ONLINE QUERY PROCESSING IN GEOGRAPHIC INFORMATION SYSTEMS

A Dissertation
Presented to
the Faculty of Engineering and Computer Science
University of Denver

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy

by
Wan D. Bae
November 2007

Advisor: Prof. Scott T. Leutenegger
Co-advisor: Prof. Seon Ho Kim
GRADUATE STUDIES

AT

THE UNIVERSITY OF DENVER

Upon the recommendation of the chairperson of the Department of Computer Science this dissertation is hereby accepted in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

______________________________
Professor in charge of dissertation

______________________________
Vice Provost for Graduate Studies

______________________________
Date
Contents

1 Introduction ...................................................... 1
  1.1 Incremental Refining Spatial Joins for Vector Data .............. 6
  1.2 An Interactive Framework for Raster Data Spatial Joins .......... 9
  1.3 Range Queries on Web Data using k-Nearest Neighbor Search .... 11
  1.4 Contributions ............................................... 13
  1.5 Organization of Dissertation .................................. 14

2 Literature Survey .............................................. 15
  2.1 Geographic Information Systems (GIS) .......................... 15
     2.1.1 GIS Data and Applications .................................. 16
     2.1.2 Spatial Queries in GIS ..................................... 18
     2.1.3 Spatial Access Methods and Data Structures ................ 21
  2.2 Statistical Techniques in Databases ............................ 23
     2.2.1 Statistical Techniques for Spatial Joins .................... 23
     2.2.2 Random Sampling and Probabilistic Query Techniques ........ 26
  2.3 Online Query Processing ...................................... 28

3 Incremental Refining Spatial Joins for Vector Data ................ 30
  3.1 Introduction ............................................... 30
     3.1.1 Notations ................................................. 31
  3.2 Overview of Incremental Refining Spatial Joins .................. 31
  3.3 Incremental Refining Spatial Join (IRS J) Algorithms ............. 33
     3.3.1 IRS J_i ("i" for "tuple") .................................. 34
     3.3.2 IRS J_s ("s" for "stratified") .............................. 35
     3.3.3 IRS J_p ("p" for "page") ................................... 39
CONTENTS

3.4 Statistics for the IRS J Algorithms .......................... 40
  3.4.1 Population Proportion and Confidence Intervals ....... 40
  3.4.2 The Confidence Interval for IRS J_t ....................... 43
  3.4.3 The Confidence Interval for IRS J_k ....................... 44
  3.4.4 The Confidence Interval for IRS J_p ....................... 47

3.5 Experiments .............................................. 48
  3.5.1 Data sets and Experimental Methodology ................. 49
  3.5.2 Accuracy of Estimates .................................. 50
  3.5.3 Time for Confidence Interval ............................ 54
  3.5.4 Index structures and IRS J Performance ................. 59

3.6 Summary ................................................. 62

4 An Interactive framework for Raster Data Spatial Joins 64
  4.1 Introduction ............................................ 64
     4.1.1 Notations ............................................ 66
  4.2 A Framework for Spatial Joins Over Raster Data ............ 66
     4.2.1 Augmented Quad-tree ................................ 66
     4.2.2 Probabilistic Joins vs. Random Sampling ............... 67
     4.2.3 Framework Overview ................................ 68
  4.3 Probabilistic Join (PJ) .................................... 70
  4.4 Incremental Stratified Sampling Join (ISSJ) ............... 74
     4.4.1 Stratified Random Sampling ............................ 74
     4.4.2 Estimates for Stratified Random Sampling ............... 77
  4.5 Experiments .............................................. 78
     4.5.1 Data Sets and Experimental Methodology ............... 78
     4.5.2 Experimental Results ................................ 79
  4.6 Summary ................................................. 86

5 Spatial Range Queries on Web Data using k-Nearest Neighbor Search 87
  5.1 Introduction ............................................ 87
     5.1.1 Notations ............................................ 89
  5.2 Range Queries using fixed values of k: QDD and DCDT Algorithms 90
     5.2.1 Quad Drill Down (QDD) Algorithm ..................... 92
     5.2.2 Dynamic Constrained Delaunay Triangulation (DCDT) Algorithm 98
  5.3 Range Queries using flexible values of k: DB Algorithm .... 108
     5.3.1 Algorithm of DB ...................................... 109
     5.3.2 Statistical Analysis of DB ............................. 111
  5.4 Hybrid (HB) Algorithm ................................... 112
  5.5 Experiments .............................................. 114
     5.5.1 Results of QDD and DCDT .............................. 115
CONTENTS

5.5.2 Results of the DB Algorithm ............................................. 117
5.6 Summary .............................................................................. 119

6 Conclusions ........................................................................... 121
6.1 Summary of Results ............................................................... 121
6.2 Future Research Work ............................................................... 123

Bibliography .............................................................................. 125
# List of Tables

3.1 Notations used in *IRS J* .................. 31  
3.2 Index size of the synthetic data sets in pages (of 4KB) ................. 51  
3.3 Index size of the real data sets in pages (of 4KB) ........................ 51  
3.4 I/Os and node accesses of *IRS J*, and a full R-tree join for the real data sets:  
   C.I.= Confidence Interval, Buffer size=Percent of relation size (indexed  
   by R-tree) ................................................. 58  
4.1 Notations used in *PJ* and *ISS J* ........................................... 66  
4.2 Synthetic data sets: N.E. = non empty data cells .............................. 79  
4.3 Real data sets: N.E. = non empty data cells .................................. 79  
4.4 Example of a lookup table for 2-d join probability ......................... 80  
4.5 1-d vs. 2-d Probabilistic Joins ................................................. 80  
4.6 *PJ* (4th level) vs. actual joins ................................................ 81  
5.1 Notations for range queries using k-NN search ......................... 89  
5.2 Statistical analysis of *QDD*: 4K uniform synthetic data set ........... 97  
5.3 The average percentage reduction in the number of k-NN calls (%) .... 117
List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>Examples of georeferenced data sets and their spatial joins [48]</td>
<td>3</td>
</tr>
<tr>
<td>1.2</td>
<td>Examples of vector data sets: natural resources in Colorado in the U.S.</td>
<td>8</td>
</tr>
<tr>
<td>1.3</td>
<td>Examples of raster data sets: 256×256 cells</td>
<td>10</td>
</tr>
<tr>
<td>1.4</td>
<td>A range query on Web data</td>
<td>12</td>
</tr>
<tr>
<td>3.1</td>
<td>Block diagram of the Incremental Refining Spatial Join (IRS J) process</td>
<td>32</td>
</tr>
<tr>
<td>3.2</td>
<td>An example of the Window Query</td>
<td>33</td>
</tr>
<tr>
<td>3.3</td>
<td>An example of strata and clusters in the R-tree</td>
<td>35</td>
</tr>
<tr>
<td>3.4</td>
<td>Synthetic data sets distribution</td>
<td>49</td>
</tr>
<tr>
<td>3.5</td>
<td>Real data sets</td>
<td>50</td>
</tr>
<tr>
<td>3.6</td>
<td>Estimates and Actual deviation for synthetic data sets: ( U-800K \bowtie S-800K ) (indexed by R-tree)</td>
<td>52</td>
</tr>
<tr>
<td>3.7</td>
<td>Estimates and Actual deviation for the real data sets (indexed by R-tree)</td>
<td>53</td>
</tr>
<tr>
<td>3.8</td>
<td>Confidence intervals and I/Os for synthetic ( U-800K \bowtie S-800K ) and real data sets</td>
<td>56</td>
</tr>
<tr>
<td>3.9</td>
<td>I/O comparison of IRS J, and a R-tree join for varying size of synthetic data sets with 10% buffer size: ( R(\text{uniform}) \bowtie S(\text{skewed}) ) (indexed by R-tree)</td>
<td>57</td>
</tr>
<tr>
<td>3.10</td>
<td>Node access comparison of IRS J, and a R-tree join for varying size of synthetic data sets with 10% buffer size: ( R(\text{uniform}) \bowtie S(\text{skewed}) ) (indexed by R-tree)</td>
<td>57</td>
</tr>
<tr>
<td>3.11</td>
<td>I/O comparison of IRS J, and a R-tree join for varying size of synthetic data sets with a fixed buffer size 1200: ( R(\text{uniform}) \bowtie S(\text{skewed}) ) (indexed by R-tree)</td>
<td>58</td>
</tr>
<tr>
<td>3.12</td>
<td>I/O and node access comparison of IRS J, and a R-tree join for real data sets w/10% buffer (indexed by R-tree)</td>
<td>59</td>
</tr>
<tr>
<td>3.13</td>
<td>I/Os of IRS J for 5% confidence interval in STR, R*-tree and R-tree</td>
<td>60</td>
</tr>
</tbody>
</table>
### LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.14</td>
<td>I/Os of IRS $J$ comparison of STR, R*-tree and R-tree</td>
<td>61</td>
</tr>
<tr>
<td>3.15</td>
<td>I/O ratio ($\text{IRS}<em>{J_1}/\text{IRS}</em>{J_2}$) for STR, R*-tree, and R-tree</td>
<td>62</td>
</tr>
<tr>
<td>4.1</td>
<td>Raster cells of Datasets $R$ and $S$</td>
<td>65</td>
</tr>
<tr>
<td>4.2</td>
<td>Examples of augmented quad-tree of data sets $R$ and $S$</td>
<td>67</td>
</tr>
<tr>
<td>4.3</td>
<td>A framework for raster joins</td>
<td>69</td>
</tr>
<tr>
<td>4.4</td>
<td>Join probability</td>
<td>70</td>
</tr>
<tr>
<td>4.5</td>
<td>Join of two intervals</td>
<td>71</td>
</tr>
<tr>
<td>4.6</td>
<td>Example of $1 - d$ join probability</td>
<td>72</td>
</tr>
<tr>
<td>4.7</td>
<td>Estimates, confidence intervals and I/Os of $ISS_J$: synthetic (uni1 $\Rightarrow$ exp1) and real data sets (unconsolidated sediment $\Leftarrow$ mineral in CO)</td>
<td>82</td>
</tr>
<tr>
<td>4.8</td>
<td>Real data sets: Mineral resources in Colorado in the U.S.</td>
<td>83</td>
</tr>
<tr>
<td>4.9</td>
<td>$ISS_J$ vs. actual joins vs. $PJ$ for real data sets in CO</td>
<td>84</td>
</tr>
<tr>
<td>4.10</td>
<td>Number of I/Os of $PJ$, $ISS_J$ and the full quad-tree join</td>
<td>85</td>
</tr>
<tr>
<td>5.1</td>
<td>Range queries on Web data</td>
<td>88</td>
</tr>
<tr>
<td>5.2</td>
<td>A range query using 5-NN search</td>
<td>90</td>
</tr>
<tr>
<td>5.3</td>
<td>A Quad Drill Down range query</td>
<td>93</td>
</tr>
<tr>
<td>5.4</td>
<td>Data distribution in a query region</td>
<td>96</td>
</tr>
<tr>
<td>5.5</td>
<td>Example of Constrained Delaunay triangulation</td>
<td>98</td>
</tr>
<tr>
<td>5.6</td>
<td>An example of first $k$-NN call in query region $R$</td>
<td>101</td>
</tr>
<tr>
<td>5.7</td>
<td>Example of pList, cEdges and tList: covering $\Delta_{max}$</td>
<td>103</td>
</tr>
<tr>
<td>5.8</td>
<td>Example of checkCoverRegion()</td>
<td>104</td>
</tr>
<tr>
<td>5.9</td>
<td>Examples of the cases for covered region $R_c$</td>
<td>105</td>
</tr>
<tr>
<td>5.10</td>
<td>Example of $DCDT$ w/k=15 (coverage rate)</td>
<td>107</td>
</tr>
<tr>
<td>5.11</td>
<td>Data distribution of the number of points in a query region</td>
<td>111</td>
</tr>
<tr>
<td>5.12</td>
<td>Number of $k$-NN calls for 4K synthetic data set with 3% range query</td>
<td>116</td>
</tr>
<tr>
<td>5.13</td>
<td>Number of $k$-NN calls for real data set with 3% range query</td>
<td>116</td>
</tr>
<tr>
<td>5.14</td>
<td>Percentage of coverage</td>
<td>117</td>
</tr>
<tr>
<td>5.15</td>
<td>Percentage of $k$-NN calls of $DB$</td>
<td>118</td>
</tr>
<tr>
<td>5.16</td>
<td>Accuracy of $DB$</td>
<td>118</td>
</tr>
</tbody>
</table>
Acknowledgements

I would like to thank my advisors, Prof. Scott Leutenegger and Prof. Seon Ho Kim, for their constant encouragement and support. They guided me and taught me to penetrate into the problem and to identify the variations of the problems.

