specifically the R-Tree [48], and evaluate the performance of insertions and spatial searches on the R-Tree. The hardware NOW platform in this exercise consists of eight UltraSPARC Enterprise Model 170 servers connected by switched 100 Mbit/sec Ethernet and 1.28 Gbit/sec Myrinet [20]. TCP/IP sockets (kernel-based) on this hardware is used for communication.

This chapter [9, 8] extensively evaluates a spatial data structure, such as an R-Tree, experimentally on a Network of Workstations (NOW) platform and investigate different trade-offs in design alternatives. The specific contributions of this chapter are:

- A taxonomy for data distribution of hierarchical spatial data structures, particularly an R-Tree, is proposed.
- A generic architecture for prototyping different distribution schemes is developed. This architecture is used to specifically implement four distribution schemes.
- The impact of number of workstations, size and nature of the data set on the performance of insert and spatial search operations is studied for these distribution schemes on an experimental platform.
- The results show that a distributed R-Tree structure, specifically the one proposed in this chapter, can significantly lower response times and increase throughput for spatial searches compared to a traditional implementation. It attains this goal without compromising on the time taken to build this structure.

The rest of this chapter is organized as follows. Section 5.2 identifies other efforts related to this exercise. Section 5.3 discusses important design issues in distributing spatial data structures, and a description of the framework that has been built for this study is given in Section 5.4. The performance results from different experiments are detailed in Section 5.5. Finally, Section 5.6 summarizes the lessons learned from this study and outlines directions for future work.

## 5.2 Related Work

Several spatial data structures [102] such as Grid Files [53], Quad-trees [36], $k-d$-trees [18], $k-d$-B-trees [97], Cell-trees [46], BANG files [37], $h$B-trees [76], R-Trees [48] and its variations [82], have been proposed, and techniques for efficiently implementing spatial queries on these data structures have been evaluated [92]. Of these, R-trees (and its variants) are amongst

the most popular. As pointed out in [70], R-trees have the property that they do not cut spatial regions into any more pieces than needed and are more robust for higher dimensionalities than many other spatial data structures.

To speed up I/O access, several techniques [67] for partitioning spatial data across multiple disks of a RAID system have been proposed. While these techniques can help ease I/O bottlenecks, the problem may now shift to the channel between the server and the RAID. Further, these solutions can only exploit I/O parallelism and cannot speed up the processing part of the queries as is possible in a NOW environment. It is also not clear if these techniques can be applied directly to distribute the R-tree across the disks of different machines since much of this work does not consider communication and synchronization costs.

Recently, there has been a great deal of interest in exploring parallel processing for SDBMS applications [114]. However, most of these studies are conceptual and/or theoretical in nature. Very few [110, 109] have actually considered implementations and evaluations of these systems. Shekhar et al. [110] examine declustering and load-balancing methods for parallelizing a SDBMS on a distributed memory MIMD machine, namely the Cray T3D. In another related study [109], the authors consider data partitioning and load balancing issues for vector data on a shared address space machine. However, these studies are for main memory databases, and I/O is not considered. Data partitioning and load balancing issues have also been discussed in the context of grid files [42]. Monet [21] is a database system that uses main-memory algorithms for processing spatial data while relying on the underlying operating system to efficiently manage this memory. However, this system is intended for symmetric multiprocessor (SMP) machines with a small number of processors.

An extensive implementation and evaluation of a SDBMS for shared nothing architectures has been undertaken in the Wisconsin Paradise project [92]. Multiple data servers running on different machines (Intel Pentium PCs) across an ATM network have been integrated using a query coordinator. While this is a comprehensive summary of experiences in developing a complete environment, it is not very informative on how best to distribute the data between the different servers for load balancing and minimizing communication. A global R-tree index structure across a network of workstations using a shared memory substrate has been proposed in [38] to alleviate processing and I/O bottlenecks. However, there has been no implementation or evaluation of this proposal.

The closest research to what is presented in this chapter are the ones by Koudos et al. [70], and Schnitzer and Leutenegger [103]. Koudos et al. [70] outline a technique to decluster an R-tree across a network of workstations. In this technique, a machine is dedicated as the master to maintain all but the leaf level of the R-tree. The leaves (the data) reside on different server machines, and the authors quantify the optimal capacity of the leaf nodes (blocking factor). This is, however, just one possible way of distributing the R-tree structure and has certain problems (the master can become the bottleneck in serving multiple queries as outlined in [103]). Schnitzer and Leutenegger [103] avoid some of the drawbacks of [70] by proposing a Master-Client R-Tree (MC-Rtree) which is similar to the structures evaluated in this chapter and has been concurrently developed with our research. The difference is that the distributed structures discussed here are built top-down (item by item dynamically) which is more generic than the bottom-up bulk-loading scheme used in [103] (which uses preprocessing of data for optimizations), and there are more data distribution strategies compared here. Further, all the above experimental studies have only looked at average response time for a query as the performance metric. In addition to

the response time, we also investigate the throughput supported by the distributed data structure as well as the impact of the size and location of query window.

## 5.3 Design Issues

The R-tree (and its variants) is an effective and well-understood multi-dimensional access method. For this reason we focus on various methods of adapting it to the NOW platform rather than try to develop a new spatial index. The nodes that comprise the R-tree are distributed among a number of virtual machines. These virtual machines can eventually be mapped on to a physical machine. As the tree grows, new portions are allocated and integrated into the index. We partition the space of design alternatives to implement the R-tree in terms of (1) The unit of allocation; (2) The frequency of these allocations; and (3) The distribution of these allocations across the machines. We discuss each of these issues in greater detail below.

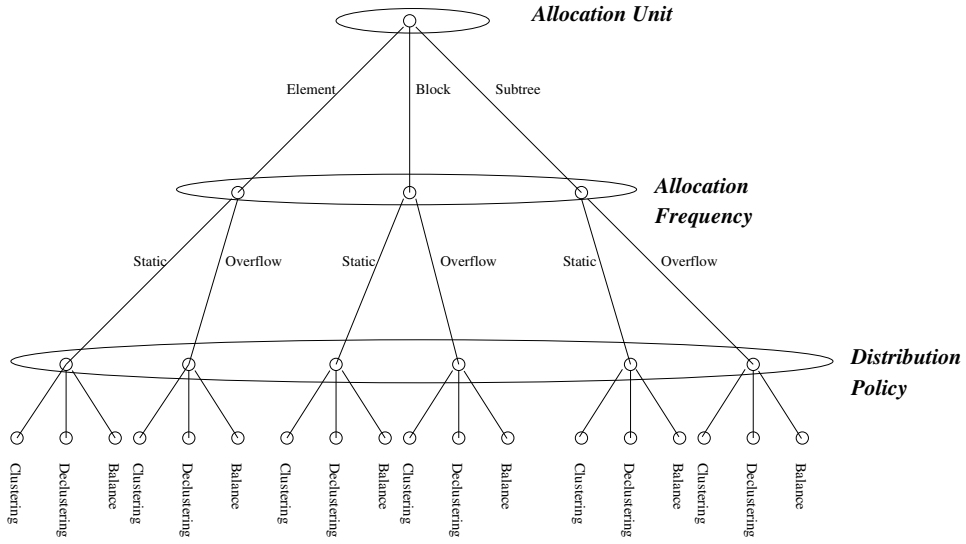



Fig. 5.1. Design Combinations

**Allocation Unit:** We identify three units of allocation:

- Element— Every element (data item) is individually allocated to a machine.
- Block— A small fixed number of index nodes are allocated to machines independently.
- Subtree— A large number of blocks are allocated as a unit.

*Element* allocation allows the greatest control over distribution. The data can be clustered, declustered, or balanced (these terms are explained later in Distribution Policy) effectively across the machines. *Block* allocation allows either effective clustering and declustering, or effective clustering and data balance (it may not provide enough control to achieve all three goals concurrently). However, because each block may be allocated to a different machine, there may be significant communication and synchronization cost associated with this choice. *Subtree* allocation places all elements within a relatively large spatial region on the same virtual machine. The subtrees allocated can be of a fixed or variable depth. Similar to block allocation, subtree allocation can combine clustering with either declustering or data balance, but perhaps to a lesser degree. However, it allows a simpler implementation than block allocation, and provides the flexibility of letting the virtual machine choose its own indexing scheme and data storage methods. Thus, with subtree allocation, it would be possible to let a virtual machine use a commercial off-the-shelf (COTS) database product to store and retrieve the data assigned to it.

**Allocation Frequency:** Once we have decided what we allocate, the next design issue is when to make these allocations to the virtual machines. Allocation frequency impacts efficiency. Infrequent allocation can lead to poor tree balance while frequent allocations increase overhead. We consider the following possibilities:

- Static— Allocation is done once, when the index is built.
- Overflow— Allocation decisions are made when an allocation unit overflows.

The advantage of *static* allocation is low allocation overhead. It should be sufficient when the tree is largely static, or the distribution of the data is well understood. If this is not the case, it can result in poor tree balance. *Overflow* allocation in the case of element allocation corresponds to insertion. In the case of subtrees, overflow may be defined in terms of depth, number of elements in the subtree, etc. Allocation at overflow can provide better tree balance.

**Distribution Policy:** The final design choice we consider is the policy used to distribute allocation units. The allocation may either directly pertain to physical machines or allocation to virtual machines and then mapping these virtual machines on to physical machines to achieve the intended distribution. The principal choices are:

- Clustering— Attempt to allocate units that are spatially near each other on the same machine.
- Declustering— Allocate units that are spatially near each other on different machines.
- Balance— Units are allocated so as to maintain data balance. One possibility is allocating units to machines in a round-robin fashion.

*Clustering* generally contributes toward the goal of minload, i.e., small query windows should activate a small number of machines. *Declustering* contributes toward unispread, i.e., large queries should be distributed uniformly across a large number of machines [67]. The *balance* allocation method attempts to balance the load across the machines.

Figure 5.1 shows the various combinations of design alternatives. We will refer to them as sequences of the form *allocation type/allocation frequency/distribution policy*. For example, we would represent a design with element allocation, allocation on overflow, and a balance distribution policy as *element/overflow/balance*.

## 5.4 System Implementation

We have designed and developed an extensive experimental platform on a network of Sun UltraSPARC workstations to prototype the above design choices.