I would also like to thank Prof. Petr Vojtěchovský who introduced me to the field of mathematical statistics and provided invaluable discussions.

Special thank must go to Prof. Cyrus Shahabi who brought the problem “Range queries on Web data using $k$-Nearest Neighbor search” and provided us with indispensable comments and support.

I also wish to thank the members of my dissertation committee: Prof. Joel Cohen and Prof. Roger Salters, for patiently reading my dissertation and providing many useful comments.

I must thank Dr. Sada Narayanappa, my friend who was always present to help me whenever I encountered problems during my research. I am thankful to my colleagues, Shayma Alkobaisi and Brandon Haenlein, for their continued discussions and invaluable feedback. Without their constant support and invaluable friendship, I would never have completed my dissertation.

Finally, I wish to express the deepest appreciation to my family for the patience, support and love during the years.
Chapter 1

Introduction

Geographic Information Systems (GIS) support collecting, storing, retrieving, and analyzing georeferenced data [28]. During the last decade, GIS have been widely used in many applications for exploring large georeferenced data in order to obtain intuition and insight into the stored information. Examples of such applications include weather forecasts, environmental controls (e.g., air dispersion modeling), public health and safety (e.g., disease surveillance and fire monitoring), traffic controls (e.g., aviation route), location-based commercial services, and homeland security. Due to the recent advances in georeferenced data acquisition and the emergence of diverse Web applications, a large amount of georeferenced data has become available on the Web. For example, numerous businesses release the locations of their branches on the Web. The Web sites of government organizations such as the US Postal Office provide a list of their offices close to one’s residence. Various non-profit organizations also publicly post a large amount of georeferenced data for different purposes.

Data analysis is frequently used in many applications for efficient decision support. Queries for data analysis consist of a complex sequence of spatial queries such as spatial joins and range queries of large amounts of georeferenced data sets. Due to the large size of the data sets and complicated nature of spatial query execution, evaluating GIS decision
support queries is a time-consuming process. These queries often run for many hours or days to obtain exact answers.

In many GIS applications, query response time is critical, and hence obtaining an approximate query result in a reasonably short time is far more important than evaluating an exact answer that takes a long time. For example, real-time air traffic control systems need to obtain route planning as quickly as possible. The systems need to continuously monitor changes in the weather to report the regions that could be affected by severe weather and to make timely decisions for safe route planning. Current query processing techniques may not be directly applicable for such decision support systems. In typical GIS, spatial queries are processed in a blocking manner where users have no control over the query execution. Users merely submit queries and then must wait for a long time without seeing any results until a final answer is returned. Consider the following query that retrieves all U.S. airports in severe weather:

```
Query 1:
SELECT Count (a.name)
FROM   airport a
WHERE  Intersect (a.geom, (  
    SELECT  r.geom ∩ w.geom  
    FROM     rainfall r, wind w  
    WHERE   Intersect (r.geom, w.geom) = 'TRUE' and  
             r.rate > 1.0 and w.speed > 55.0 )) = 'TRUE'
GROUP BY a.state
```

Query 1 provides information of airports that are affected by severe weather conditions and can be used for decision making process by a traffic control system. Figure 1.1 (a) and (b) show the weather data sets for “rainfall” and “wind”, respectively. Figure 1.1 (c) illustrates their spatial joins with the airport data set, and the final query result is displayed in Figure 1.1 (d). In a traditional query processing system, one possible query plan would be Query 1, which scans the entire data sets of “rainfall” and “wind” to obtain subsets
of the entries that satisfy the given predicates: “rainfall.rate > 1.0” and “wind.speed > 55.0”. This query then spatially joins the two subsets and finds intersecting regions of the two data sets as a set of polygons of rainfall.geom ∩ wind.geom, which represent severe weather regions. Next, the airport data set and the obtained intersecting regions are joined to retrieve only the airports affected by severe weather. Finally, the resulting airports are grouped by state, and the number of airports in each state is returned to the users with their visualized locations. The users of the above system need to wait until it completes processing the query and returns a final result. Note that query optimization may choose a different plan, but regardless of plan users must wait for a final answer. A large number
of data sets are involved in the decision of route planning and hence, it could take the systems hours to find possibly affected regions. The returned exact result may be no longer valid due to sudden changes in weather conditions. This is why blocking query processing techniques are not suitable for many GIS applications that require timely decisions. On the contrary, online query processing deals with exploratory queries through the Internet and can offer flexible and interactive data analysis. In addition, it provides visualization of intermediate query results, which greatly increases the interactivity of query processing of georeferenced data by allowing users to reach data sets of interest to them quickly and efficiently. The following query example illustrates online spatial query processing:

```
Query 1-1:
UNTIL ConfInt(0.95) <= 0.05
SELECT Estimate (Count(a.name)), Visualize(a.location)
FROM airport a
WHERE Intersect (a.geom, (SELECT r.geom ∩ w.geom
FROM rainfall r, wind w
WHERE Intersect(r.geom, w.geom) = 'TRUE' and r.rate > 1.0 and w.speed > 55.0 )) = 'TRUE'
GROUP BY a.state
```

Query 1-1 continuously displays incrementally refined running estimates of the final answer, i.e., the total number of airports in severe weather for each state. This query also obtains partial results such as airports found so far that are affected by severe weather and displays the partial results to the user (i.e., Visualize()). In this way, the user can get a “big picture” of how the query result will look without having to wait until the query is completed. The corresponding confidence intervals are calculated based on a confidence level. For example, if the approximate number of airports is 100 bounded by a 5% confidence interval with a 95% confidence level, then the exact answer is between 95 and 105 (100 ± 5) in 95% of the time. To make the system more interactive, users are given the ability to stop queries whenever the answer is “good enough” (i.e., within users’ accuracy
range) or to specify their desired accuracy when they issue queries. Query 1-1 continues processing the data until a 5% confidence interval is obtained. This approach allows users to quickly obtain an idea of whether or not the query result is useful and can therefore modify query processing or stop it accordingly. For instance, if users detect possible severe weather in only a subset of the regions (states) through the overall observation of approximate answers and partial results, they can speed up data analysis for these regions by stopping processing data in other regions.

Another issue for supporting online query processing is limited access to publicly available georeferenced data. Due to the lack of standard implementation of Web interfaces, many Web applications allow users to access the data through their restrictive interfaces. For example, some types of queries such as spatial range queries are not supported by many commercial Web sites, e.g., the McDonalds Web site. Query 1-1 may need to be changed if the data sets are stored in different Web applications and are only accessed through their Web interfaces. The information of U.S. major airports is available to limited users for security reasons. Many commercial Web sites for aviation provide information such as airport locations, weather conditions and flight paths, but only through their Web interfaces. In Query 1-1, a simple approach to retrieve the airports affected by severe weather is to conduct range queries using approximate minimum bounding boxes of each of the intersecting regions of two weather data sets. However, this can be a solution only if the Web applications support rectangular shaped spatial range queries. This assumption is not realistic since some Web applications may not support some query types based on their interfaces. For example, a Web application may only return the 10 nearest objects. This problem hinders online data retrieval, which necessitates for methods that utilize certain query types to provide solutions to non-supported queries.

This dissertation addresses the technical, practical and theoretical issues for supporting online query processing in GIS. Although there are many different issues such as data
1.1 Incremental Refining Spatial Joins for Vector Data

representation, data conversion, storage, data management (such as integrity and consistency) and security issues, this dissertation focuses on applications and techniques for online spatial queries and Web data retrieval. Since georeferenced data is usually stored based on one of either vector or raster models, this dissertation deals with online query processing techniques for both models. The characteristics of these models are taken into consideration for obtaining good approximations of their spatial joins. We explore spatial join techniques for each data model to enhance the efficiency and interactivity of online query processing. Furthermore, we conceptualize the Web data retrieval problem into a more general problem of supporting range queries through restrictive Web interfaces.

In this dissertation, we consider three such problems:

1. **Incremental Refining Spatial Joins for Vector Data**: This deals with the problem of interactive spatial joins for vector data

2. **An Interactive Framework for Raster Data Spatial Joins**: This problem focuses on developing an interactive query processing framework for raster data spatial joins

3. **Range Queries on Web Data using $k$-Nearest Neighbor Search**: This problem identifies the general range query problem for Web data retrieval and deals with deriving range query algorithms that utilize $k$-Nearest Neighbor queries.

In the rest of this chapter, we present the motivations of the problems, a brief preview of the results, and the main contributions of this dissertation.

1.1 Incremental Refining Spatial Joins for Vector Data

Many georeferenced data sets are represented by the vector model: each data entity (object) is represented by a combination of structured geometrical objects such as points, lines and polygons. Since the vector model provides a continuous and accurate representation
1.1 Incremental Refining Spatial Joins for Vector Data

of spatial objects’ boundaries in space, it is frequently used for high-quality cartography where accuracy and precision are important [53].

Data analysis of vector data is directly related to aggregate queries such as spatial joins. The spatial join is one of the most commonly used query types in GIS. A spatial join finds all object pairs satisfying some spatial predicates (e.g., intersection, containment and adjacency) from two or more relations (data sets). One example of such a query is to find all roads within 100 feet of rivers located at 1000 feet altitude or less in the U.S. In many GIS applications, spatial joins require accessing a large volume of georeferenced data sets in order to compute exact numerical query answers. However, computing an exact answer of a spatial join in large databases can be time-consuming due to the large size of data sets. While GIS and spatial databases offer many sophisticated join algorithms for computing exact query answers to spatial joins of georeferenced data, most of these techniques are time prohibitive for efficient data exploration of large data sets. As an example of vector data analysis in GIS, consider scientific data analysis where a great number of data sets are involved. Examples in Figure 1.2 show a portion of the real data sets used in the experiments in Chapter 3 in which minerals are joined with geochemical sediments in the U.S. The query result could be used to analyze how sediments like lakes and rivers affect mineral resources, such as salt rocks. Supposing that many such data sets are involved in join processing, obtaining the exact number of intersections between the pairs of the data sets could take several days. An approximate answer might be sufficient to see intuition of data sets or to find a scientific trend from their relationship, and can prevent wasted computation time, thus allowing a more efficient data exploration.

In Chapter 3, we propose a new approach to interactive online query processing for vector data spatial joins. This new approach allows users to obtain fast approximate answers to the query results, observe the progress of their queries and control the query processing. Our goal is to speed up the exploratory query processing while providing a
1.1 Incremental Refining Spatial Joins for Vector Data

Figure 1.2: Examples of vector data sets: natural resources in Colorado in the U.S.

statistical confidence of preliminary results through an incremental refining process. For example, instead of waiting a long time for an exact answer, we compute an approximate answer bounded by a 5% confidence interval with a 95% confidence level, in $\frac{1}{10}$ of time needed for an exact answer by using a subset of the data. Using this approach would not be appropriate if a query requires only 30 seconds to obtain an exact answer compared to say 3 seconds for an estimate using our approach. On the other hand, our approach can be attractive if we can compute a confidence interval bounded estimate in 15 minutes versus 3 hours for the exact answer. This scenario is likely since we are dealing with large georeferenced data sets that can be gigabytes or terabytes in size. Thus, many GIS applications can benefit from this approach that returns useful estimates quickly. The approach presented in this dissertation provides efficient and interactive data exploration and results in more effective decision support.
1.2 An Interactive Framework for Raster Data Spatial Joins

Raster data models, such as images and grids, represent geographic attributes by dividing the world (region) into discrete quadratic cells. They are suitable to represent continuous geographic phenomena such as elevation, slope and precipitation [23]. In raster models, each data set is usually called a layer. Example layers are vegetation, land use, temperature, etc. Layers are related if they have the same geographic coordinates.