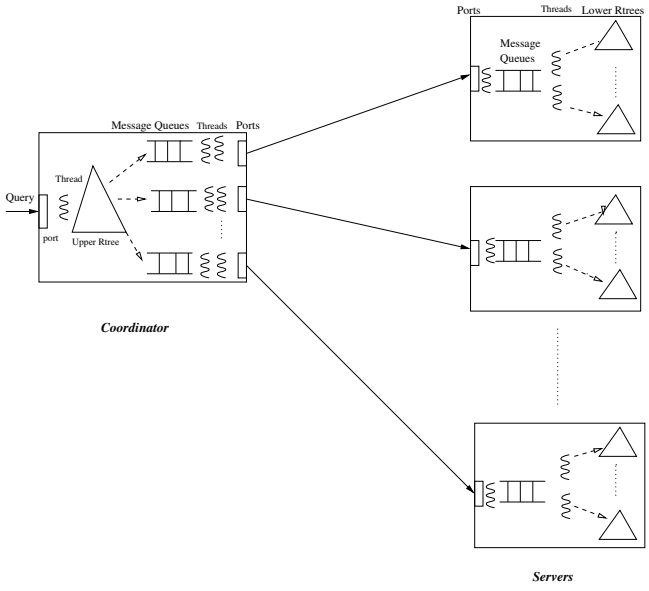



Fig. 5.2. Architecture of Prototype

### 5.4.1 Architecture

A schematic of our system is shown in Figure 5.2. As in the two other related studies [70, 92], our system dedicates one machine (*coordinator*) that coordinates the activities of the other machines (*servers*) in the system. The client service requests are presented to the coordinator. These requests include insertions and queries. The queries, as discussed earlier, are limited to spatial selection specified by a bounding rectangle (our ongoing work is extending this to spatial joins as well). The result being a set of element identifiers (EIDs) for the elements from the dataset that intersect the bounding rectangle. We assume that the elements are stored separately, and that the EID provides efficient access to the element (e.g. file, page, offset).

The system uses R-tree structures at the coordinator and servers. Typically, as soon as the client request arrives, the coordinator will have to search its R-tree structure to find the server(s) to which the request should be directed. The servers themselves use their R-trees to locate the EIDs that need to be sent back to the coordinator. We shall refer to the R-trees maintained at the coordinator and servers as *Upper* and *Lower* R-trees respectively, since when one visualizes the global structure across these machines, the coordinator R-tree would be the higher levels of the hierarchical structure while the server R-trees would constitute the lower levels. In fact, the implementation discussed in [70] is a specialized case of this structure where the Lower R-tree contains only the leaf level and the Upper R-tree contains all higher levels. The Upper and Lower R-trees on the coordinator and servers are R-trees [48] adapted from an implementation that is available from a public domain site at UC Berkeley. The R-trees are disk resident. We map the file containing the R-tree into memory with *mmap*. This provides good buffering of the file, but makes it difficult for us to isolate I/O costs.

Both the client and server implementations are multi-threaded. A thread in the coordinator listens on a well-known port for service requests from the client. When a request arrives, it uses the Upper R-tree to determine which Lower R-Tree (virtual server) will process the request. The reader should note that our implementation provides the view of a virtual server, thus hiding details about where the server actually resides. There can be another level of mapping of virtual servers to physical machines, thus potentially allowing a machine to export more than one virtual server. The coordinator thread then places a request in the service queue for each server that will process the client's request. There are several threads dedicated to interactions with each server potentially allowing more than one pending request between the client and the server. One of these threads retrieves requests from its service queue and forwards it to the server over a dedicated communication channel. This thread then blocks waiting for a response. A pool of threads at each server services requests from the coordinator. The thread receiving the message executes the request using its local indexes and returns the result to the coordinator. In the case of an insertion, the result is simply an acknowledgment. However, a spatial search may return a large number of elements. which are then transferred to the blocked thread at the coordinator. The result is transferred from the coordinator to the client in a similar manner. The coordinators and servers communicate through dedicated communication channels via the standard TCP/IP socket interface.

### 5.4.2 Implementation of Data Distribution Schemes

The above architecture makes it easy to prototype and evaluate a range of R-Tree distribution schemes. In this study, we consider five specific distribution schemes of the R-Tree

(named Schemes A, B, C-RR, C-DC and D). These implement *element/overflow/balance*, *subtree/static/cluster*, *subtree/static/balance*, *subtree/static/decluster*, and *subtree/overflow/balance* respectively (see Figure 5.1). The reasons for selecting these five schemes are as follows. As mentioned in [8], the *block/*/** schemes are either not very interesting, or are expected to not perform very well on our experimental platform because of high communication/synchronization costs. From the *element/*/** schemes, Scheme A is relatively straightforward to implement as described below. The *subtree/static/** schemes (Schemes B and C) have also been used by other studies [90], since spatial partitioning makes it easier for certain queries (such as spatial joins). Finally, we would like to explore at least one scheme in the *subtree/overflow/** category, which is implemented by Scheme D.

It is important for the reader to note that the issues identified in Section 5.3 are design goals to be met in distributing the hierarchical data structure. They do not in any way enforce how these design goals should be met. For instance, when the goal is *element/*/balance*, the implementation could use round-robin or any other method to allocate an equal number of elements between the servers. Hence, before we study the performance of these schemes, it is important to understand how these design choices can be implemented. Note that for each scheme, this is just one possible implementation to achieve the design goals.

For Schemes A and B, there are exactly $n$ (where $n$ is the number of server machines) virtual servers, and each machine holds one virtual server (one Lower R-Tree). In both the schemes, when the server receives a query, the necessary (local) actions are performed on the Lower R-Tree it holds.

In Scheme A, the coordinator uses a simple round-robin criteria to allocate insertions to virtual servers. To decompose queries, it maintains the bounding rectangle for each server.

Initially, these bounding rectangles are nonexistent. When a tuple is inserted at a server, the bounding box for that server (maintained at the coordinator) is updated (if necessary) to include the new element. In Scheme B, we assume that the data extent is known in advance. The coordinator partitions the extent into *n* equal area regions/tiles and makes the mapping between a tile and an associated virtual server (Lower R-Tree). Each data item is inserted into the appropriate virtual server for the region where the data item falls. In this approach, insertions never cause the bounding boxes to be modified, though the servers can become imbalanced.

Schemes C and D can result in more than one Lower R-Tree (virtual server) per server machine. This number is decided statically in Scheme C while it is dynamic in Scheme D.

Scheme C is similar to Scheme B in that the spatial extent is known in advance, and is divided into tiles (with each tile assigned to a virtual server). The difference is that the number of tiles can be higher than the number of server machines, with each machine potentially containing more than one virtual server. Within Scheme C, we consider two variations, namely C-RR and C-DC. Tile $i$ in both schemes falls on machine ($i \bmod n$). The difference between these two schemes is in the numbering of tiles themselves. Scheme C-RR numbers the tiles in row-major order, while Scheme C-DC numbers the tiles based on $z$-ordering. The reader should note that Scheme B is a specific instance of Scheme C when the number of tiles equals the number of server machines.

Scheme D is significantly different from the other three, and is quite complicated because of its dynamic (overflow) nature in creating new Lower R-Trees. Our novel algorithm to implement this scheme is shown in Figure 5.3. The Upper R-Tree traversal uses the traditional algorithm to find candidate Lower R-Trees into which the data item can be inserted (Step 1). At this point, we are not only trying to limit the data items on the candidate Lower R-Trees, but we

```
Step 1. Traverse Upper R-Tree at coordinator till leaf level
        (leaf-level points to Lower R-Trees)
Step 2. Search leaf entries to find a candidate Lower R-Tree
        (candidate is the one which would result in a minimum
         change in area because of the new addition, and the
         change in area is less than a certain threshold, and
         has less than a specified number of elements)
Step 3. If candidate found
             Insert element into the corresponding Lower R-Tree
        else
             Identify a server machine
             Create Lower R-Tree on it
             Insert created Lower R-Tree into the Upper R-Tree
```

Fig. 5.3. Insertion Algorithm for Scheme D

would also like to insert this item into a candidate whose bounding box (MBR) would change the least. It may happen that this does not suffice, and we may be better off creating a new Lower R-Tree altogether. We weigh this decision by comparing the change in MBR with a threshold to decide whether to create a new Lower R-Tree or to insert it in the candidate. If and when a decision is made to create a new Lower R-Tree (virtual server), it is created on a machine whose closest Lower R-Tree to the newly created one is further than that on the other machines (for declustering). The threshold for the change in MBR is itself made dynamic, starting off with a large value and progressively decreases. This parameter has been tuned based on experimental results.

Since the bounding boxes of Lower R-Trees for Schemes A and D can be overlapping, a data item resides in exactly one of the Lower R-Trees. However, Schemes B and C do not allow overlaps. As a result, a data item spanning more than one tile could reside (has to be duplicated) on all the corresponding Lower R-Trees. Further, in Schemes A and B, the number of levels in

the Upper R-Tree at the coordinator is one (since $n$ is typically much smaller than the number of Lower R-Tree pointers that a block can hold). In Schemes C and D, this depends on the number of Lower R-Trees that are created.

### 5.4.3 Performance Metrics

The principle performance metrics of interest are response time (in seconds) for large queries and insertions, and throughput for small queries (in queries/second). These two metrics capture the desirable unispread and minload characteristics that we require from the distributed data structure [67]. When the query window is large, the computational and I/O parallelism provided by multiple servers would lower response time. When the query window is small, ideally, only a restricted number of servers should be activated. This implies that other servers should be able to respond to queries with different windows, thus maximizing the number of queries serviced per unit time (throughput).

## 5.5 Performance Results


Fig. 5.4. Data Sets

The schemes discussed in the previous section have been evaluated on 167 MHz Sun UltraSPARC Enterprise workstations connected by 100 MBit/sec switched Ethernet and 1.28 GBit/sec Myrinet [20] running Solaris 2.6. We present results on the performance of Insert, Large Spatial Query and Small Spatial Query operations. For insert and large queries, the performance metric is the response time. For small queries, the throughput over multiple queries is the metric of interest. The results that follow in Sections 5.5.1 through 5.5.4 are for the switched Ethernet platform, and Section 5.5.5 studies the schemes with the faster Myrinet network.

It is interesting to study the impact of the characteristics of the data set on the performance of the schemes. Specifically, we are interested in the impact of clustering (skews in the spatial distribution) in the data set. We have used four data sets in our experiments. The first two contain polyline data of the streets in the states of PA (223K polylines which are more or less uniformly distributed in space) and CA (248K entries that are more skewed/clustered than PA) drawn from the Tiger [83] data set. We have also used a point data set (209K entries) drawn from a Computation Fluid dynamics application [103], and a polygon data set representing the census block data [83] for the state of CA (400K entries). Henceforth, these four data sets are referred to as Polyline-Uniform, Polyline-Clustered, Point and Polygon respectively. A pictorial view of these data sets is given in Figure 5.4.

To limit the number of experiments, we first conduct comparisons between the different alternatives for Scheme C so that we can fix its parameters. The first is related to whether to use Scheme C-RR (which tries to assign tiles/regions in a round-robin manner to the servers) or Scheme C-DC (which declusters the tiles/regions amongst the servers). The second is related to the number of tiles that should be assigned to each server (and the spatial extent is tiled based on this number). We determine these parameters experimentally. Based on the best performance,

we have chosen Scheme C-DC (which will henceforth be referred to as just Scheme C) and use 6, 4, 2 and 8 tiles/machine for the four data sets respectively in subsequent experiments.