With advances in the Internet, a great volume of raster data sets are available on the Web and can be used for decision support queries in many GIS applications. GIS users want to issue a query, see results immediately, and control the query processing as the query runs. Being able to do so in an interactive fashion would greatly increase the utility of the GIS. However, little research has been done on optimization or approximation techniques for spatial queries over raster data. Currently, performing spatial joins on raster data requires layers to be compared on a cell-by-cell basis. This spatial join process, referred to as map overlay, requires intensive computation time without any result until the join is completed.

Fast response times are especially important for user-driven data exploration used in GIS. For example, many different types of scientific data sets, e.g., satellite images of the earth, are available on the Web, and scientists need to analyze these data on a regular basis. Figure 1.3 shows examples of raster data sets used in the experiments in Chapter 4; minerals ($P$), chemical sediments ($Q$) and soil ($R$) of the state of Colorado. These data sets are represented by a $256 \times 256$ raster grid. Among more than one hundred such data sets, the scientists may only need the join results of the more “interesting” data sets. Thus, visualization of approximate answers in near instantaneous time provides them the chance to narrow down their “interesting” data sets or regions (subregions) quickly. We believe GIS users should be given the chance to see which are the “interesting” data set join pairs
1.2 An Interactive Framework for Raster Data Spatial Joins

without having to wait to compute the actual full joins. The duration of data validity is another issue for online query processing over raster data. As an example, consider raster data sets that are used for weather forecasts such as wind, rainfall and temperature. They are continuously changing in time and space, and hence the duration of data validity is relatively short. A blocking manner query processing may invalidate query results if long execution time is required. These issues motivate us to study techniques for fast approximation and interactive query processing of raster data.

In chapter 4, we propose a new interactive framework for raster data spatial joins, which enhances online query processing in GIS. The proposed framework utilizes two statistical techniques: Probabilistic Join (PJ) and Incremental Stratified Sampling Join (ISSJ). PJ is a novel spatial join technique for raster data to obtain approximate answers in near instantaneous time and ISSJ is a sampling technique that provides more accurate results than PJ but requires more query processing time (i.e., more I/Os). In the new framework, users specify queries and get near instantaneous visualizations of the answer using the PJ method. These result visualizations are approximations with reasonable errors. For queries that have interesting results, users can either use the ISSJ method to get a confidence bounded answer estimate or compute a full quad-tree join. By allowing
1.3 Range Queries on Web Data using $k$-Nearest Neighbor Search

users to get near instant approximate answers, they are able to explore far greater numbers and sizes of data sets than previously possible. This framework enables users to obtain a “big picture” of the answer in two orders of magnitude faster time than the time required for an exact answer.

1.3 Range Queries on Web Data using $k$-Nearest Neighbor Search

Recent advances in Internet technologies have provided a large amount of georeferenced data for many GIS applications. Unfortunately, access to these abundant and useful data sets is only possible through Web interfaces. These interfaces are usually designed for one specific query type and hence cannot support more general access to the data. For example, the McDonalds Web site provides a restaurant locator service through which one can ask for the five closest locations from a given zip code. This type of Nearest Neighbor query ($k$-Nearest Neighbor query) supported by Web interfaces is to search for a number of “nearest locations” from a given geographical point (e.g., a mailing address) or an area (e.g., a zip code). It is very popular for accessing georeferenced data on the Web and nicely serves the intended goal of quick and convenient dissemination of business location information to potential customers. However, if the consumer of the data is a computer program, as in the case of the Web data integration utilities (e.g., wrappers) and search programs (e.g., crawlers), such an interface may be a very inefficient way of accessing the data. For example, suppose a Web crawler wants to access the McDonalds Web site to retrieve all the restaurants in the state of California. Even though the site has the required information, the interface only allows the retrieval of five locations at a time. Even worse, the interface needs the center of the search as input. Hence, the crawler needs to identify a set of nearest location searches that both covers the entire state of California (for completeness) and has minimum overlap between the result sets (for efficiency).
1.3 Range Queries on Web Data using $k$-Nearest Neighbor Search

Figure 1.4 illustrates the challenges of this general problem using an example. Suppose that we want to find the locations of all the points in a given region (rectangle), and the only available interface is the $k$-Nearest Neighbor ($k$-NN) search with a fixed $k$ ($k = 3$ in the example of Figure 1.4). The only parameter that we can vary is the query point for the $k$-NN search. Given this query point $q$, the result of the $k$-NN search is a set of $k$ nearest points to $q$. It defines a circle centered at $q$ with radius equal to the distance from $q$ to its $k^{th}$ nearest neighbor (i.e., the 3$^{rd}$ closest point in this example). The area of this circle, covering part of the rectangle, determines the searched (i.e., covered) portion of the rectangular region. To complete the range query, it is required to identify a set of input locations of $q$ corresponding to a series of $k$-NN searches that would result in a set of circles covering the entire rectangle.

The optimization objective of this problem is to perform as few $k$-NN searches as possible. This is because each $k$-NN search results in communication overhead between the client and the server in addition to the actual execution cost of the $k$-NN search at the server. Thus, the circles should have as few overlaps as possible. In addition, the fact that we do not know the radius of each circle prior to the actual evaluation of its corresponding
1.4 Contributions

$k$-NN query makes the problem even harder. Hence, the sequence of our $k$-NN searches becomes important as well.

In Chapter 5, we conceptualize the aforementioned problem into a more general problem of supporting spatial range queries by using only $k$-Nearest Neighbor ($k$-NN) queries. We propose a set of range query algorithms that utilize different values of $k$ for the $k$-NN query. The proposed algorithms completely cover the rectangular shape of a given spatial range (completeness) while minimizing the number of $k$-NN queries as possible (efficiency). Statistical analysis of the proposed algorithms is conducted and the efficiencies of the algorithms are measured through empirical experiments.

1.4 Contributions

The main contributions of this dissertation can be summarized as follows:

- A new interactive spatial join technique for vector data in order to reduce query response time for obtaining query estimates with reasonably accurate confidence intervals while allowing users to have control over the query processing [5]

- A comprehensive study of three random sampling methods that can be used for interactive spatial joins of vector data [6]

- An interactive framework for raster data spatial joins that allows fast data exploration and more effective decision support in GIS. We present a novel spatial join technique, Probabilistic Join, for obtaining a “big picture” visualization of the query answer in near instantaneous time [7]

- Range query algorithms that utilize $k$-Nearest Neighbor search for Web data retrieval through restrictive Web interfaces [8]
1.5 Organization of Dissertation

The remainder of this dissertation is organized as follows:

- Chapter 2 provides an introduction of Geographic Information Systems (GIS) and spatial queries in GIS. It also provides a discussion of related work.

- Chapter 3 proposes a family of Incremental Refining Spatial Join algorithms, interactive spatial joins that can be used to report incrementally refined running estimates for aggregate queries while simultaneously displaying the actual query result of the sampled data.

- Chapter 4 presents An Interactive framework for Raster Data Spatial Joins, which allows users to get approximate answers in near instantaneous time thus allowing for truly interactive data exploration. This framework utilizes two proposed statistical approaches: probabilistic joins and quad-tree based sampling.

- Chapter 5 covers the Range Queries on Web Data using k Nearest Neighbor Search problem. Our proposed range query algorithms utilize k-Nearest Neighbor searches in order to support spatial range queries on Web data.

- Chapter 6 concludes with a summary of results and presents future research problems.
Chapter 2

Literature Survey

In this chapter we survey the literature on techniques for spatial databases and Geographic Information Systems (GIS) and approaches for online query processing. We first present an introduction to GIS and related work on spatial queries. Statistical techniques in databases are then explained with the details of random sampling techniques. Finally, we discuss approaches and techniques for online query processing. This literature survey summarizes past work on spatial queries and online query processing and shows how our work is related.

2.1 Geographic Information Systems (GIS)

Geographic Information Systems (GIS) are database management systems for the capture (collection), storage, retrieval, analysis and display of georeferenced data [45]. GIS are common ground between information processing and many fields that need spatial data analysis. They have been widely used in decision support systems involving the integration of georeferenced data in a problem solving environment.
2.1 Geographic Information Systems (GIS)

2.1.1 GIS Data and Applications

Georeferenced data is represented by spatial properties (i.e., location, shape) and non-spatial properties (i.e., name and type), and it describes the geometry and location of various types of geographic phenomena [45]. Georeferenced entities (objects) are characterized by their specific properties and their spatial relationships with other entities (e.g., distance). Georeferenced data is usually represented by one of two data models: vector model or raster model.

Vector data models represent geographic properties (attributes) with geometric units of spatial information such as points, lines, and polygons. Each object is composed of a series of one or more geometric units in the form of coordinates (locations). For example, a line is a collection of related points, and a polygon is a collection of related lines [45]. Vector models provide a continuous representation of spatial object boundaries in space, and therefore accurately represent the geometric shapes of objects [23]. In comparison to raster models, vector models are characterized by their comparatively low memory requirements and short computing times. However, the vector data format structure is much more complex than raster data format, e.g., topologies (spatial relationships) and geometric representations [66].

In raster data models, attributes are recorded by assigning each cell a single value describing the feature (attribute) being observed in the cell [23]. For instance, the cell values in an aerial photograph can represent the amount of light reflecting off the earth’s surface. Raster data is suitable to represent continuous geographic phenomena such as elevation, rainfall, and land use. Hence, they are heavily used for spatial modeling and analysis over data represented as continuous surfaces such as the dynamics of population change over time [66]. However, this model has a fixed resolution within a layer. To be able to represent localities of high spatial variation, the cell size has to be very small. This makes
2.1 Geographic Information Systems (GIS)

the size of raster data very large unless it is compressed [66]. The necessary memory requirements and the accompanying high demand on computing resources, which rise exponentially with the increase of resolution, are disadvantages of raster data. Additionally, raster data is not associated with spatial relationships (topologies) because each pixel is defined by its own situation in the coordinate system [23].

An increasing number of emerging applications have used GIS for exploring georeferenced data in order to support decision making process. Many different domains of knowledge are involved in these applications. Examples include urban planning, route optimization, demography, cartography, agriculture, natural resources and hazard administration, businesses, and epidemics control. Transportation management applications such as vehicles, ships and airplanes, constitute an interesting area of the use of GIS together with a positioning system such as Global Positioning System (GPS). As technology advances, more sophisticated location reporting devices are available. As a result, GIS are widely used in transport monitoring and routing applications such as traffic controls and airplane collision detection systems. GIS are also good tools for environmental management (pollution control), natural resources exploration/extraction (petroleum, mining, mineral resources), scientific research, and planning (roads and railways). Public services (crime analysis, weather forecasts, emergency management) have been identified as applications that can benefit from utilizing GIS. Public health planning is another good example of an application for the use of GIS. Public health organizations collect significant volumes of georeferenced data, monitor and analyze trends of environmental exposures. The analyzed results are used to identify problems to develop and evaluate their policies and services. Moreover, the use of GIS in marketing seems to have great potential. Numerous businesses release the locations of their branches and their services on the Web. GIS can be also deployed in economical trend analysis.
2.1 Geographic Information Systems (GIS)

2.1.2 Spatial Queries in GIS

Georeferenced entities are characterized by their specific properties and their spatial relationships with other entities (e.g., distance). Spatial relationships are defined as pairs of spatial entities, dependent on their relative locations in space. More definitions and classifications of spatial relationships are found in [19, 20]. Examples of spatial relationships include intersection, containment, and adjacency. For example, the intersection of two sets of areas (i.e., geometric objects) can be a set of areas, lines and points.