In the following graphs, we first show the performance of implementing the operation on an ordinary R-tree without any distribution (called *local*). We compare this performance with the distributed R-tree implementations as a function of the number of server machines that are employed.

### 5.5.1 Insert Operation

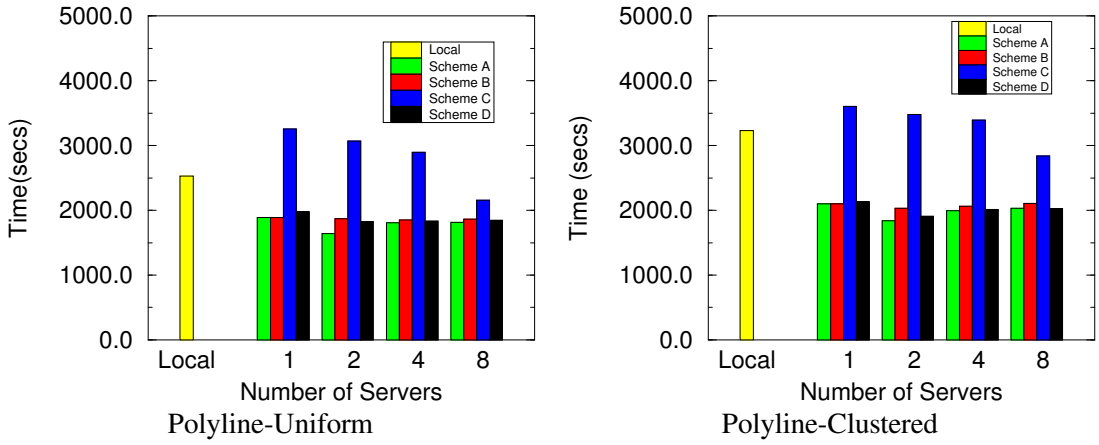



Fig. 5.5. Time for Inserting all the Data Items (Building)

Figure 5.5 shows the total time for inserting all the records (building the R-Tree element by element) in the polyline data sets for the four schemes. The results for the other data sets are similar.

The first observation we can make is that for each of the four schemes, the time for building the distributed data structure changes marginally as we increase the number of server machines. As the number of servers increases, there is potential parallelism in the insert operation itself (several more can be outstanding from the coordinator to the data servers, and these can potentially be performed in parallel). However, there is the higher overhead due to communication and synchronization costs. These two factors nearly balance each other giving almost similar performance (except for, perhaps, Scheme C where the former factor is a little more dominant until 8 servers) as the number of servers is increased. Between the schemes, we find that the overheads for Scheme C are much higher than the rest, due to the cost of supporting multiple R-trees at each node as well as due to the duplication of data items. One may wonder how the building time can go down when one moves from the Local to the 1 server case for Schemes A, B and D. This anomaly is because our local implementation is not multithreaded (no overlap of computation with I/O), while in the 1 server case, activities at the server end could overlap with the activities at the coordinator. Except for Scheme C, the time for creating the distributed structure is roughly comparable for the remaining schemes, and this time is comparable (if not better) than the time for building the local (traditional) R-Tree.

The reader should note that it is typically more important to optimize search and other queries rather than inserts/builds since they are less frequent. Consequently, the rest of this discussion is focussed on spatial searches.

### 5.5.2 Large Range Query

Figure 5.6 shows the response times for a large range query (that covers 25% of the spatial extent) for the four schemes on the data sets. The size and location of the query window play an

important role on performance, as will become apparent later on. To take this into consideration, we have run range queries at five different locations in the spatial extent, and with five different aspect ratios (1:1, 1:2, 2:1, 1:3, and 3:1). Figure 5.6 shows the average of these response times.

Uniformly, we find that distributing the data structure leads to an improvement in the response time for a large query with an increase in the number of servers. In fact, all the schemes outperform the search on a local (traditional) R-Tree beyond two servers. Scheme A does the best (reducing response time with increase in the number of servers) on all data sets because it distributes each data item in a round robin fashion. The large query retrieves a large number of data items, and Scheme A distributes this load evenly (data balance) between the servers. In some sense, one could hypothesize that Scheme A sets an upper bound on performance for a large range query because it provides the best data balance (unispread). The poorer performance for Schemes B and C (compared to A) is mainly due to the duplicates that they retrieve. In fact, these schemes are not very scalable for the point data set, with the performance worsening when we move from 4 to 8 servers. Of these two, Scheme B performs worse than Scheme C for the following two reasons. First, Scheme C can potentially achieve better data balance because of the tiling and declustering. Second (and perhaps more importantly), since Scheme C accommodates more Lower R-Trees at each machine, there are more threads which service a request (which has been broken down into smaller requests) to provide parallelism between I/O and CPU activity at a server. On the other hand, Scheme B has devoted only one server thread per query, and there is no CPU activity that can overlap with I/O operations required for servicing the large query at the server. It should be noted that both Schemes C and D have additional overheads (due to switching/swap costs) in handling multiple Lower R-Trees at each machine. Despite these

overheads, Scheme D still performs very well, and comes quite close to Scheme A, suggesting that it achieves good data balance for all the data sets.

### 5.5.3 Small Range Query

Figure 5.7 shows the delivered throughput (queries served per second) for the four schemes on the data sets as a function of the number of servers. Randomly generated small query windows (1.5% of the spatial extent) are used to obtain these results. The reader should note that a higher throughput is preferable in these experiments.

An important criteria to be met in maximizing the throughput of the system is to activate as few Lower R-Trees (minload) as possible (specifically the only one containing the data for the queried region) in serving each query. This would allow the remaining Lower R-Trees to handle other queries resulting in higher throughput. It is intuitively clear that Scheme A should perform poorly in terms of the minload criterion since each query (however small) would result in activating a large number of machines. Figure 5.7 confirms this intuition. Because of spatial partitioning (which tries to meet the minload criteria), both Schemes B and C give relatively higher throughput. Of these, Scheme B does better on the polyline-uniform data set because the data imbalance is not a significant problem, and it does not have the overheads of Scheme C in supporting multiple Lower R-Trees at each machine. However, Scheme C outperforms for the polyline-clustered data set because of the better balance it provides. For the point dataset, we find that the throughput with Schemes B and C is significantly higher than with the other two. As mentioned earlier, the duplication overheads with spatial partitioning of point datasets is much lower than for the other data sets, since duplication is necessary only when a point falls exactly on the boundary of two spatial partitions (this probability is lower for points). Overall, we again

find that Scheme D does rather well, giving the best throughput for the polyline-clustered and polygon datasets, and doing better than Scheme A for the other two.

### 5.5.4 Varying Query Size and Query Location

In the previous two sets of experiments, we have looked at 2 specific query sizes. Traditional studies [70, 103] compare distribution strategies by taking the average response times for different locations/sizes. However, it is imperative to note that the query size and location can have an important influence on the results. To illustrate this, we vary the query window size (specified as a percentage of the total spatial extent of the data set), and the query window location and show the response times for the four schemes in Figure 5.8 for the polyline-clustered data set. The number of server machines is fixed at 8 for these experiments. The center point of the query window is fixed at the center of the spatial extent in the window size experiments. The window location experiments consider 2 locations for a 10% spatial extent window: one at the center and the other at the lower left corner.

The window size results confirm our earlier observation that Scheme A is not very suitable for small query windows (less than 1% of extent) because it does not meet the minload criteria. At larger sizes, it provides good unispread (data balance). Schemes B and C are able to meet the minload criteria for small query windows, but do not provide sufficient data balance (unispread) for larger windows. Scheme D, on the other hand, performs uniformly well across the spectrum of window sizes.

The reader can observe that a window falling on the lower left corner of the polyline-clustered data set in Figure 5.4 contains very little data. This explains the significantly lower

response times in Figure 5.8 for the corner window. We can observe that even though the window is reasonably large (10% of extent), Scheme A is worse than Schemes B or D (there is a discrepancy from the results in the window size graphs). Since the data retrieved is small, the goal is minload rather than unispread here. Also, Scheme B does much better than Scheme C, because the entire query window fits within a single Lower R-Tree, thereby achieving the minload goal. In Scheme C, it tends to fall on multiple Lower R-Trees. Here again, we observe that Scheme D is able to adapt itself to the nature of the dataset very well to achieve the minload objective despite a reasonably large query window.

### 5.5.5 Results with a Faster Network

We have conducted experiments with the polygon dataset on the faster Myrinet hardware (using the same TCP/IP socket mechanism), and we compare these results with the switched Ethernet hardware in Figure 5.9. It is apparent that the faster hardware helps lower the response time for large queries (25% of extent) and improves the throughput for small queries (1.5% of extent) for all schemes. The lowering of response times for Schemes B and C is much more pronounced than for the other two. The difference in times between that taken by a server sending back the maximum amount of data and the server sending the minimum of data would be smaller on a faster network than on a slower network. A faster network, thus, tends to reduce the impact of data imbalance between the servers. Schemes A and D are already data balanced, and the effect of a faster network on the response time of a large query is not as pronounced. In the throughput results, Scheme B tends to show the least improvement with Myrinet. This is because each server can handle only one request at a time (and the disk access time to service the

request is more dominant). As a result, at any particular time, there are not that many messages concurrently traversing the network to benefit from Myrinet.

The faster Myrinet platform still preserves the overall trends and results that were observed with the Ethernet hardware. It should be noted that many of these newer network interfaces (such as Myrinet) offer programmable processors that make it possible to implement low-latency user-level messaging [122]. If we were to use such user-level messaging layers instead of TCP/IP, then we can expect higher throughput for small queries and even lower response times for large queries.

### 5.5.6 Discussion of Results

The above results show that spatial partitioning of the data extent (Schemes B and C) is not a good idea unless the goal is only to optimize the throughput for extremely small searches. As we mentioned, the main reason for this is in the duplicates it necessitates and data imbalance between the server machines (breaking the extent into many more tiles results in higher overheads and more duplicates). One approach of alleviating the latter problem could be to explore partitioning the space into non-equal regions (based on the data set under consideration). This, however, becomes very data specific and we have not explored this approach in this chapter (and is part of our future work). However, Schemes B and C could be beneficial for other queries, such as spatial joins, which can benefit from spatial partitioning.

While Scheme A does well in terms of unispread, it results in poor performance when we need minload. This need arises either with small query windows, or when the data to be retrieved is small even for a large query window. An advantage of Scheme A is that it is relatively easy to implement.

Scheme D uniformly gives good performance across the spectrum of workloads and problem parameters. Despite incurring switching/swap overheads in supporting multiple Lower R-Trees at each machine, it is able to provide minload (lower response times and higher throughput) when the amount of data to be retrieved is small, and unispread when the amount of data to be retrieved is large. Scheme D thus offers a scalable data distribution option, and could be the scheme of choice for developing a large-scale high performance SDBMS (at least when the GIS is used primarily for spatial searches since we have not evaluated these schemes for other queries such as spatial joins). It provides these benefits with little additional building costs over the other three schemes.