Spatial queries are queries that use locations and spatial relationships as a basis for retrieving information about spatial phenomena. Some commonly used spatial queries are as follows:

- Spatial join, e.g., how many rivers intersect highways in the U.S. and where?
- Range queries, e.g., find all gas stations that are located in a given rectangular shaped region defined by two points in a coordinate format (latitude, longitude), lower-left corner point (30.23, 40.24) and upper-right corner point (50.82, 68.20)
- Nearest Neighbor queries, e.g., find the closest bank to the University of Denver
- *k* Nearest Neighbor queries, e.g., find the 5 closest McDonalds to 6350 S. Havana St. in Englewood, Colorado

A spatial join is one of the most commonly used queries in spatial databases and GIS. A spatial join finds all object pairs satisfying some spatial predicates (e.g., intersect) from two or more relations (data sets). A spatial join of raster data sets is often called an overlay. In traditional geographic analysis, the overlay operations have been performed by putting different map layers of the same region on top of each other for visual inspection or for printing the result to produce a new map [66].

18
2.1 Geographic Information Systems (GIS)

Many spatial join algorithms [12, 38, 43, 52] based on R-trees have been proposed to optimize query performance. Our research work in this dissertation builds upon the substantial previous work in spatial databases and spatial joins. In [52] the authors proposed a multiway spatial join algorithm, a window reduction algorithm that was used to join two or more R-trees. A data item from one data set is joined with the other data set by executing a window query. For example, if we join “cities” and “rivers”, then the MBR of a city provides the query minimum bounding rectangle (MBR) to be executed against the river data set.

Our proposed algorithms, Incremental Refining Spatial Joins (IRSJ) in Chapter 3, follow a similar approach to that found in [52]. We use the R-tree and its variants as index structures for IRSJ, and the window query in [52] is used as part of our algorithms. Our work differs from the past work in that we only execute intersection queries for a subset of the outer data set through sampling. The previous spatial join algorithms are designed to minimize the time to complete the query. However, our algorithms are designed to minimize the time until an acceptably accurate estimate of the query result is available measured by a confidence interval. Furthermore, our algorithms not only return an estimate of aggregate query values but also a partial result from actual joins.

A multiway join combines data originated from more than two relations. Although many techniques have been proposed for pairwise spatial joins, only limited work has focused on multiway spatial join processing [44, 51, 52]. Analysis of previous work on spatial joins were conducted in [44]. Various techniques that combine local and evolutionary searches with underlying indexes to prune the search space were proposed in [51]. The authors in [52] presented a solution for multiway spatial joins by applying systematic search algorithms that exploit R-trees without building temporary indexes or materializing intermediate results. They provided cost models and proposed an algorithm that efficiently computed the spatial join between two inputs, only one of which is indexed by
2.1 Geographic Information Systems (GIS)

an R-tree, and showed how it can be implemented to join more than two data sets. Also
the authors proposed a dynamic programming algorithm that optimizes the execution of
complex spatial queries.

One common raster data spatial join technique is map overlay [66]. It is used to com-
bine different types of geographical information for a region. An example of an overlay is
combining a rainfall information layer with a landscape layer to produce a new informa-
tion layer that shows certain areas that receive little rainfall and thus are likely to dry up.
Raster overlay is straightforward when the input rasters have the same cell boundaries.
The resulting raster can be obtained cell by cell from the originals using the relevant op-
erations on the cell values. However, little research work has been done on map overlay
optimization techniques. Since GIS data can reach gigabytes and possibly terabytes in
size, full layer overlays could take hours and even days to complete. This necessitates the
need for approximation techniques.

A range query in a spatial database can be formally defined as follows: Let $R$ be a
given rectangular query region represented by two points, $P_{ll}$ and $P_{ur}$, where $P_{ll} = (x_1, y_1)$
and $P_{ur} = (x_2, y_2)$. $P_{ll}$ and $P_{ur}$ refer to the lower-left corner and the upper-right corner of
$R$, respectively. Given a set of objects $S$ and a region $R$, spatial range query searches for
all objects of $S$ in $R$ [25].

Given a set of objects $S$ and a query point $q$, a Nearest Neighbor query for a point
$p \in S$ such that $\text{dist}(q, p) \leq \text{dist}(q, r)$ for any $r \in S$. Point $p$ is the nearest neighbor
of $q$. Users can be interested in finding more than one point, say $k$ points, which are
the nearest neighbors of the query point $q$. This is called the $k$-Nearest Neighbor ($k$-
NN) problem also known as the Top-$k$ selection query. Given a set of objects $S$ (with
$\text{cardinality}(S) = N$), a number $k \leq N$ and a query point $q$, a $k$-NN query searches for a
subset $S' \subseteq S$ with $\text{cardinality}(S') = k$ such that for any $p \in S'$ and $r \in S - S'$, then
$\text{distance}(q, p) \leq \text{distance}(q, r)$ [62].
2.1 Geographic Information Systems (GIS)

Many $k$-NN query algorithms have been proposed [22, 54, 64]. In [22], given a set of points, the authors proposed two algorithms to solve two $k$ nearest neighbor related problems. The first enumerates the $k$ smallest distances between pairs of points in non-decreasing order, and the second finds the $k$ nearest neighbors of each point. Both are based on a Delaunay triangulation [22]. The branch-and-bound R-tree traversal algorithm that finds the $k$ nearest neighbors of a given query point was proposed in [54]. It also introduced two metrics for both searching and pruning, which are MINDIST that produces the most optimistic ordering and MINMAXDIST that produces the most pessimistic ordering possible. A new cost model for optimizing nearest neighbor searches in low and medium dimensional spaces was proposed in [64]. The authors proposed a new method that captures the performance of nearest neighbor queries based on approximation.

2.1.3 Spatial Access Methods and Data Structures

Geographical or spatial queries are mostly performed on a spatially indexed database in order to obtain answers that depend on spatial relationships between data entities [40]. The R-tree [29] and its variants are the most common index structures in spatial databases. R*-trees and packed R-trees using different packing algorithms are examples of R-tree variants. The R-tree is a height balanced tree structure adapted from the B-tree to support spatial data. An R-tree stores the minimum bounding rectangles (MBRs) of objects. There are three main splitting algorithms that were proposed by Guttman in [29] to deal with nodes overflow during insertion, namely, linear split, quadratic split and exponential split. R*-tree is the first variance of the R-tree. The main difference between the R*-tree and the R-tree is the concept of forced re-insertion in the R*-tree [10], which results in a better tree structure. The STR-tree uses the Sort-Tile-Recursive (STR) algorithm [41]. The main idea of the STR-tree is to fully pack leaf level nodes by tiling data. The STR algorithm sorts MBRs by $x$-coordinate that are then partitioned into vertical slices, then sorts the
2.1 Geographic Information Systems (GIS)

MBRs of each slice by y-coordinate. The sorted MBRs are divided into sorted runs and each run is packed into a node. It is clear that the data structure is more balanced and better utilized in the STR-tree than in the R-tree or any other R-tree variant. When performing a region query on an R-tree indexed data set, all rectangles (MBRs) in the R-tree which intersect the query region are retrieved. This is done in a recursive way, starting from the root and following the paths down to the leaf level.

The quad-tree is a very popular hierarchical data structure for the representation of binary images and maps, and it is commonly used in spatial databases [56, 57, 68], i.e., indexing for query processing and optimizing decomposition. A quad-tree is a tree whose nodes either are leaves or have four children [57]. To represent spatial data by a quad-tree, a given region is first divided into four quadrants of equal sizes. Then, each quadrant will be further subdivided if it has more objects than its capacity. The process continues until each quadrant contains a certain number of objects. In terms of the tree representation, the root node corresponds to the entire region, and each child node represents a quadrant. The quad-tree represents a recursive quaternary decomposition of space wherein at each level a subregion is divided into four equal sized subregions (quadrants). The properties of the quad-tree provide a natural framework for optimizing decomposition [70].

The use of Voronoi diagrams [3] or Delaunay triangulations for answering spatial queries in GIS has been discussed in [27]. The authors in [27] showed that the Voronoi diagram is a data structure suitable for spatial queries. The Delaunay triangulation and Voronoi diagram based approaches have been studied for various purposes in wireless sensor networks [46, 61, 69, 71]. The Delaunay triangulation and Voronoi diagram were used to determine the maximal breach path (MBP) and the maximal support path (MSP) for a given sensor network in [46]. In [69], the authors used the Voronoi diagram to discover the existence of coverage holes, assuming that each sensor knows the location of its neighbors. The region is partitioned into Voronoi cells, and each cell contains one
sensor. If a cell is not covered by a sensor, the coverage holes are found. A wireless sensor network deployment method was proposed based on the Delaunay triangulation [71]. This method was applied for planning the positions of sensors in an environment with obstacles. This method retrieved the location information of obstacles and pre-deployed sensors, and then constructed a Delaunay triangulation to find candidate positions of new sensors.

2.2 Statistical Techniques in Databases

Statistics is used as the most common approach for analyzing spatial data. It handles numerical data well and usually comes up with realistic models of spatial phenomena. However, this approach is usually based on the assumption of statistical independence among the spatially distributed data which may cause problems as most spatial data is in fact correlated. We present related work on statistical techniques for spatial join selectivity, with an emphasis on random sampling techniques of spatial joins.

2.2.1 Statistical Techniques for Spatial Joins

Despite the existence of a lot of work on the approximation of joins, most are focused on relational databases and can not be applied directly to GIS and spatial databases. Selectivity estimation for spatial joins has been studied in [1, 11, 21, 24, 39, 63, 65]. Accurate selectivity estimation is crucially important during query optimization for finding good query plans. The selectivity estimation can also be used for fast estimation of an aggregation query, for example, a query to find correlation between two data sets using join selectivity estimation. Statistical techniques are used for spatial join selectivity estimates, of which the three most common techniques are parametric methods, histograms and sampling.
2.2 Statistical Techniques in Databases

Parametric methods use some properties of data distribution to present a formula for the estimation [11, 15, 24]. However, parametric methods make some assumptions about the data sets: uniformly distributed data sets [15], fractal behavior [11], or obedience power laws [24]. The methods in [11, 24] were proven to work with only point data sets. Due to these restrictive assumptions (data distribution dependent), parametric methods are not yet applicable to real data sets in GIS since real data may not necessarily have such properties. The major difference between our incremental sampling approach (Chapter 3 and 4) and parametric methods is that our method returns both an estimated join result and an incrementally accumulating set of actual result tuples whereas the parametric methods only return an aggregate value.

Since histograms and sampling are less restrictive than parametric methods, they are applicable to a wide range of data sets in GIS. Histograms keep certain information for different regions of the spatial extent in an auxiliary data structure (histograms), and this structure is used to estimate the join selectivity when a query is given. Current work has shown that histograms are superior to sampling for a wide range of queries [1, 63]. However, histograms require extensive experimental evaluation with a known data distribution to obtain an error bound. Although the experimental results in [1, 63] show low error, it is hard to quantify the tradeoff between storage and accuracy due to the lack of error guarantees. A method that permits high-quality selectivity estimations for spatial joins was introduced in [21]. This approach returned approximate results that come with provable probabilistic quality guarantees. As for parametric methods, histograms also return only aggregate values and not actual result tuples. Thus, the histogram approach does not solve the same problem discussed in this dissertation.

Sampling techniques are used to calculate an estimate of the final result from a subset (samples) of a data set (the population) in GIS and to provide a confidence interval for the accuracy of the result. These approaches use information associated with the popula-
2.2 Statistical Techniques in Databases

tion, samples drawn from the population and distribution of the samples. While sampling techniques have been used in estimation in relational databases [30–32], there have been few prior attempts at investigating the usability and accuracy of sampling to spatial data sets. Different methods for selecting random samples of the query results from R-trees and quad-trees were proposed in [49]. The authors in [68] discussed four different random sampling methods applied to quad-trees and presented analytical cost results of these methods. Three sampling methods to estimate the selectivity of spatial joins were evaluated in [1]. The difficulty in picking a representative sample with low overhead makes sampling undesirable. However, incremental random sampling overcomes that problem in addition to providing a non-blocking and interactive query processing.

Not only is fast and accurate join selectivity important, but also an intermediate result (the subset of actual join result) is important for exploratory decision support in GIS. We investigate sampling techniques to obtain spatial join selectivity estimation in the environment of online query processing in GIS. Our approach provides error confidence intervals for joins and the ability to incrementally tighten the intervals. Each sampling is used to produce a subset of the actual join results (actual join location) rather than just an estimate of the number of joins (intersections).

Quad-tree based sampling has been proposed in [49, 68]. In [68], the authors presented the analysis of four different sampling methods proposed by [49]. They applied sampling algorithms to specific quad-tree implementations to obtain approximate aggregate query results. They proposed two models to analyze sampling costs.

The idea of incremental sampling technique using R-trees to provide interactive spatial join processing was proposed in [5]. The authors proposed two R-tree based sampling methods that were used to incrementally refine the estimated join result while providing a bounded confidence interval. Their approach was applied for vector-based data rather than raster data.
2.2 Statistical Techniques in Databases

In [4, 73] the authors presented an approximation technique of vector-based spatial joins. First, they converted vector data to raster format and filtered the possible joined pairs using the *Four Color Raster Signature* in [73] and the *Three Color Raster Signature* in [4]. They combined progressive and conservative approximations [13] in a single approximation to speed up the filtering step in identifying intersecting polygons. Their proposed techniques motivated us to obtain the join probability of two raster data sets.

2.2.2 Random Sampling and Probabilistic Query Techniques

In data sampling, a subset of the data is chosen as samples to obtain query estimates. The most common application of sampling is for aggregate queries such as AVG, SUM, and COUNT, where reasonably accurate answers of the aggregates can be obtained from a sample of the data. Sampling can also be used to obtain a random subset of the actual tuples in relations to speed up data mining and analysis. Chosen samples should accurately represent the entire data set; a confidence interval is used to reflect the accuracy of the estimates [58, 59]. In this dissertation, sampling techniques are used as part of the overall approach to interactive spatial joins in GIS. We studied three common sampling methods [58] for the incrementally refining running estimates of the final result to spatial joins: simple random sampling (tuple-level sampling), stratified random sampling, and cluster sampling (page-level sampling). These three sampling methods are used in interactive spatial joins for vector data (Chapter 3), and stratified random sampling is used in an interactive framework for raster spatial joins (Chapter 4).

In simple random sampling, a number of tuples are chosen as samples, each with the same probability. Simple random sampling obtains random samples regardless of data clustering. However, its performance is poor if tuples are not indexed. If an index structure is available, the performance of simple random sampling can be improved since the sampling predicate is applied to the index of pages [50].
2.2 Statistical Techniques in Databases

In stratified random sampling, the population is divided into non-overlapping groups called strata. Samples are then chosen from each stratum using simple random sampling. Some groups of the population may be of interest and thus should be defined as strata. If strata are well-defined and the population is carefully divided into strata, this sampling method may provide several advantages [59]. Stratified random sampling may result in smaller error bounds on the estimation. It may also reduce the sampling cost.

In cluster sampling, each sample is a collection of elements (tuples) called a cluster. Cluster sampling provides an inexpensive way for obtaining information when a good clustering technique is available, and the cost of simple random sampling is usually more expensive [58]. In this paper, clusters are the leaf level pages of R-trees of the outer relation. In cluster sampling, leaf level pages are randomly chosen as samples. If a page is chosen, then all tuples in that page are processed.

Cluster sampling has better performance of a sampling process than simple random sampling or stratified random sampling in terms of I/Os [60]. However, the aggregate estimate accuracy tends to be worse when using cluster sampling compared to the other two sampling methods. Tuples within a cluster are physically (geographically) close together. As a consequence, they tend to have similar characteristics (correlation). In our experiments in Chapter 3, we compare the accuracy and the time (I/Os) needed for cluster sampling to those of simple random sampling and stratified random sampling for different desired confidence intervals. The main difference between stratified and cluster sampling is the construction of strata and clusters in order to get optimal results. In stratified random sampling, strata should be as homogeneous as possible from other strata with respect to the measured criteria. On the other hand, in cluster sampling, clusters should be as heterogeneous as possible within each cluster, and clusters should look alike with respect to the measured criteria [59].

The query estimations and confidence intervals are statistically meaningful only if
2.3 Online Query Processing

samples are retrieved at random. [49] presented techniques for random sampling from various indexes to produce meaningful confidence intervals. We assume that any one of these random access techniques is available. The weighted random sampling methods used in our experiments is the **Acceptance/Rejection** method [49, 55], in which the inclusion probability is proportional to some parameters of the item sampled.

Probabilistic queries in relational databases have been studied in [16, 17]. Probabilistic query evaluation was studied for uncertain continuously changing data in relational databases [16]. In [17], the authors proposed probabilistic join over uncertain data. They provide techniques to answer queries that return results with probabilities exceeding a given threshold.

2.3 Online Query Processing

Online query processing work was proposed by the University of California at Berkeley [34–36]. The authors proposed an interface for online aggregation. The goal was to perform online aggregation that allows users both to observe the progress of their queries (e.g., the average of GPA with 14% data processed) and to control execution on the fly (e.g., speed up the retrieval of specific relation over the other). They provided methods for computing running confidence intervals and the estimated proximity of each running aggregate to the corresponding final result. In [35], the authors provided initial motivation, statistics and algorithms for supporting online aggregation. The proposed join algorithms in [36] were designed to meet the performance needs of an online query processing system. However, their work was geared towards online aggregation in relational databases. Due to spatial properties of georeferenced data, spatial queries require complex and sophisticated algorithms. As a result, the work for relational databases is not directly applicable for GIS and spatial databases. In this dissertation, we follow the general ap-
2.3 Online Query Processing

An approach for estimating query answer using sampling techniques in [35] but focus on query processing techniques for spatial queries of large georeferenced data.

Some studies have discussed the problems of data retrieval and integration on the Web [37, 47, 72]. A survey of the general problem of the integration of Web data sources can be found in [37]. The paper discussed challenges of the heterogeneous nature of the Web and different ways of describing the same data, and proposed general ideas of how to tackle those problems. A query processing framework for Web-based data integration was presented in [47] and implemented the evaluation of query planning and statistics gathering modules. The problem of querying Web sites through limited query interfaces has been studied in the context of information mediation systems [72]. Based on the capabilities of the mediator, the authors proposed an idea to compute the set of supported queries.

Several studies have focused on performing spatial queries on the Web [9, 14, 42]. In [9], the authors demonstrated an information integration application that allows users to retrieve information about theaters and restaurants from various U.S. cities, including an interactive map. Their system showed how to build applications rapidly from existing Web data sources and integration tools. The problem of spatial coverage using only the k-NN interface was introduced in [14]. The authors used the quad-tree data structure for checking covered regions. A quad-tree data structure was used to check the complete coverage. The problem of supporting k-NN queries using range queries was studied in [42]. The idea of successively increasing query ranges to find k points was described assuming statistical knowledge of the data set. This is the reverse of the problem we study in Chapter 5, Range Queries on Web Data using k-Nearest Neighbor Search.
Chapter 3

Incremental Refining Spatial Joins for Vector Data

3.1 Introduction

In this chapter, we present a family of Incremental Refining Spatial Join (IRS J) algorithms, which return incrementally refined running estimates of the final answer of spatial joins with bounded confidence intervals using three random sampling techniques: simple random sampling (tuple-level), stratified random sampling and cluster sampling (page-level). In addition, the IRS J algorithms display the partial results of the data sets joined so far allowing users to gain more insight into the query results.

Our objective is to minimize the time until an acceptably accurate estimate of a spatial join is available (to users) measured by a confidence interval. While similar work has been done in relational databases, to the best of our knowledge, this is the first work using this approach for spatial joins in GIS. We investigate and evaluate different sampling methodologies through extensive experimental performance comparisons. Finally, we demonstrate the efficiency of the IRS J algorithms with empirical evaluations. Experiments on both real and synthetic data show an order of magnitude response time improvement rela-
3.2 Overview of Incremental Refining Spatial Joins

The Incremental Refining Spatial Join (IRS J) algorithms consist of three steps: sampling, spatial joining, and refining the estimation of the query result using statistics:

1) Sampling step: the samples (tuples) are randomly chosen from the outer relation $R$

The notation used in this chapter is summarized in Table 3.1

<table>
<thead>
<tr>
<th>Notation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$R$</td>
<td>the outer relation</td>
</tr>
<tr>
<td>$S$</td>
<td>the inner relation</td>
</tr>
<tr>
<td>$N$</td>
<td>the size of population (the total number of tuples in $R$) for IRS $J_t$ and IRS $J_s$</td>
</tr>
<tr>
<td>$N_p$</td>
<td>the size of population (the total number of pages in $R$) for IRS $J_p$</td>
</tr>
<tr>
<td>$n_t$</td>
<td>the sample size for a sampling step for IRS $J_t$</td>
</tr>
<tr>
<td>$n_s$</td>
<td>the sample size for a sampling step for IRS $J_s$</td>
</tr>
<tr>
<td>$n_p$</td>
<td>the sample size for a sampling step for IRS $J_p$</td>
</tr>
<tr>
<td>$I$</td>
<td>the total number of tuples that intersect tuples of $S$ in a sampling step</td>
</tr>
<tr>
<td>$I_{i}$</td>
<td>the total number of tuples that intersect tuples of $S$ in stratum $i$, where $i = 1, ..., k$</td>
</tr>
<tr>
<td>$I_{ij}$</td>
<td>the number of tuples within the $j$th page intersecting tuples of $S$</td>
</tr>
<tr>
<td>$k$</td>
<td>the total number of strata</td>
</tr>
<tr>
<td>$N_i$</td>
<td>the number of tuples in stratum $i$, where $i = 1, ..., k$</td>
</tr>
<tr>
<td>$n_i$</td>
<td>the sample size of stratum $i$ for a sampling step, where $i = 1, ..., k$</td>
</tr>
<tr>
<td>$n_{ij}$</td>
<td>the sample size for the $j$th sampling step</td>
</tr>
<tr>
<td>$S_j$</td>
<td>the incremental sample size for the $j$th sampling step</td>
</tr>
<tr>
<td>$s_{ij}$</td>
<td>the incremental sample size of stratum $i$ for the $j$th sampling step, where $i = 1, ..., k$</td>
</tr>
<tr>
<td>$T_j$</td>
<td>the total number of tuples within the $j$th page for a sampling step</td>
</tr>
<tr>
<td>$A_j$</td>
<td>the average number of tuples within the $j$th page intersecting tuples in $S$</td>
</tr>
<tr>
<td>$C$</td>
<td>the number of intersections in a sampling step</td>
</tr>
<tr>
<td>$C_i$</td>
<td>the number of intersections for stratum $i$ in a sampling step</td>
</tr>
<tr>
<td>$C_I$</td>
<td>confidence interval</td>
</tr>
<tr>
<td>$EV$</td>
<td>an estimate of the number of intersections</td>
</tr>
</tbody>
</table>

Table 3.1: Notations used in IRS $J$
3.2 Overview of Incremental Refining Spatial Joins

![Block diagram of the Incremental Refining Spatial Join (IRS J) process]

Figure 3.1: Block diagram of the Incremental Refining Spatial Join (IRS J) process

Based on the sampling method used.

1) Sampling step: samples tuples from outer dataset $R$ based on sampling method.

2) Spatial joining step: the MBR of each tuple is used as the MBR of a window query to the inner relation $S$. Each window query returns the number of intersections (and the number of joins) it finds. It also returns the actual join locations of $R$ and $S$. The number of intersections as well as the actual join data are denoted as the intermediate result.

3) Refining step: the number of intersections (joins) found in each step is used to calculate a running estimate and a confidence interval for the final aggregates. Finally, the calculated running estimate and the confidence interval are combined with the intermediate result into a query result through a visualization process.
3.3 Incremental Refining Spatial Join (IRS J) Algorithms

The query result is then reported to the user. The user can stop the query process if the given confidence interval is sufficient or the user sees trends from the visualized actual join locations (intermediate result), otherwise the process continues. Each step of the process is repeated in an incremental manner to calculate a new statistic until a desired confidence interval is achieved. Figure 3.1 shows the overview of the IRS J process, where the two relations $R$ and $S$ are joined.