## 5.6 Chapter Summary

In this chapter, we have explored the possibility of building a shared storage architecture for spatial data utilizing a cluster of workstations and their disks (a "shared nothing" system) connected by a fast network. Specifically, we have used R-Trees as a case study. We have provided a framework for exploring design decisions in distributing the R-tree across the workstations. We have also developed an extensive architecture to implement and evaluate design alternatives. Specifically, we have implemented four data distribution schemes and evaluated their performance for insert and spatial search operations on different data sets. The novelty of this study is that in addition to evaluating a range of distribution schemes on an experimental platform, the impact of different problem and query parameters (such as search window size and location) have been investigated. Many previous exercises have used average case results, and this study has shown that such an approach can lead to discrepancies in some cases. Further, traditional studies have focussed mainly on lowering response times. While this is an important

goal in answering the demands of a single user, it is imperative to note that current environments (such as the growing number of SDBMS applications on the Internet) are increasingly necessitating the need to provide high throughput for multiple concurrent users.

Of the four schemes discussed here, the novel distributed structure (Scheme D) that we have proposed has been shown to achieve the unispread (spread out the load to maximize CPU and I/O parallelism when the query requires a lot of data) and minload (keep the spatial query localized when the data required is small and accommodate several such spatial queries in parallel) properties, over a spectrum of data set and query parameters. This structure can be built as efficiently as the other three simple distribution schemes.

There are interesting extensions for this research. We would like to incorporate and evaluate more paths (schemes) in the design combination tree identified here. We plan to evaluate these distribution schemes for use with spatial join operations which are very important to many SDBMS applications.

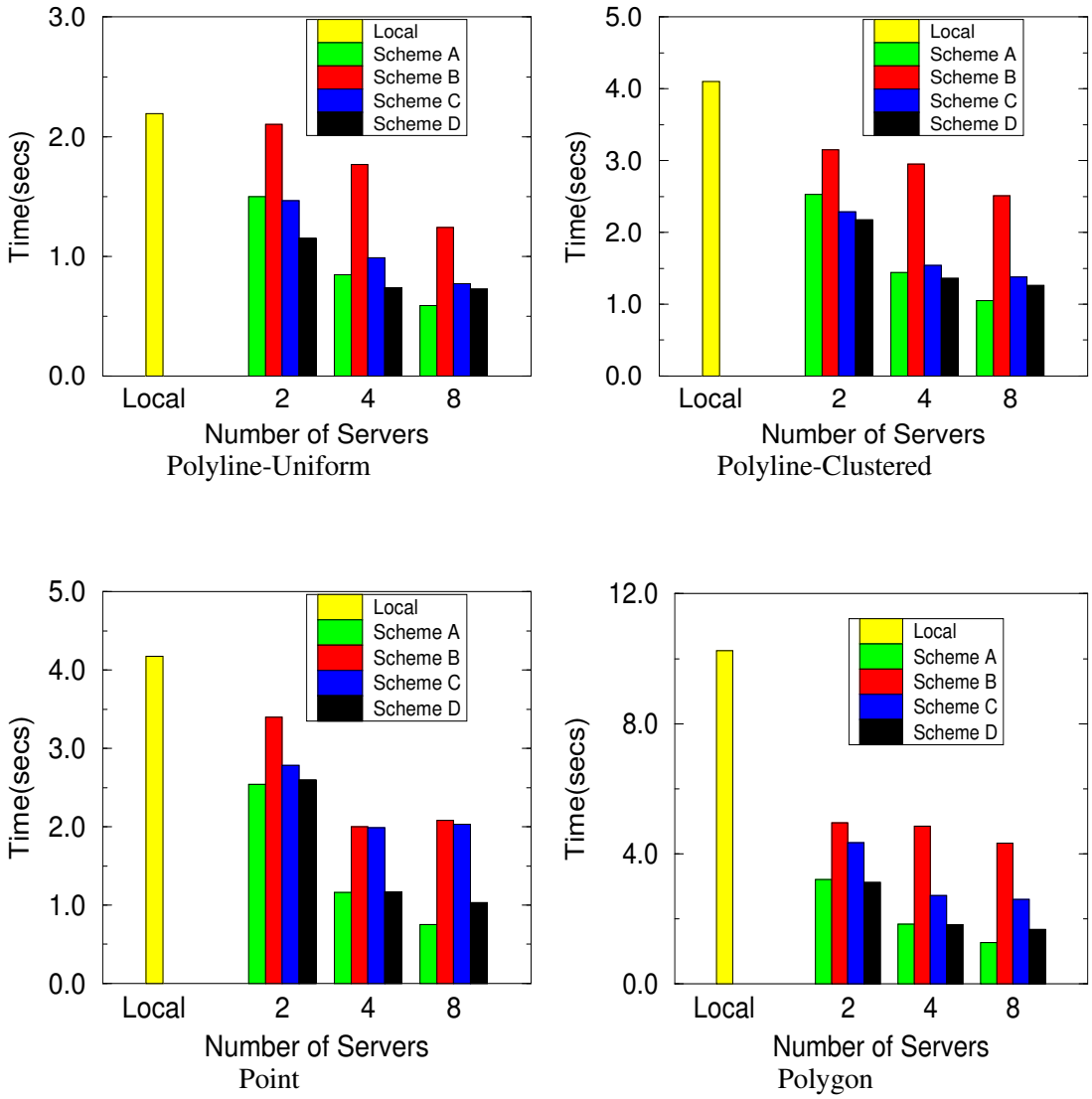



Fig. 5.6. Response Time for a Large Query (25% of spatial extent)

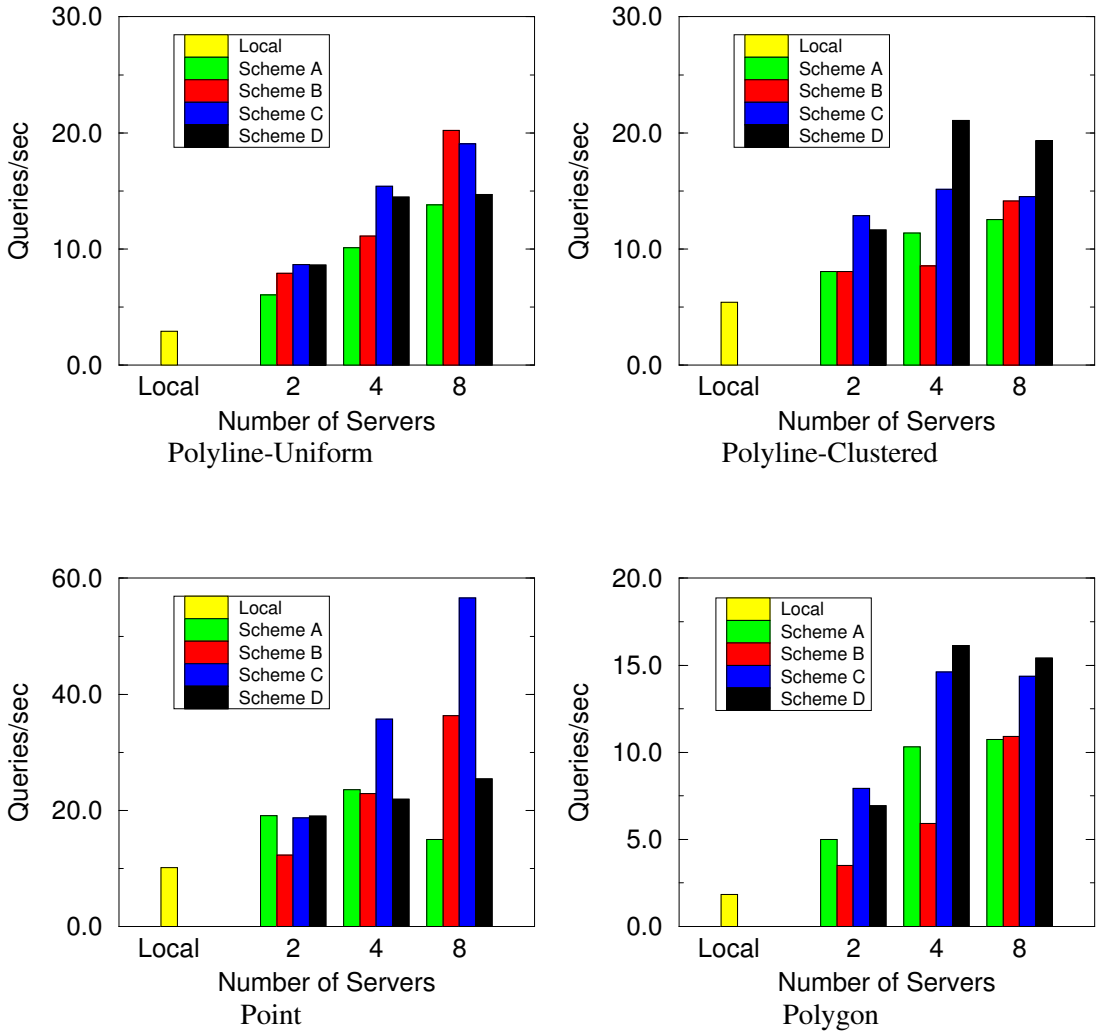



Fig. 5.7. Throughput for Small Queries (1.5% of spatial extent)

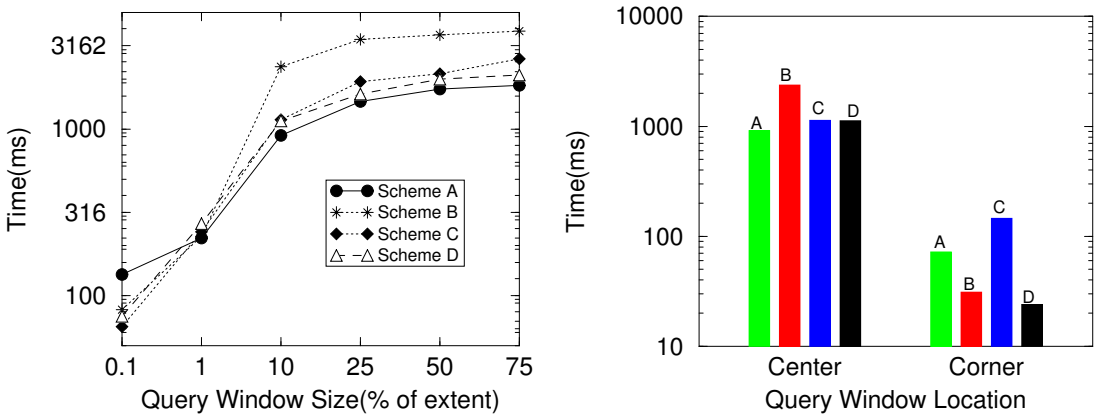



Fig. 5.8. Response Times for Different Query Windows (Polyline-Clustered with 8 servers)

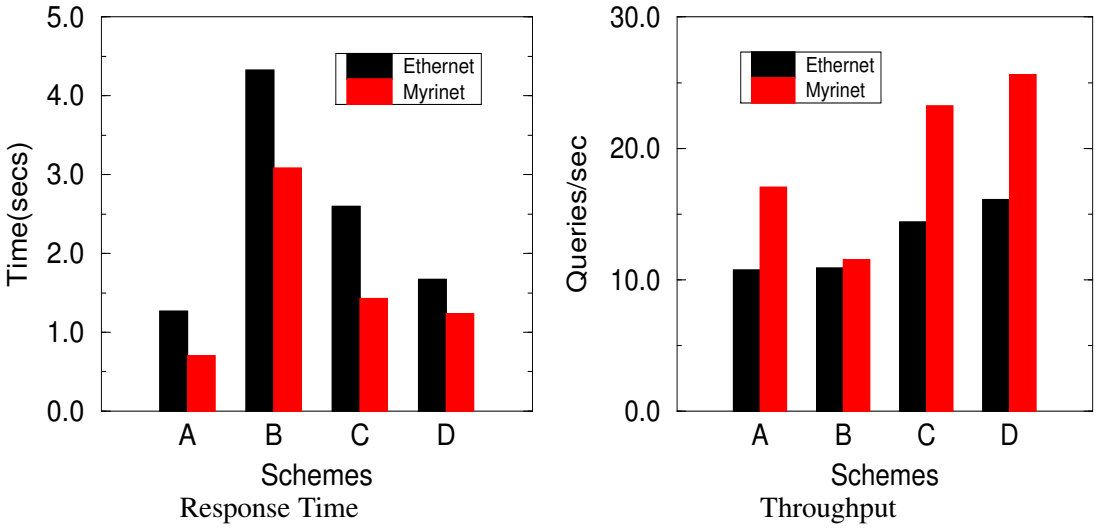



Fig. 5.9. Myrinet vs. Ethernet (Polygon with 8 servers)

# Chapter 6

# Geospatially Crawling the Web

## 6.1 Introduction

In this chapter, we study the spatial information beyond the well-structured spatial data that are discussed throughout the thesis until now. Over the last two decades or so, Spatial Database Management Systems (SDBMS) [108] have made considerable progress in storing, processing and retrieving spatial data. The technology advancements, however, have only extended people's need for spatial information beyond the well-structured and preprocessed kind that has been considered until now. They may need facts as to how far away an apartment complex is from their office or may need historic information as to how many accidents happened at one particular intersection in the past year. The proliferation of mobile devices in recent years has further stressed the need for comprehensive spatial information. For instance, tourists may wish to access their mobile devices for information about the district that they are travelling through or travelling to. We will refer to these kinds of spatial information that are not well structured as location-related information in the rest of this chapter.

With the rapid growth of Internet, people have started looking on the World Wide Web (shorten as the web henceforth) for such location-related information. Service providers like Yahoo! and Yellowpages.com are providing standard yellow page like directory services. The web, on the other hand, has more to offer than the directory services. McCurley [79] estimated that 4.5% of all the web pages contained a US zip code, 8.5% contained a recognizable phone

number, and 9.5% contained at least one of them. He also indicated that actual percentages might be higher than he reported. Even if a web page doesn't explicitly contain location information such as a zip code, it may still be location-related. For instance, Ding et al. [29] map web pages, whether containing explicit location information or not, into a geographic hierarchy. The New York Times web site, for example, has a national geographic scope in their hierarchy. On the other hand, Ambite et al. [5] have developed an application called WorldInfo that can integrate location-related information from the web with those from the SDBMS.

However, we are interested in a more fundamental problem in this growing domain: how to retrieve information related to a given location as thoroughly and fast as possible. The subsequent location-related queries and analyses could then be conducted on this confined information set instead of the entire web. Consequently, the desired information could be more relevant and delivered faster. Solving this problem could certainly augment the service provider's capability. Moreover, examining the link graph or other structures of this location-related information set may reveal additional knowledge about the social and economic structure of the target place in this Internet era.

General search engines, such as Google and AltaVista, are not very suitable for our task because of their generality, biases, and limited coverage [72, 19]. Among existing technologies, focused crawling techniques [24, 28, 2] are perhaps the closet ones to tackle our problem. Aiming at a better coverage and crawling speed, focused crawling is designed to fetch the information related only to a specific topic instead of general information on the web. Although a location can be seen as a topic, it is quite different from other general topics in many aspects. How good a focused crawler is on fetching location related information is still questionable. We may need

to design a new kind of focused crawlers to effectively retrieve location-related information on the web.

This chapter reports our current status and future plan to build such a focused crawler - a geo-spatial crawler. In the next section, we discuss the existing focused crawling techniques, especially the one we will be using [28]. We will then offer our hypotheses on the characteristics of the location-related web information in section 6.3. Section 6.4 will detail our plan to conduct a series of experiments with a state of the art focused crawler [28] to validate our hypotheses, in addition to characterizing the crawling result. At last, Section 6.5 concludes this chapter.

## 6.2 Focused Crawlers

The goal of a focused crawler is to effectively locate web pages that are relevant to a specific topic. Often, a focused crawler defines a topic not by keywords, but by a set of seed web pages. Chakrabarti et al. [24] built the first focused crawler, which consisted of two key components: a classifier and a distiller. The classifier guided the focused crawler by evaluating the relevance of a web page with regard to the topic - seed pages. An advanced classifier could evolve based on the retrieved documents during the crawling course. Working rather differently, the distiller tries to identify the hub pages on the target topic along the crawling path, then leads the focused crawler to more relevant web pages. A self-evolving classifier and an effective distiller could take the focused crawler from one topic related page to other topic related pages. However, in reality, some off-topic web pages often lead to highly relevant web pages that can not be reached otherwise.

To identify and utilize this kind of off-topic web pages, Diligenti et al. [28] proposed a novel focused crawler called the Context Focused Crawler (CFC). Before starting the crawling

process, CFC first constructs a context graph for the seed pages by back-crawling the web. The back-crawled pages make up different layers of the context graph, and which layers they belong to are determined by their link distances to a seed page. The seed pages themselves are also part of the context graph. Since their link distances to a seed page are zero, they all belong to level 0. For each layer of the context graph, CFC would train a classifier specifically for it.

When the actual crawling starts, CFC uses these classifiers to determine which layer the currently examining page needs to go to, and dynamically decides which layer the crawler would be crawling next. While keeping the merits of the standard focused crawler, CFC identifies and exploits off-topic pages on the out layer of the context graph that eventually lead it to the more desired on-topic pages.

As to our task at hand, we can treat one specific location as a general topic, and feed it to CFC for crawling web pages related to this location. Without understanding the special properties of location information, this crawling process may be ineffective. For instance, the classifier at the level 0 of the context graph, where the on-topic pages are kept, needs to identify the location information on web pages, such as the name, the zip code, and the area code of a location. We are currently implementing our CFC, and will use it in our ongoing study on location-related web information.

## 6.3 Hypothesis

As shown in Figure 6.1, researchers commonly view the web as a directed graph with web pages as nodes and hyperlinks as edges. The link structure of the web has been proven to be a powerful means to help people discover the resources on the web, classify the collected

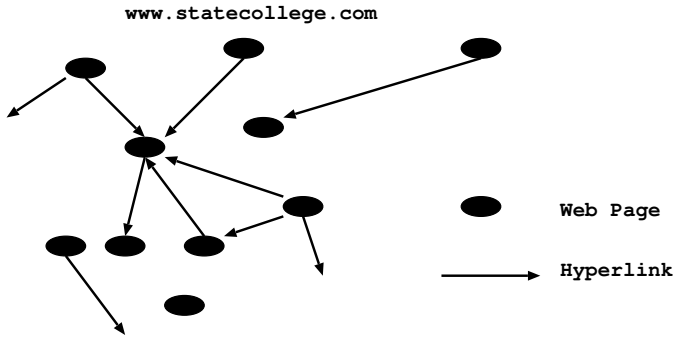



Fig. 6.1. Directed Graph

information, and more. We believe that thoroughly understanding link structures of location-related information, and utilizing them effectively are crucial to our task - building a geo-spatially focused crawler. Toward this goal, we offer some hypotheses on link structures of location-related information that need to be confirmed by the subsequent experiments.

**Hypothesis I: The overall link structure of information related to one particular location resembles the overall link structure of the World Wide Web.** Kumar et al. [71] depicted the web as a directed graph with various components (see Figure 6.2): a CORE set, a IN set, a OUT set, TENDRIL sets and DISCONNECTED sets. The CORE set consists of massive strongly connected web pages. The IN set is the set of web pages that have paths to the CORE, but have no path from the CORE; the OUT set, on the contrary, is the set of web pages that have paths from the CORE, but have no path to the CORE. The web pages in a TENDRIL set have either only paths to the OUT set, or only the paths from the IN set. They have no connection to the CORE set whatsoever. At last, a DISCONNECTED set consists of web pages that have paths only to other pages in the same set. The web pages related to one particular location is a sample

of the entire web. No matter how small or how big this sample is, it is a web by itself. We hence hypothesize that the link structure of this web will resemble the one of the entire World Wide Web.

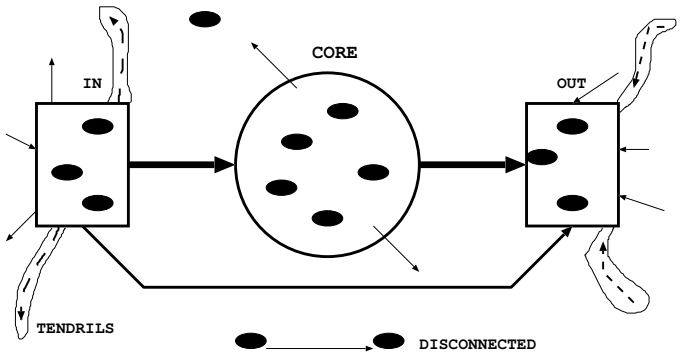



Fig. 6.2. The structure of location-related information

**Hypothesis II: Spatial proximity implies web proximity.** In Figure 6.3, State College, Pennsylvania is the home of the Pennsylvania State University and Bellefonte is one of its adjacent towns. Both of them can find web pages related to them. Our question is whether these two set of web pages have some kind of proximity in the web space because their subjects, namely, State College and Bellefonte, are adjacent to each other in the real world. Before we can validate it, however, we need to define the web proximity of two sets of web pages. One simple definition could be the percentage of web pages that overlap. Using this definition, our hypothesis basically says this: the closer two locations are, the more overlaps exist between web pages related

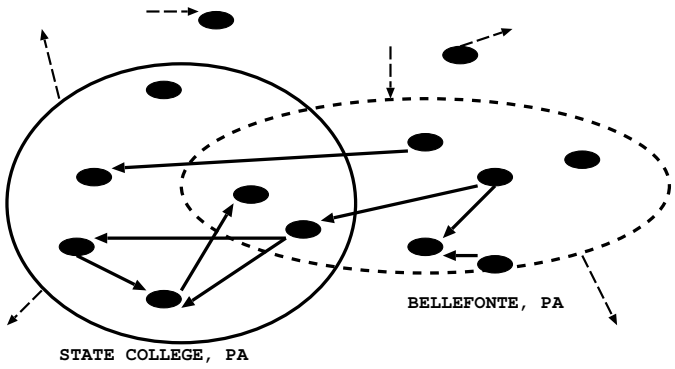



Fig. 6.3. Spatial proximity and Web proximity

to them. Provided our hypothesis I holds, we can elaborate this definition further: what percentage of overlapping web pages belong to the core components, belong to in components and out components, and belong to disconnected components? The drawback of this definition is that it overlooks non-overlapping web pages. Another possible definition of the web proximity is the link distances between these two sets of web pages. However, without constructing the link structure of the entire World Wide Web, we may not be able to have a meaningful measurement of their link distances.