In the IRS J algorithms, the index of the inner relation is required, but the index of the outer relation is not necessary. However, we assume that the outer and the inner relations are both indexed by R-trees or its variants since our comparison metric is the time to get join estimates against the full R-tree join.

3.3 Incremental Refining Spatial Join (IRS J) Algorithms

The IRS J algorithms are a family of algorithms based on the general framework described above. The three algorithms, $IRS_J_t$, $IRS_J_p$, and $IRS_J_s$ differ only in how the outer relation samples are obtained as described in the following subsections.

![Diagram of IRS J algorithms](image)

**Figure 3.2:** An example of the Window Query
3.3 Incremental Refining Spatial Join (IRS J) Algorithms

Algorithm 3.1 IRS J t(R, S)

1: \( C \leftarrow 0 \) // the number of current intersections
2: \( n_t \leftarrow 0 \) // the sample size
3: \( n \leftarrow 30 \) // the incremental sample size of a sampling step
4: repeat
5: \( n_t \leftarrow n_t + n \)
6: for \( i = 1 \) to \( n \) do
7: \( L \leftarrow \) Choose a leaf from \( R \) at random
8: \( t \leftarrow \) choose a tuple \( t \) from \( L \) at random
9: \( M \leftarrow \) MBR of a chosen tuple \( t \)
10: \( I \leftarrow \) number of intersections of a Window Query(\( M, S \))
11: \( C \leftarrow C + I \)
12: end for
13: \( C_I \leftarrow \) Compute a confidence interval using \( C \) and \( n_t \)
14: \( EV \leftarrow \) Compute an estimate using \( C \) and \( n_t \)
15: report \( EV, C_I, \) and \( C \)
16: until \( C_I \) is sufficient to the user

3.3.1 IRS J t ("t" for "tuple")

This IRS J variant corresponds to the “simple random sampling” method as described in Section 2.2.2. In IRS J t, we select a tuple from the outer relation \( R \) at random by selecting a page at random and then randomly selecting a tuple within that page. We use the MBR of this tuple as the MBR of a window query to query data set \( S \) using its R-tree to find intersections. Figure 3.2 shows an example of a window query, where the two relations \( R \) and \( S \) are rivers and cities, respectively. For example, if we choose a leaf page, \( L_1 \), at random from \( R \) and randomly choose a tuple, \( r_3 \), from \( L_1 \), the MBR of \( r_3 \) is then used to query data set \( S \). The query returns the number of intersections. In this example,
3.3 Incremental Refining Spatial Join (IRS J) Algorithms

the query result is one intersection with tuple \( s_2 \) in \( S \). In general, each window query is used to obtain the number of intersections for an outer relation tuple. The number of intersections found by the sampling step is used in calculating a running estimate and a confidence interval.

Algorithm 3.1 describes the IRS \( J \) algorithm. A tradeoff exists between the rate at which the confidence intervals are updated and the time to which the interval length decreases at each update. In our experiments, \( n = 30 \) is used as the number of tuples in each sampling step for IRS \( J \). Since pages may contain a different number of tuples, it is necessary to choose pages with probabilities relative to the number of tuples within each page.

3.3.2 IRS \( J_s \) ("s" for "stratified")

![Figure 3.3: An example of strata and clusters in the R-tree](image)

This IRS \( J \) variant corresponds to the "stratified random sampling" method as described in Section 2.2.2. In general, the strata are pre-defined in stratified random sam-
Algorithm 3.2 createStrata($R, k$)

1: $I_{\text{node}} \leftarrow$ the number of children of the root of $R$
2: while $I_{\text{node}} < k$ do
3: \hspace{1em} go to the next level of the tree
4: \hspace{1em} $I_{\text{node}} \leftarrow$ the number of children at the current level
5: \hspace{1em} end while
6: $P \leftarrow \{P_i | P_i$ is a child node at the current level, where $i = 1$ to $I_{\text{node}}\}$
7: $index \leftarrow 0$
8: for $i = 1$ to $k$ do
9: \hspace{1em} if $i \leq (k - I_{\text{node}} \% k)$ then
10: \hspace{2em} $|ST_i| \leftarrow \lfloor \frac{I_{\text{node}}}{k} \rfloor$
11: \hspace{1em} else
12: \hspace{2em} $|ST_i| \leftarrow \lfloor \frac{I_{\text{node}}}{k} \rfloor + 1$
13: \hspace{1em} end if
14: \hspace{1em} $N_i \leftarrow |ST_i|$ 
15: \hspace{1em} for $j = index + 1$ to $index + N_i$ do
16: \hspace{2em} $ST_j \leftarrow$ all leaf level pages in the subtree of $P_j$
17: \hspace{1em} end for
18: \hspace{1em} index $\leftarrow N_i$
19: end for
20: $ST \leftarrow \{ST_1, ..., ST_k\}$
21: return $ST$

The IRS $J_x$ algorithm defines the strata based on the structure of the R-tree. We first compare the number of children of the root of $R$ to $k$, where $k$ = the number of strata desired for the given data set $R$. If this number is less than $k$, then we proceed down the tree until the current level nodes of the tree have at least $k$ number of children in total. The children nodes at that level are then grouped and assigned to one of the strata so that all
3.3 Incremental Refining Spatial Join (IRS J) Algorithms

Algorithm 3.3 IRS J(R, S, k)

1: \( ST = \{ST_1, \ldots, ST_k\} \) // \( ST \) is a set of strata
2: \( C_1, \ldots, C_k \leftarrow 0 \) // the current intersections for stratum \( i \)
3: \( n_s \leftarrow 0 \) // the sample size for a sampling step
4: \( n_{init} \leftarrow 30 \) // the initial incremental sample size for a sampling step
5: \( n_1, \ldots, n_k \leftarrow 0 \) // the sample size for stratum \( i \)
6: \( s_1, \ldots, s_k \leftarrow 0 \) // the incremental sample size for stratum \( i \)
7: repeat
8: \( \) compute \( s_1, s_2, \ldots, s_k \) for \( ST_1, ST_2, \ldots, ST_k \) using \( n_{init} \)
9: \( S \leftarrow \sum_{i=0}^k s_i; n_s \leftarrow n_s + S \)
10: for \( i = 1 \) to \( k \) do
11: \( n_i \leftarrow n_i + s_i \)
12: for \( j = 1 \) to \( n_i \) do
13: \( L \leftarrow \) Choose a leaf from stratum \( i \) at random
14: \( t \leftarrow \) choose a tuple \( t \) from \( L \) at random
15: \( M \leftarrow \) MBR of a chosen tuple \( t \)
16: \( I_i \leftarrow \) number of intersections of a Window Query(\( M, S \))
17: \( C_i \leftarrow C_i + I_i \)
18: end for
19: end for
20: \( C \leftarrow \sum_{i=0}^k C_i \)
21: \( C_I \leftarrow \) Compute a confidence interval using \( \{C_1, \ldots, C_k\} \) and \( \{n_1, \ldots, n_k\} \)
22: \( EV \leftarrow \) Compute an estimate using \( \{C_1, \ldots, C_k\} \) and \( \{n_1, \ldots, n_k\} \)
23: \( \) report \( EV, C_I, \) and \( C \)
24: until \( C_I \) is sufficient to the user
3.3 Incremental Refining Spatial Join (IRS J) Algorithms

Pages at the leaf level are allocated to the strata. In the example of Figure 3.3, there are 4 children nodes of the root, $R_1$, $R_2$, $R_3$, and $R_4$. Assuming that the number of strata is two, $k = 2$, and $N_1$ and $N_2$ are the total numbers of tuples in the first strata $ST_1$ and the second strata $ST_2$, respectively, we then group these four second-level nodes into the two strata, $ST_1$ and $ST_2$. By grouping the second-level nodes, all leaf nodes are partitioned into the two strata as shown in Figure 3.3. $ST_1$ includes four leaf pages, $L_1$, $L_2$, $L_3$, and $L_4$, and $ST_2$ has four leaf pages, $L_5$, $L_6$, $L_7$, and $L_8$. Therefore, $N_1 = 8$ and $N_2 = 10$. Algorithm 3.2 describes the process of defining the strata.

In IRS $J_s$, the sample size (the number of tuples) of each stratum is calculated for every sampling step, and the sample size of a stratum is proportional to the total number of tuples within that stratum. In the example of Figure 3.3, we suppose that the initial sample size is $n_{init} = 3$, and $n_1$ and $n_2$ are the sample sizes of $ST_1$ and $ST_2$, respectively. For the first sampling step, we calculate $n_1 = 1$ and $n_2 = 2$, thus $n_s = 3$. In the second sampling step, $N_1 = 7$ and $N_2 = 8$, and $n_s = 3$ since again, $n_1 = 1$ and $n_2 = 2$. However, $n_1 = n_2 = 2$, thus $n_s = 4$ since we have $N_1 = 6$ and $N_2 = 6$ in the third sampling step.

Samples are then chosen from each stratum using simple random sampling. The MBRs of the sampled tuples are used as the MBRs of window queries to query data set $S$ to obtain intersections. For each stratum, we obtain the number of intersections, and this number is used to calculate the estimate and confidence interval for the corresponding stratum. The sum of the intersections of each stratum is the current intermediate result, and the estimates and confidence intervals of all strata are combined for the statistics (an estimate and a confidence interval) of the current sampling step. This incremental process continues until the returned confidence interval is sufficient to the user. In the example of Figure 3.3, assuming that $n_1 = 1$ and $n_2 = 2$ in a sampling step, the IRS $J_s$ algorithm then randomly chooses one tuple from $ST_1$ and two tuples from $ST_2$ using simple random sampling. The MBRs of these sampled tuples are used as the MBRs of window queries.
3.3 Incremental Refining Spatial Join (IRS J) Algorithms

Algorithm 3.4 IRS Jₚ(ₚ, S)

1: \( C \leftarrow 0 \) // the number of current intersections
2: \( n_p \leftarrow 0 \) // the sample size for a sampling step
3: repeat
4: \( L \leftarrow \) Choose a leaf from \( R \) at random; \( n_p \leftarrow n_p + 1 \)
5: while a non visited tuple \( t \) exists in \( L \) do
6: \( t \leftarrow \) choose a tuple \( t \) from \( L \)
7: \( M \leftarrow \) MBR of a chosen tuple \( t \)
8: \( I \leftarrow \) number of intersections of a Window Query(\( M, S \))
9: \( C \leftarrow C + I \)
10: end while
11: \( C_I \leftarrow \) Compute a confidence interval using \( C \) and \( n_p \)
12: \( EV \leftarrow \) Compute an estimate using \( C \) and \( n_p \)
13: report \( EV, C_I \), and \( C \)
14: until \( C_I \) is sufficient to the user

to query \( S \) to obtain intersections for this step. Algorithm 3.3 describes the IRS Jₙ algorithm, where \( k \) is the number of strata desired. The formulas of statistics and the proof of IRS Jₙ are provided in Section 3.4.

3.3.3 IRS Jₚ (“p” for “page”)

This IRS J variant corresponds to the “cluster sampling” method as described in Section 2.2.2. In IRS Jₚ, a page at the leaf level of the R-tree of \( R \) is randomly picked. The difference between IRS Jₙ and IRS Jₚ is that in IRS Jₚ, we compute a window query for each tuple within a sampled page to obtain the number of intersections, and we then calculate the average of intersections for that page and use it as a single sample. Due to the likely correlation of tuples within a page, we can not use the result from each outer tuple
3.4 Statistics for the IRS J Algorithms

As a valid sample when calculating a confidence interval. Thus, it is necessary to treat
the average over all tuples within a page as a single value when calculating a confidence
interval.

In the example in Figure 3.3, the leaf level pages, $L_1$ through $L_8$ are defined as the
clusters. Supposing that $L_3$ is randomly chosen as a sample, then all the tuples in $L_3$, $r_5$
and $r_6$, are processed, and the MBRs of $r_5$ and $r_6$ are used for window queries. The details
of $IRS J_p$ are shown in Algorithm 3.4.