**Hypothesis III: The implicit location-related information could be detected by using the explicit location-related information and the associated link structures.** In section 6.2, we have mentioned the explicit location information on the web could be detected by the name, zip code or area code of a location. Although some web pages don't have any explicit location information, they are actually related to some location. How to detect them, and include them in our crawling result becomes an interesting problem. As shown in Figure 6.4, we can view the direct neighbors of a web page as the web pages that have link to or from this web page.

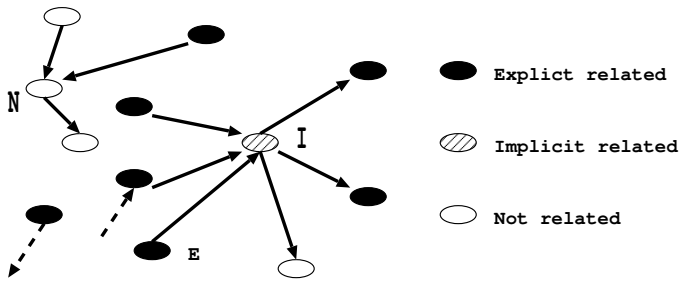



Fig. 6.4. Implicitly related web page

Suppose a web page has no explicit information about one particular location. But if X% of this web page's direct neighbor has explicit information on this location, we hypothesis that this given web page is implicitly related to the given location. The exact value for X needs to be determined in the experiments.

**Hypothesis IV: The update frequency of information related to one location is related to the social and economic factors of this location.** The web is a dynamic entity: in every second, there is new information added to it and obsolete information removed from it. Compared to the web information related to Bellefonte, Pennsylvania, we hypothesize that the web information related to New York City will be refreshed more frequently because New York City has a much bigger population body. Besides the population size, other social and economic factors could also influence the update frequency of location-related web information. For instance, although Reading, Pennsylvania is bigger than State College, Pennsylvania, we will hypothesize that the web information related to State College will be updated more often than the web information related to Reading. The reason being that State College is a college town and its population

accesses and utilizes the web more actively compared to the non-college town. If this hypothesis holds, based on the social and economical factors of one location, we can determine how often we need to run our geo-spatial crawler on it to keep the crawled information fresh. Of course, we also need to decide which social and economic factors are relevant here.

## 6.4 Future Work

This section briefly discusses our plan on gathering and processing the location-related information on the web. At the moment, we are implementing Diligenti's context focused crawler [28]. Using this focused crawler on various locations, we are hoping to validate the hypotheses discussed in section 6.3.

In addition, we want to study other characteristics of this location-related information. For instance, what is the top-level domain composition of these location-related information? Which domain will have a significant percentage, .com, .edu or .org? Do different locations have different top-level domain compositions or similar ones? Do these top-level domain composition resemble the top-level domain composition of the entire web?

More importantly, we hope to draw from these experiments some knowledge about geo-spatial presence on the web, and use them to build a new kind of focused crawler - geo-spatial crawler. Here are some thoughts on this matter.

- If the implicit location-related information could be detected as we hypothesized, we can use it to augment our crawling result. Since our detecting algorithm is only an approximation and not error free, we need to find a way to eliminate the undesired ones. Below, we detail a possible way to do so.

- If our hypothesis II holds, i.e., spatial proximity indeed implies web proximity, we could use it to remove some undesired crawling result. We start by choosing two far away places, like Los Angels, California and State College, Pennsylvania. After crawling on both of these places, we can detect whether there is an intersect between these two crawling result sets. If such an intersect exists, it needs to be eliminated from both crawling result sets. The reason is that if some web pages are related to two far away locations at the same time, they are most likely not specific to either one of them, and could be eliminated from the results.

- Hypothesis II could be used the other way around. If the location-related web pages are not well connected, finding DISCONNECT components and TENDRIL components of the location-related web pages could be difficult for the standard focused crawler. If spatial proximity implies web proximity, we could improve our possibilities to locate these components by focused crawling two adjacent locations together, such as State College and Bellefonte mentioned earlier. Although a web page in the DISCONNECT component does not have any link to web pages in the other components of the same result set, it may have paths to or from the result set of the adjacent location.

## 6.5 Chapter Summary

This chapter reports the current status of our ongoing work on a challenging problem, i.e., how to effectively retrieve web pages related to a given location. We have offered four hypotheses on location-related web pages, and plan to validate them in the following experiments.

The validated hypotheses could then be used to build a more specific geo-spatial crawler. Retrieving location-related information effectively is only the first step. How to classify them and rank them will be the next task. Moreover, by analyzing the web information related to one location, we are hoping to improve our understanding of its social and economic structures in this Internet era. For example, which kind of web pages have link to the local business? Can this information be used to identify potential customers?

# Chapter 7

# Conclusion

Spatial information has always been a need of human society, and its importance becomes even more apparent in this information age: numerous applications can benefit from utilizing it. Although Spatial Database Management Systems (SDBMS) have been developed to handle spatial information effectively to meet demands from various applications, the growing volume and complexity of spatial data, the rapid shift of the computing paradigm to a pervasive and ubiquitous one, the unexplored spatial information hosted by the World Wide Web and many other factors challenge researchers to study spatial information accessing and processing techniques in greater depth and breadth.

To meet these challenges, this thesis carefully examines various issues in accessing spatial information in both emerging resource-constrained and evolving resource-rich environments.

## 7.1 Summary of Contributions

In this thesis we have conducted five studies, and their main contributions are summarized here.

- Selectivity estimations for spatial selections and spatial joins were posted as open problems. Our first and second studies deliver satisfactory histogram-based solutions to these problems providing very good accuracies at very little cost.

- This thesis is perhaps the first one in the literature to investigate the energy behavior of memory-resident spatial index structure on handheld mobile devices, and opens the door for more research work in this domain.

- This thesis has developed various methods for distributing a R-Tree index structures across a network of workstations. The distributed R-Tree index structure has shown better performance than the stand alone R-Tree structure.

- Lastly, this thesis poses a novel problem: how to effectively retrieve spatial (location-related) information from the World Wide Web? By offering four intuitive hypotheses, we draw a road map to build a feasible solution - a geo-spatial crawler.

## 7.2 Future Directions

This thesis has studied accessing spatial information in resource-constrained and resource-rich environments. A natural question to ask next is what is the impact of the interactions between these environments on accessing spatial information. Can the wireless network techniques provide reliable and rapid means to transfer the spatial information back and forth between the resource-constrained and resource-rich environments? Can our accurate selectivity estimation technique be used in this new scenario?

For the resource-constrained environment, can we design an energy-conscious spatial index structure? In the resource-rich environment, can we develop a parallel spatial join algorithm based on our proposed R-Tree distribution techniques?

Of course, we are looking to complete our geo-spatial crawler project so that we can make sense of geo-spatial presence on the World Wide Web, and utilize them for further analyses.

Clearly, spatial information will always be a need for the human society, and new challenges in accessing and processing spatial information will always come along.

# References

[1] S. Acharya, V. Poosala, and S. Ramaswamy. Selectivity Estimation in Spatial Databases. In *Proceedings of the ACM SIGMOD*, pages 13–24, Philadephia, Pennsylvania, 1999.

[2] C. Aggarwal, F. Al-Garawi, and P. Yu. Intelligent Crawling on The World Wide Web with Arbitrary Predicates. In *Proceedings of 10th Interrnational WWW Conference*, pages 96–105, 2001.

[3] A. Ailamaki et al. DBMSs On a Modern Processor: Where Does Time Go? In *Proceedings of Very Large Databases Conference*, pages 266–277, Edinburgh, Scotland,, 1999.

[4] R. Alonso and H. F. Korth. Database System Issues in Nomadic Computing. In *Proceedings of the ACM SIGMOD Conference*, pages 388–392, Washington, D.C., 1993.

[5] J. Ambite, C. Knoblock, et al. The WorldInfo Assistant: Spatio-Temporal Information Integration on the Web. In *Proceedings of 27th International Conference on Very Large DataBases*, pages 717–718, 2001.

[6] N. An, S. Gurumurthi, et al. Energy-Performance Trade-offs for Spatial Access Methods on Memory-Resident Data. *To appear in the International Journal on Very Large Data Bases (invited submission)*, 11(3), 2002.

[7] N. An, J. Jin, and A. Sivasubramaniam. Towards an Accurate Analysis of Range Queries on Spatial Data. *To appear in IEEE Transactions on Knowledge and Data Engineering*.

[8] N. An, R. Lu, L. Qian, A. Sivasubramaniam, and T. Keefe. Storing Spatial Data on a Network of Workstations. *Cluster Computing, The Journal of Networks, Software Tools & Applications: Special Issue on I/O in Shared-Storage Clusters*, 2(4):259–270, 1999.

[9] N. An, L. Qian, et al. Evaluating Parallel R-Tree Implementations on a Network of Workstations. In *Proceedings of ACM Symposium on Advances in Geographical Information Systems (ACM-GIS)*, pages 159–160, 1998.

[10] N. An, A. Sivasubramaniam, et al. Analyzing Energy Behavior of Spatial Access Methods for Memory-Resident Data. In *Proceedings of International Conference on Very Large Data Bases*, pages 411–420, 2001.

[11] N. An, Z-Y. Yang, and A. Sivasubramaniam. Selectivity Estimation for Spatial Joins. In *Proceedings of IEEE International Conference on Data Engineering (ICDE)*, pages 368–375, 2001.

[12] W.G. Aref and H. Samet. Optimization for Spatial Query Processing. In *Proceedings of the International Conference on Very Large Data Bases*, pages 81–90, Barcelona, Spain, 1991.

[13] W.G. Aref and H. Samet. A Cost Model for Query Optimization Using R-Trees. In *Proceedings of the Second ACM Workshop on Advances in Geographic Information Systems*, pages 60–67, Gaithersburg, Maryland, November 1994.