3.4 Statistics for the IRS J Algorithms

In this section, we discuss the statistics for spatial joins and provide the formulas of the
population proportions and confidence intervals for each of the three $IRS J$ algorithms.
The proof of the incremental stratified random sampling is provided.

3.4.1 Population Proportion and Confidence Intervals

To provide bounds on the accuracy of the result, we incrementally calculate and return the
current estimate with a confidence interval to the user. Statistics of the $IRS J$ algorithms,
such as estimates and confidence intervals, are based on population proportion and the
Central Limit Theorem (CLT) [33, 58, 59]. CLT states that the sampling distribution of
the sample mean approximates a normal distribution for a specified number of samples
from any population. This approximation improves with large samples. We use the bi-
nominal probability distribution for statistics of $IRS J$. The outcome of each trial in the
binomial distribution is either 0 or 1. In spatial joins, either “intersect” or “does not inter-
sect” could be the outcome of each join. The binomial distribution is determined by the
number of trials $n$ and the probability $p$ of success in a single trial. The probability of a
success remains the same from one trial to the next. We assume that the normal curve is a
3.4 Statistics for the IRS J Algorithms

good approximation to the binomial distribution. Empirical studies have shown that these methods are quite good when both \( np > 5 \) and \( nq > 5 \), where \( q = 1 - p \) [59].

Now we discuss confidence intervals for the estimates of the final number of joins. If \( r \) is the number of successes out of \( n \) trials in a binomial experiment, we take the sample proportion of successes, \( \frac{r}{n} \), as our point estimate for \( p \), the population proportion of successes. Then the point estimate for \( p \) is \( \hat{p} = \frac{r}{n} \), and the point estimate for \( q \) is \( \hat{q} = 1 - \hat{p} \). The difference between the actual value of \( p \) and the estimate \( \hat{p} \) is called the error estimate for using \( \hat{p} \) as a point estimate for \( p \). For large samples, the distribution of \( \hat{p} \) is well approximated by a normal curve with mean \( \mu = p \) and standard error \( \sigma = \sqrt{\frac{pq}{n}} \). Since the distribution of \( \hat{p} \) is approximately normal, we use features of the standard normal distribution to find the difference \( \hat{p} - p \). An interval that estimates a population parameter within a range of possible values at a specified probability is called a confidence level. \( z_c \) is the critical value for confidence level \( c \) taken from a normal distribution such that an area equal to \( c \) under the standard normal curve falls between \( -z_c \) and \( z_c \). Clearly, we have \( P(-z_c \sqrt{\frac{pq}{n}} < \hat{p} - p < z_c \sqrt{\frac{pq}{n}}) = c \) [59]. \( E \) is the maximal error tolerance of the error of estimate \( |\hat{p} - p| \) for a confidence level. To find the confidence interval for \( p \), we have \( P(\hat{p} - E < p < \hat{p} + E) = c \). In most cases, we may not know the actual values of \( p \) or \( q \), thus we use our point estimates \( \hat{p} \) and \( \hat{q} \) to estimate \( E \). These estimates are safe for most practical purposes since we are dealing with large-sample theory. Then the confidence interval for \( p \) is \( \hat{p} - E < p < \hat{p} + E \). Therefore, we have \( E = z_c \sqrt{\frac{p(1-p)}{n}} = z_c \sqrt{\frac{pq}{n}} \) [59].

It is often the case that a calculation of the size of a sample is needed to obtain a certain level of confidence. Unfortunately, this calculation requires prior knowledge of the population standard deviation (\( \sigma \)). Realistically, \( \sigma \) is unknown. Often a preliminary sample is conducted so that a reasonable estimate of this critical population parameter
3.4 Statistics for the IRS J Algorithms

can be made. If such a preliminary sample is not made, but confidence intervals for the population mean are to be constructed using an unknown \( \sigma \), then the Student t-distribution can be used, which has less tight intervals than normal distribution [26, 58, 59]. If we use \( t_c \) instead of \( z_c \), then

\[
E = t_c \sqrt{\frac{\hat{p}(1-\hat{p})}{n}} = t_c \sqrt{\frac{\hat{p}\hat{q}}{n}},
\]

where \( t_c \) is the critical value for confidence level \( c \) taken from a Student t-distribution. The general CLT is specialized for the formulas of the confidence interval of different sampling methods [58].

Sampling in a GIS is more challenging due to data correlation. Similar values for a variable tend to occur in nearby locations thus leading to spatial clusters in GIS. This spatial clustering implies that many samples of geographical data will no longer satisfy the usual statistical assumption of independence of observations [2]. A major consequence of the dependence in a spatial sample is that statistical inference will likely not be as efficient as for an independent sample of the same size. The loss in efficiency may be remedied by increasing the sample size or by designing a sampling scheme that spaces observations such that their interaction is negligible. In the IRS J algorithms, the Student t-distribution is used to calculate confidence intervals, thus more samples are needed to get the same confidence intervals achieved by the normal distribution.

Assuming that the two relations \( R \) and \( S \) are joined, we are interested in estimating the number of intersections of the two data sets. In the IRS J algorithms, the population is the outer relation \( R \), and \( \hat{p} \) is the fraction of the elements in the sample that possess the characteristic of interest. The characteristic of interest in this problem is the “intersection”, i.e., \( \hat{p} \) is the fraction of tuples in the sample that have intersections with the inner relation \( S \). As shown in the previous formula, confidence intervals depend on the size of samples and the distribution of sample space (i.e., a normal distribution or Student t-distribution). Therefore, a confidence interval for an estimate of the final number of intersections is the same as that for an estimate of the final number of joins.
3.4 Statistics for the IRS $J$ Algorithms

3.4.2 The Confidence Interval for IRS $J_t$

Let $I_i = 1$ if the $i$th sampled tuple intersects tuples in $S$ and $I_i = 0$ otherwise. The total number of tuples in a sampling step of size $n_t$ intersecting tuples in $S$ is $I = \sum_{i=1}^{n_t} I_i$.

Then the sample proportion $\hat{p}$ is:

$$\hat{p} = \frac{\text{number of sampled tuples intersecting tuples in } S}{\text{number of tuples sampled from } R}$$

The following equations are used for a sampling step for IRS $J_t$:

Estimator of the population proportion $p$:

$$\hat{p} = \frac{I}{n_t} = \frac{\sum_{i=1}^{n_t} I_i}{n_t} \quad (3.1)$$

Estimate variance of $\hat{p}$:

$$\hat{V}(\hat{p}) = \frac{\hat{p}(1 - \hat{p})}{n_t - 1} \cdot \frac{N - n_t}{N} \quad (3.2)$$

Confidence interval:

$$E = t_c \sqrt{\frac{\hat{p}(1 - \hat{p})}{n_t - 1} \cdot \frac{N - n_t}{N}} \quad (3.3)$$

The IRS $J_t$ algorithm adds $n$ tuples each time to get a new estimate and confidence interval in every sampling step. In the experiments in Section 3.5, $n = 30$ is used. Therefore, equations (3.1), (3.2), and (3.3) are still valid for the proposed incremental random sampling process.
3.4 Statistics for the IRS J Algorithms

3.4.3 The Confidence Interval for IRS J

Since $N_i$ is the number of tuples in stratum $i$, and $n_i$ is the sample size for stratum $i$, clearly, $N = \sum_{i=1}^{k} N_i$, and $n_s = \sum_{i=1}^{k} n_i$.

The following equations are used for a sampling step for IRS J:

Estimator of the population proportion $p$:

$$\hat{p} = \frac{1}{N}(N_1 \hat{p}_1 + N_2 \hat{p}_2 + \ldots + N_k \hat{p}_k) = \frac{1}{N} \sum_{i=1}^{k} N_i \hat{p}_i, \text{ where } \hat{p}_i = \frac{I_{si}}{n_i} = \frac{\sum_{j=1}^{n_i} I_{ij}}{n_i} \quad (3.4)$$

Estimate variance of $\hat{p}$:

$$\hat{V}(\hat{p}) = \frac{1}{N^2}[N_1^2 \hat{V}(\hat{p}_1) + N_2^2 \hat{V}(\hat{p}_2) + \ldots + N_k^2 \hat{V}(\hat{p}_k)]$$

$$= \frac{1}{N^2} \sum_{i=1}^{k} N_i^2 \hat{V}(\hat{p}_i)$$

$$= \frac{1}{N^2} \sum_{i=1}^{k} N_i^2 \left( \frac{N_i - n_i}{N_i} \right) \left( \frac{\hat{p}_i \hat{q}_i}{n_i - 1} \right) \quad (3.5)$$

Confidence interval:

$$E = t_c \sqrt{\hat{V}(\hat{p})} \quad (3.6)$$

In IRS J, the sample size of each stratum is calculated in every sampling step, and it is proportional to the total number of tuples in the corresponding stratum. Lemma 3.4.1 and Theorem 3.4.2 show that equations (3.4), (3.5), and (3.6) are valid for the incremental stratified sampling process; thus, the IRS J algorithm is statistically sound.
3.4 Statistics for the IRS J Algorithms

Lemma 3.4.1. Let $N$ be the total number of tuples in the outer relation $R$ and $N_i$ be the number of tuples in stratum $i$. The sample size for a sampling step of IRS J is $n_s = \sum_{i=1}^{k} n_i$, where $k$ is the total number of strata. Then the sample size of stratum $i$ for a sampling step is:

$$ n_i = n_s \cdot \frac{N_i}{N} $$

Proof. Supposing that we have two strata $ST_1$ and $ST_2$, let $|ST_1| = N_1$ and $|ST_2| = N_2$ and $n_s = n_1 + n_2$ for any sampling step. Note that the sample size of each stratum is proportional to the total number of tuples in the corresponding stratum so that each tuple has to have the same probability. Thus, we assume that the two following conditions hold:

1. $n_1 \leq N_1$, and $n_2 \leq N_2$,
2. $\frac{n_1}{N_1}$ is close to $\frac{n_2}{N_2}$, and $n_2 = n_s - n_1$

Let $x = n_1$ and $f(x) = \frac{x}{N_1} / \frac{n-x}{N_2}$ and we want to have $f(x) = 1$. By solving the equation $\frac{x}{N_1} / \frac{n-x}{N_2} = 1$, we get $x = n_s \cdot \frac{N_1}{(N_1+N_2)}$. Therefore, the sample size of $ST_1$ is

$$ n_1 = x = n_s \cdot \frac{N_1}{N_1 + N_2} = n_s \cdot \frac{N_1}{N} \quad (3.7) $$

Similarly, the sample size of $ST_2$ is

$$ n_2 = n_s \cdot \frac{N_2}{N_1 + N_2} = n_s \cdot \frac{N_2}{N} \quad (3.8) $$

From equations (3.7) and (3.8), the generalized formula for the sample size of stratum $i$ is as follows:

$$ n_i = n_s \cdot \frac{N_i}{N} $$

$\square$
Theorem 3.4.2. Let $S_j$ be the incremental sample size for the $j$th sampling step. Assuming that $n_{sm}$ is the sample size for the $m$th sampling step, then $n_{sm}$ is the sum of the incremental sample sizes from the first step up to the $m$th step:

$$n_{sm} = \sum_{j=1}^{m} S_j$$

Proof. The incremental sample size $S_j$ is the actual number of tuples that we need to add into the sample for the $j$th step. Then the number of tuples that need to be processed in each sampling step is:

$$S_1 = s_{11} + s_{21} + s_{31} + \ldots + s_{k1}$$
$$S_2 = s_{12} + s_{22} + s_{32} + \ldots + s_{k2}$$
$$\ldots$$
$$S_m = s_{1m} + s_{2m} + s_{3m} + \ldots + s_{km} \quad (3.9)$$

Let $n_{s0} = 0$, and the current sampling step be the $j$th step. Then $s_{ij}$ can be calculated using the sample size of stratum $i$ for the $j$th step subtracted by the sample size of stratum $i$ for the $j-1$th step:

$$s_{ij} = \frac{n_{sj} \cdot N_i}{N} - \frac{n_{s(j-1)} \cdot N_i}{N} \quad (3.10)$$