[14] L. Arge et al. Scalable Sweeping-Based Spatial Join. In *Proceedings of 24th International Conference on Very Large Data Bases*, pages 570–581, New York City, New York, 1998.

[15] J. Shepherd B. Harangsri and A. Ngu. Selectivity estimation for joins using systematic sampling. In *Proceedings of Eighth International Workshop On Database And Expert System Applications*, pages 384–389, Toulouse, France, September 1997.

[16] A. Belussi and C. Faloutsos. Estimating the Selectivity of Spatial Queries Using the 'Correlation' Fractal Dimension. In *Proceedings of 21th International Conference on Very Large Data Bases*, pages 299–310, Zurich,Switzerland, 1995.

[17] A. Belussi and C. Faloutsos. Self-spacial join selectivity estimation using fractal concepts . *ACM Transactions on Information Systems*, 16(2):161–201, April 1998.

[18] J. L. Bentley. Multidimensional Binary Search Trees Used for Associative Searching. *Communications of the ACM*, 1975.

[19] K. Bharat and A. Broder. A technique for measuring the relative size and overlap of public Web search engines. In *Proceedings of 7th International WWW Conference*, 1998.

[20] N. J. Boden et al. Myrinet: A Gigabit-per-second Local Area Network. *IEEE Micro*, 15(1):29–36, 1995.

[21] P. A. Boncz et al. Monet And Its Geographic Extensions: a Novel Approach to High Performance GIS Processing. In *International Conference on Extending Database Technology*, pages 147–166, Avignon, France, 1996.

[22] T. Brinkhoff, H. Kriegel, and B. Seeger. Efficient Processing of Spatial Joins using R-trees. In *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data*, pages 237–246, Washington, D.C., May 1993.

[23] D. Burger and T. Austin. The SimpleScalar Tool Set, Version 2.0. Technical report, Computer Sciences Department, University of Wisconsin, June 1997.

[24] S. Chakrabarti, M. Berg, and B. Dom. Focused crawling: a new approach to topic-specific Web resource discovery. In *Proceedings of 8th International WWW Conference*, 1999.

[25] A. Chandrakasan and R. Brodersen. *Low Power Digital CMOS Design*. Kluwer Academic Publishers, 1995.

[26] V. Delaluz et al. DRAM energy management using software and hardware directed power mode control. In *Proceedings of the International Conference on High Performance Computer Architecture (HPCA)*, Monterrey, Mexico, 2001.

[27] V. Delaluz et al. Hardware and software techniques for controlling dram power modes. *IEEE Transactions on Computers*, 50(11):1154 –1173, 2001.

[28] M. Diligenti, F. Coetzee, et al. Focused Crawling Using Context Graphs. In *Proceedings of International Conference on Very Large DataBases*, pages 527–534, 2000.

[29] J. Ding, L. Gravano, and N. Shivakumar. Computing Geographical Scopes of Web Resources. In *Proceedings of 26th International Conference on Very Large Data Bases*, pages 545–556, 2000.

[30] IEEE 802.11 Draft. Wireless lan media access control (mac) and physical layer (phy) specifications, May 1996.

[31] D. Duarte et al. Formulation and validation of an energy dissipation model for clock generation circuitry and distribution network. In *Proceedings of the International Conference on VLSI Design*, Bangalore, India, 2001.

[32] C. Faloutsos and I. Kamel. Beyond Uniformity and Independence: Analysis of R-trees Using the Concept of Fractal Dimension. In *Proceedings of the 13th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 4–13, Minneapolis,Minnesota, 1994.

[33] C. Faloutsos and S. Roseman. Fractals for secondary key retrieval. In *the eighth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems(PODS)*, pages 247–252, Philadelphia, PA, March 1989. ACM Press.

[34] C. Faloutsos, B. Seeger, A. Traina, and C. Traina. Spatial Join Selectivity Using Power Laws. In *Proceedings of the 2000 ACM SIGMOD Intl. Conference on Management of Data*, page to appear, Dallas, Texas, May 2000. ACM Press.

[35] C. Faloutsos, T.K. Sellis, and N. Rousspoulosand. Analysis of Object Oriented Spatial Access Methods. In *Proceedings of the 1987 ACM-SIGMOD conference*, pages 426–439, San Francisco, California, 1987.

[36] R. A. Finkel and J. L. Bentley. Quad Trees - A Data Structure for Retrieval on Composite Keys. *Acta Informatica 4*, 1974.

[37] M. Freeston. The bang file: a new kind of grid file. In *Proceedings of the 1987 ACM-SIGMOD Conference*, pages 260–269, San Francisco, California, 1987.

[38] X. Fu et al. GPR-Tree: A Global Parallel Index Structure for Multiattribute Declustering on Cluster of Workstations. In *Proceedings of the Conference on Advances in Parallel and Distributed Computing*, pages 300–306, Shanghai, China, 1997.

[39] V. Gaede and O. Gunther. Multidimensional Access Methods. *ACM Computing Surveys*, 30(2):170–230, 1998.

[40] P. Gauthier, D. Harada, and M. Stemm. Reducing Power Consumption for the Next Generation of PDAs: It's in the Network Interface. In *Proceedings of the International Workshop on Mobile Multimedia Communications (MoMuC)*, September 1996.

[41] GEOPlace.Com. Mobile Technology Takes GIS to the Field. http://www.geoplace.com/gw/2000/0600/0600IND.ASP.

[42] S. Ghandeharizadeh et al. A Performance Analysis of Alternative Multiattribute Declustering Strategies. In *Proceedings of the ACM SIGMOD Conference*, pages 29–38, San Diego, California, 1992.

[43] K. Ghose and M. B. Kamble. Reducing power in superscalar processor caches using subbanking, multiple line buffers, and bit-line segmentation . In *Proceedings of the International Symposium Low Power Electronics and Design(ISLPED)*, pages 70–75, San Diego, California, 1999.

[44] A. Gonzales et al. A data cache with multiple caching strategies tuned for different types of locality. In *Proceedings of the International Conference on Supercomputing*, pages 338–347, Barcelona, Spain, 1995.

[45] J.G. Griffiths. An Algorithm for Displaying a Class of Space-filling Curves. *Software - Practice and Experience (SPE)*, pages 403–411, 1986.

[46] 0. Gunther. The Design of the Cell Tree: An Object-Oriented Index Structure for Geometric Databases. In *Proceedings of the International Conference on Data Engineering*, pages 598–605, Los Angeles, CA, 1989.

[47] S. Gurumurthi, N. An, et al. Energy and Performance Considerations in Work Partitioning for Mobile Spatial Queries. Technical Report CSE-01-028, The Pennsylvania State University, November 2001.

[48] A. Guttman. R-trees: A Dynamic Index Structure for Spatial Searching. In *Proceedings of the 1984 ACM-SIGMOD Conference*, boston, massachusetts, June 1984.

[49] P. J. Haas, J. F. Naughton, S. Seshadri, and A. N. Swami. Fixed-precision estimation of join selectivity. In *Proceedings of the Twelfth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 190–201, Washington, DC, May 1993.

[50] P. J. Haas, J. F. Naughton, and A. N. Swami. On the relative cost of sampling for join selectivity estimation. In *Proceedings of the Thirteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 14–24, Minneapolis, Minnesota, May 1994.

[51] P. J. Haas and A. N. Swami. Sampling-based selectivity estimation for joins using augmented frequent value statistics. In *Proceedings of the Eleventh International Conference on Data Engineering*, pages 522–531, Taipei, Taiwan, March 1995.

[52] A. Henrich, H.W. Six, and P. Widmayer. The LSD tree: spatial access to multidimensional point and non-point data. In *Proceedings of the International Conference on Very Large Databases*, pages 45–53, Amsterdam, The Netherlands, 1989.

[53] K. Hinrichs and J. Nievergelt. The grid file: a data structure to support proximity queries on spatial objects. In *Proceedings of the WG'83 Intern. Workshop on Graph Theoretic Concepts in Computer Science*, pages 100–113, 1983.

[54] E. G. Hoel and H. Samet. Efficient Processing of Spatial Queries in Line Segment Databases. In *Proceedings of the Symposium on Advances in Spatial Databases(SSD)*, pages 237–256, Zurich, Switzerland, 1991.

[55] E. G. Hoel and H. Samet. A Qualitative Comparison Study of Data Structures for Large Line Segment Databases. In *Proceedings of the ACM SIGMOD*, pages 205–214, San Diego, California, 1992.

[56] Y. W. Huang, N. Jing, and E. A. Rundensteiner. A Cost Model for Estimating the Performance of Spatial Joins Using R-trees. In *Proceedings of Statistical and Scientific Database Management*, pages 30–38, Olympia, Washington, 1997.

[57] Y.W. Huang, N. Jing, and E. A. Rundensteiner. Spatial Join Using R-tree: Breadth-First Traversal With Global Optimizations. In *Proceedings of the 24th Very Large Data Base Conference*, pages 396–405, 1997.

[58] T. Imielinski, S. Viswanathan, and B. R. Badrinath. Energy Efficient Indexing on Air. In *Proceedings of the ACM SIGMOD*, pages 25–36, Minneapolis, Minnesota, 1994.

[59] K. Inoue, T.Ishihara, and K. Murakami. Way-Predicting Set-Associative Cache for High Performance and Low Energy Consumption. In *Proceedings of the International Symposium on Low Power Electronics and Design*, pages 273–275, 1999.

[60] Environmental Systems Research Institute. Digital Chart of the World. http://www.maproom.psu.edu/dcw/.

[61] Intel StrongARM SA-1110 Microprocessor Brief Datasheet. http://developer.intel.com/design/strong/datashts/278241.htm.

[62] Y. Ioannidis and V. Poosala. Histogram-Based Solutions to Diverse Database Estimation Problems. *IEEE Data Engineering*, 18:10–18, 1995.

[63] M.J. Irwin, M. Kandemir, et al. A Holistic Approach to System Level Energy Optimization. In *Proceedings of the International Workshop on Power and Timing Modeling, Optimization, and Simulation*, Gottingen, Germany, 2000.

[64] H.V. Jagadish. Linear Clustering of Objects with Multiple Atributes. In *Proceedings of the ACM SIGMOD*, pages 332–342, Atlantic City, NJ, 1990.

[65] J. Jin. Techniques for Analyzing Range Queries on R-Trees. Master's thesis, Dept. of Computer Science & Engineering, The Pennsylvania State University, May 1999.

[66] J. Jin, N. An, and A. Sivasubramaniam. Analyzing Range Queries on Spatial Data. In *Proceedings of the International Conference on Data Engineering*, pages 525–534, March 2000.

[67] I. Kamel and C. Faloutsos. Parallel R-Trees. In *Proceedings of the ACM SIGMOD Conference*, pages 195–204, San Diego, CA, 1992.

[68] I. Kamel and C. Faloutsos. On Packing R-trees. In *Proceedings of the 2nd ACM International Conference on Information and Knowledge Management*, pages 490–499, November 1993.

[69] S. Kim, N. Vijaykrishnan, et al. Power-aware Partitioned Cache Architectures. In *Proceedings of the ACM/IEEE International Symposium on Low Power Electronics and Design(ISLPED'01)*, pages 64–67, August 2001.

[70] N. Koudas et al. Declustering spatial databases on a multi-computer architecture. In *International Conference on Extending Database Technology*, pages 592–614, Avignon, France, 1996.

[71] S. Kumar, P. Raghavan, et al. The Web as a Graph. In *Symposium on Principles of Database Systems*, pages 1–10, 2000.

[72] Steve Lawrence and C. Lee Giles. Searching the World Wide Web. *Science*, 280(5360):98–100, 1998.

[73] T. Lehman and M. J. Carey. Query Processing in Main Memory Database Management Systems. In *Proceedings of the ACM SIGMOD Conference*, pages 239–250, Seattle, Washington, 1998.

[74] S. T. Leutenegger and M. A. Lopez. The Effect of Buffering on the Performance of R-Trees. In *Proceedings of the 14th International Conference on Data Engineering*, pages 164–171, Orlando, Florida, 1998.

[75] M. L. Lo and C. V. Ravishankar. The Design and Implementation of Seeded Trees: An Efficient Method for Spatial Joins. *IEEE Transactions on Knowledge and Data Engineering*, 10(1):136–151, 1998.

[76] D. B. Lomet and B. Salzberg. The hB-tree: A Multiattribute Indexing Method with good Guaranteed Performance. *ACM Transactions on Database Systems*, 1998.

[77] N. Mamoulis and D. Papadias. Integration of Spatial Join Algorithms for Processing Multiple Inputs. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 1–12, philadelphia, pennsylvania, June 1999.

[78] D. J. Mavriplis. An Advancing Front Delaunay Triangulation Algorithm Designed for Robustness. *Journal of Computational Physics*, 117:90–101, 1995.

[79] K. McCurley. Geospatial Mapping and Navigation of the Web. In *Proceedings of the 10th International WWW Conference*, pages 221–229, 2001.

[80] Microsoft. Microsoft Pocket Streets. http://www.microsoft.com/mobile/downloads/streets.asp.

[81] M. Muralikrishna and D.J. DeWitt. Equi-Depth Histograms For Estimating Selectivity Factors For Multi-Dimensional Queries. In *Proceedings of the ACM SIGMOD*, pages 28–36, Chicago, Illinois, 1988.

[82] N. Beckmann and others. The R*-tree: An Efficient and Robust Access Method for points and Rectangles. In *Proceedings of the ACM SIGMOD Conference*, pages 322–331, Atlantic City, New Jersey, 1990.

[83] U. S. Bureau of the Census. TIGER/Line(R) 1995 Data. http://www.esri.com/data/online/tiger/index.html. last visited: Feb. 10, 2000.

[84] M. A. Olson. Selecting and Implementing an Embedded Database System. *Computer*, 33(9):27–34, 2000.

[85] J. A. Orenstein. Spatial Query Processing in an Object-Oriented Database System. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 326–336, Washington, D.C., May 1986.

[86] S. Ortiz. Embedded Databases Come out of Hiding. *Computer*, 33(3):16–19, 2000.

[87] B. Pagel, H-W. Six, H. Toben, and P. Widmayer. Towards an Analysis of Range Query Performance in Spatial Data Structures. In *Proceedings of the 12th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 214–221, Washington,DC, 1993.

[88] D. Papadias, N. Mamoulis, and V. Delis. Algorithms for Querying by Spatial Structure. In *Proceedings of Very Large Databases Conference*, pages 546–557, 1998.

[89] A. Papadopoulos and Y. Manolopoulos. Performance of Nearest Neighbor Queries in R-Trees. In *Proceedings of International Conference on Database Theory*, pages 394–408, Delphi, Greece, 1997.

[90] J. M. Patel. *Efficient Database Support for Spatial Applications*. PhD thesis, University of Wisconsin-Madison, 1998.

[91] J. M. Patel and D. J. DeWitt. Partition Based Spatial Merge Join. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 259–270, Montreal, Canada, June 1996.

[92] J. M. Patel et al. Building a Scalable Geo-Spatial DBMS: Technology, Implementation, and Evaluation. In *Proceedings of the ACM-SIGMOD Conference*, pages 336–347, Tucson, Arizona, 1997.

[93] V. Poosala. *Histogram-based Estimation Techniques in Databases*. PhD thesis, Univ. of Wisconsin-Madison, 1997.

[94] International Research Institute For Climate Prediction.

[95] F. P. Preparata and M. I. Shamos. *Computational Geometry: An Introduction*. Springer-Verlag, 1985.

[96] G. Proietti and C. Faloutsos. I/O Complexity for Range Queries on Region Data Stored Using an R-tree. In *Proceedings of the 15th International Conference on Data Engineering*, pages 628–635, Syndey, Austrialia, 1999.

[97] J. T. Robinson. The k-d-b-tree: a search structure for large multidimensional dynamic indexes. In *Proceedings of the ACM SIGMOD Conference*, pages 10–18, 1981.

[98] N. Roussopoulos et al. Nearest Neighbor Queries. In *Proceedings of the ACM SIGMOD*, pages 71–79, San Jose, California, 1995.

[99] N. Roussopoulos and D. Leifker. Direct Spatial Search on Pictorial Databases Using Packed R-Trees. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 17–31, Austin, Texas, 1985.

[100] K. Roy and M. C. Johnson. Software Design for Low Power. *NATO Advanced Study Institute on Low Power Design in Deep Sub-Micron Electronics*, 1996.

[101] K. Roy and S. C. Prasad. *Low-Power CMOS VLSI Circuit Design*. John Wiley & Sons, Inc, 2000.

[102] H. Samet. *The Design and Analysis of Spatial Data Structures*. Addison-Wesley, 1990.

[103] B. Schnitzer and S. T. Leutenegger. Master-Client R-trees: A New Parallel R-tree Architecture. In *Statistical and Scientific Database Management*, pages 68–77, Cleaveland, Ohio, 1999.

[104] B. Seeger. Performance Comparison of Segment Access Methods Implemented on Top of the Buddy-Tree. In *Proceedings of the International Symposium on Advances in Spatial Databases (SSD)*, pages 227–296, Zurich, Switzerland, 1991.

[105] B. Seeger and H-P. Kriegel. The Buddy-Tree: An Efficient and Robust Access Method for Spatial Data Base Systems. In *Proceedings of the International Conference on Very Large Data Bases*, pages 590–601, Queensland, Australia, 1990.

[106] T.K. Sellis et al. The R+-Tree: A Dynamic Index for Multi-Dimensional Objects. In *Proceedings of the International Conference on Very Large Data Bases*, pages 507–518, Brighton, England, 1987.

[107] G.P. Shapiro and C. Connel. Accurate etimation of the number of tuples satisfing a condition. In *Proceedings of the ACM SIGMOD Conference*, pages 256–276, Boston, Massachusetts, 1984.

[108] S. Shekhar, S. Chawla, S. Ravada, et al. Spatial Databases - Accomplishments and Research Needs. *IEEE Transactions on Knowledge and Data Engineering*, 11(1):45–55, 1999.

[109] S. Shekhar et al. Parallelizing a GIS on a Shared Address Space Architecture. *IEEE Computer Magzine*, 1996.

[110] S. Shekhar et al. Declustering and Load-balancing Methods for Parallelizing Geographic Information Systems. *IEEE Transactions on Knowledge and Data Engineering*, 1998.

[111] E. Shih, S-H. Cho, et al. Physical Layer Driven Protocol and Algorithm Design for Energy-Efficient Wireless Sensor Networks. In *Proceedings of the ACM SIGMOBILE Conference of Mobile Computing and Networking (MOBICOM)*, 2001.

[112] W-T. Shiue and C. Chakrabarti. Memory exploration for low power, embedded systems. Technical Report Technical Report CLPE-TR-9-1999-20, Arizona State University, 1999.

[113] P. Song. Embedded DRAM Finds Growing Niche. *Microprocessor Report*, August 1997.

[114] Special Issue on Parallel Processing in GIS. International Journal of Geographical Information Systems, 10(6), 1996.

[115] M. Stonebraker et al. The Sequoia 2000 Benchmark. In *the ACM SIGMOD International Conference on Management of Data*, pages 2–11, Washington, D.C., May 1993.

[116] Y. Theodoridi and D. Papadias. Range Queries Involving Spatial Relations: A Performance Analysis. In *Proceedings of Conference on Spatial Information Theory*, pages 537–551, Semmering, Austria, 1995.

[117] Y. Theodoridis and T. Sellis. A Model for the Prediction of R-tree Performance. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 161–171, Montreal, Canada, June 1996.

[118] Y. Theodoridis, E. Stefanakis, and T. Sellis. Cost Models for Join Queries in Spatial Databases. In *Proceedings of the International Conference on Data Engineering*, pages 476–483, February 1998.

[119] Y. Theodoridis, E. Stefanakis, and T. K. Sellis. Efficient Cost Models for Spatial Queries Using R-Trees. *IEEE Transactions on Knowledge and Data Engineering*, 12(1):19–32, 2000.

[120] N. Vijaykrishnan et al. Energy-driven integrated hardware-software optimizations using SimplePower. In *Proceedings of the International Symposium on Computer Architecture*, pages 95–106, 2000.

[121] R. Viswanath et al. Thermal performance challenges from silicon to systems. *Intel Technology Journal*, Q3, 2000.

[122] T. von Eicken et al. U-Net: A User-Level Network Interface for Parallel and Distributed Computing. In *ACM Symposium on Operating System Principles*, pages 40–53, Copper Mountain Resort, Colorado, 1995.

[123] T. Shimizu Y. Nunomura and O. Tomisawa. M32RD-Integrating DRAM and Microprocessor. *IEEE Micro*, 17(6):40–48, November/December 1997.

# Vita

Ning An was born in Lanzhou, Gansu, P.R.China that is at the foot of Gaolan Mountain, runs along the bank of Yellow River and lays on the outskirts of Gabi Desert. After spending his joyful childhood and most of his restless teens, he remained in his hometown and was enrolled in Lanzhou University in 1989. In college, Ning tried hard to become a top student, but was not very successful. He did, however, manage to stay at Lanzhou University for his Master of Science degree that was awarded in 1993.

Later, in the same year, Ning travelled far away from home, and started his Ph. D. study at the Department of Computer Science and Engineering, The Pennsylvania State University. Since then, he has worked on campus as a Teaching Assistant, a Research Assistant, and even a Graduate Lecturer. Intermittently, he spent the summer as an intern in the industry: once was in 1998 at Environmental Systems Research Institute (ESRI), Redlands, California and the other one was in 2001 at Scientific Research Lab, Ford Motor Company, Dearborn, Michigan.

Ning's research interests are in the areas of Spatial Database Systems, Location-based Computing and Web Computing.