By Lemma 3.4.1, and equations (3.9) and (3.10), the sample size of stratum 1 in the the $m$th sampling step is:

$$\sum_{j=1}^{m} s_{1j} = s_{11} + s_{12} + \ldots + s_{1m}$$
$$= \left( \frac{n_{s1}N_1}{N} - \frac{n_{s0}N_1}{N} \right) + \left( \frac{n_{s2}N_1}{N} - \frac{n_{s1}N_1}{N} \right) + \ldots + \left( \frac{n_{sm}N_1}{N} - \frac{n_{s(m-1)}N_1}{N} \right)$$
$$= \frac{n_{sm}N_1}{N}$$
3.4 Statistics for the IRS J Algorithms

Similarly, we get the sample size of each of the strata for the \( m \)th step as follows:

\[
\begin{align*}
\sum_{j=1}^{m} s_1i &= \frac{n_{m}N_1}{N} \equiv \text{the sample size of stratum 1 for the \( m \)th step} \\
\sum_{j=1}^{m} s_2i &= \frac{n_{m}N_2}{N} \equiv \text{the sample size of stratum 2 for the \( m \)th step} \\
\vdots \\
\sum_{j=1}^{m} s_ki &= \frac{n_{m}N_k}{N} \equiv \text{the sample size of stratum } k \text{ for the \( m \)th step}
\end{align*}
\]

Therefore, we have

\[
\sum_{j=1}^{m} S_j = \sum_{j=1}^{m} s_{1j} + \sum_{j=1}^{m} s_{2j} + \sum_{j=1}^{m} s_{3j} + \ldots + \sum_{j=1}^{m} s_{kj} \\
= \frac{n_{sm}N_1}{N} + \frac{n_{sm}N_2}{N} + \frac{n_{sm}N_3}{N} + \ldots + \frac{n_{sm}N_k}{N} \\
= \frac{n_{sm}}{N} \cdot (N_1 + N_2 + N_3 + \ldots + N_k) \\
= \frac{n_{sm}}{N} \cdot N = n_{sm}
\]

\[\square\]

3.4.4 The Confidence Interval for IRS \( J_p \)

Since \( I_{pj} = \sum_{i=1}^{T_j} I_i \), the average number of tuples within the \( j \)th page intersecting tuples in \( S \) is:

\[
A_j = \frac{I_{pj}}{T_j} = \frac{\sum_{i=1}^{T_j} I_i}{T_j}
\]

The following equations are used for a sampling step of IRS \( J_p \):

47
3.5 Experiments

Estimator of the population proportion $p$:

$$\hat{p} = \frac{\sum_{j=1}^{n_p} A_j}{n_p}$$  \hspace{1cm} (3.11)

Estimate variance of $\hat{p}$:

$$\hat{V}(\hat{p}) = \frac{\hat{p}(1 - \hat{p})}{n_p - 1} \cdot \frac{N_p - n_p}{N_p}$$  \hspace{1cm} (3.12)

Confidence interval:

$$E = t_c \sqrt{\frac{\hat{p}(1 - \hat{p})}{n_p - 1} \cdot \frac{N_p - n_p}{N_p}}$$  \hspace{1cm} (3.13)

Since we incrementally add one page each time for a new estimate and a confidence interval, the above equations are valid for the proposed incremental sampling process in the IRS $J_p$ algorithm.

3.5 Experiments

In this section, we present experimental results of the IRS $J_t$, IRS $J_s$, and IRS $J_p$ algorithms for both synthetic and real GIS data sets. The algorithms are compared with each other as well as with the full R-tree join algorithm [12]. We present results demonstrating the performance impact for using different index structures including R-trees, R*-trees, and STR-trees. We also present the number of I/Os for different buffer sizes and the number of node (page) accesses. For all presented results, the estimates and the corresponding confidence intervals are shown with a 90% or a 95% confidence level.
3.5 Experiments

3.5.1 Data sets and Experimental Methodology

We evaluated the IRS algorithms with both synthetic and real data sets. The synthetic data sets consist of (i) uniform (random) and (ii) skewed (hyper-exponential) distributions. For the uniform data set, \((x, y)\) locations are distributed uniformly and independently between 0 and 1. The \((x, y)\) locations for the skewed data sets are independently drawn from a hyper-exponential distribution with a mean 0.3 and a standard deviation 0.5. We generated hyper-exponentially distributed data sets because GIS data are often highly skewed distributed. The notations \(U\) and \(S\) were used to denote uniform and skewed distributed data sets, respectively. All possible join combinations of the data sets were conducted: \(S \bowtie S, U \bowtie S, S \bowtie U, U \bowtie U\). The number of tuples of each data set are varied between 100K and 1000K tuples. It is not necessary that outer and inner relations have the same size nor that their R-trees have the same height. Figure 3.4 illustrates the synthetic data set distribution.

![Uniform distribution](image1.png) ![Hyper-exponential distribution](image2.png)

(a) Uniform distribution  (b) Hyper-exponential distribution

*Figure 3.4: Synthetic data sets distribution*

3.5 Experiments

The two data sets have 300,434 and 199,850 tuples (MBRs), respectively. Figure 3.5 shows the two real data sets.

The experiments were conducted using the following parameters: R-tree page size of 4Kbytes with fan-out size of 100 for leaf and non-leaf nodes and minimum capacity of 40 for the R-tree, R*-tree and STR-tree. Since R-trees are disk-based index structures, the natural performance metric is the number of page I/Os required for a given buffer size. Assuming an LRU buffer, we vary the buffer size between 300 and 1800 pages resulting in the buffer holding between 3% and 60% of the R-trees. In Table 3.2 and Table 3.3, we present the index size of each data set and each of the three different index structures. On the average, the STR-tree has 33% fewer nodes than the R-tree, and the R*-tree has 5% fewer nodes than the R-tree.

3.5.2 Accuracy of Estimates

In this section, we present the accuracy of the \( IRS J_I \), \( IRS J_s \), and \( IRS J_p \) algorithms. The estimates and confidence intervals are plotted versus the percentage of data processed as well as the exact answer. By showing the deviations from the exact query answer, we
3.5 Experiments

### Synthetic data

<table>
<thead>
<tr>
<th>Size</th>
<th>100k</th>
<th>200k</th>
<th>300k</th>
<th>400k</th>
<th>500k</th>
<th>600k</th>
<th>700k</th>
<th>800k</th>
<th>900k</th>
<th>1000k</th>
</tr>
</thead>
<tbody>
<tr>
<td>R</td>
<td>1518</td>
<td>3077</td>
<td>4613</td>
<td>6170</td>
<td>7678</td>
<td>9202</td>
<td>10762</td>
<td>12271</td>
<td>13807</td>
<td>15372</td>
</tr>
<tr>
<td>R*</td>
<td>1487</td>
<td>2984</td>
<td>4376</td>
<td>5901</td>
<td>7355</td>
<td>8842</td>
<td>10246</td>
<td>11723</td>
<td>13243</td>
<td>14772</td>
</tr>
<tr>
<td>STR</td>
<td>1011</td>
<td>2021</td>
<td>3031</td>
<td>4041</td>
<td>5051</td>
<td>6063</td>
<td>7073</td>
<td>8083</td>
<td>9093</td>
<td>10105</td>
</tr>
</tbody>
</table>

**Table 3.2:** Index size of the synthetic data sets in pages (of 4KB)

### Real data

<table>
<thead>
<tr>
<th>Size</th>
<th>Mineral resources in US (300K)</th>
<th>Geochemical sediments in US (200K)</th>
</tr>
</thead>
<tbody>
<tr>
<td>R</td>
<td>4604</td>
<td>3172</td>
</tr>
<tr>
<td>R*</td>
<td>4400</td>
<td>3098</td>
</tr>
<tr>
<td>STR</td>
<td>3037</td>
<td>2020</td>
</tr>
</tbody>
</table>

**Table 3.3:** Index size of the real data sets in pages (of 4KB)

demonstrate that the IRS J algorithms provide good estimates.

In Figure 3.6, we present the results for synthetic data where the outer relation has 800,000 uniformly distributed tuples and the inner relation has 800,000 skewed-distributed tuples: $U_{-800K} \bowtie S_{-800K}$. In this experiment, both data sets are indexed by an R-tree. In Figure 3.6 (a), (c), and (e), we plot the exact answers and the running estimates returned by the algorithms versus the percentage of data (tuples) processed. The two smoother lines are for IRS $J_t$ and IRS $J_s$ whereas the jagged line is for IRS $J_p$. As can be seen in the figure, the accuracy of the estimates is significantly worse for IRS $J_p$ than for both IRS $J_t$ and IRS $J_s$. Both IRS $J_t$ and IRS $J_s$ rapidly achieve an estimate close to the exact answer. Only one sample can be used for an entire outer relation page of data as discussed in Section 2.2.2. Thus, the accuracy of IRS $J_p$ tends to be worse for the same amount of processed data. In Figure 3.6 (b), (d) and (f), the actual deviations of the estimates from the exact answers are plotted while showing the confidence intervals achieved for 90%
3.5 Experiments

Figure 3.6: Estimates and Actual deviation for synthetic data sets: $U$-800K $\bowtie$ $S$-800K (indexed by R-tree)
3.5 Experiments

Figure 3.7: Estimates and Actual deviation for the real data sets (indexed by R-tree)
3.5 Experiments

and 95% confidence levels. The top curve is the confidence interval for 95% confidence level, the middle one is the confidence interval for 90% confidence level, and the bottom jagged curve is the actual percent deviation of the estimates from the exact answer. As can be seen in Figure 3.6 (b), IRS \( J_t \) needs to process less than 1% of the data to obtain a 5% confidence interval with a 90% confidence level while, as can be seen in Figure 3.6 (f), IRS \( J_p \) needs to process approximately 20% of the data to obtain the same confidence interval. The result of IRS \( J_e \) was similar to that of IRS \( J_t \). It is clear that obtaining a given confidence interval requires far less data processing for the IRS \( J_t \) and IRS \( J_s \) algorithms than for the IRS \( J_p \) algorithm. The results of other experiments not shown, \( S \bowtie U, S \bowtie S \), and \( U \bowtie U \), are similar to the results shown in Figure 3.6.

In Figure 3.7, experimental results of the real data sets are presented. We plot the estimates, confidence intervals, and actual percent deviation from the exact answer for the three IRS \( J \) algorithms versus the percentage of data (tuples) processed. Again, the smoother estimate lines are for IRS \( J_t \) and IRS \( J_s \) while the jagged line is for IRS \( J_p \). Both IRS \( J_t \) and IRS \( J_s \) have better estimates than IRS \( J_p \) with the same number of tuples processed, and they need less data than IRS \( J_p \) for a given confidence interval.

Experiments of the IRS \( J \) algorithms for both the synthetic and the real data sets when indexed by the R*-tree and STR-tree provide similar qualitative results, and hence are omitted due to space considerations.

3.5.3 Time for Confidence Interval

Since cluster sampling generally achieves better results relative to simple random and stratified random samplings as described in Section 2.2.2, it might be expected that IRS \( J_p \) requires fewer I/Os than IRS \( J_t \) and IRS \( J_s \). However, the advantage of IRS \( J_p \) with respect to I/Os are less significant than the disadvantage of the size of data needed to achieve a given confidence interval. The experimental results show that both IRS \( J_t \) and IRS \( J_s \) re-
3.5 Experiments

quire fewer I/Os than IRS \( J_p \).

In Figure 3.8, we show results illustrating the impact of the index choice. We plot the confidence interval versus time measured as the number of buffer misses (I/Os) for the R-tree, R*-tree and STR-tree for the synthetic data sets \( U-800K \mapsto S-800K \) and the real data sets. The time to process a page of tuples in IRS \( J_p \) is greater than the time to process a tuple in IRS \( J_t \) and IRS \( J_s \). Thus, the most fair comparison is time, which is directly proportional to the number of buffer misses. Regardless of the index structure used, IRS \( J_p \) requires more I/Os to achieve the same confidence interval as IRS \( J_t \) and IRS \( J_s \). In Figure 3.8, the top curve is for IRS \( J_p \), the middle curve is for IRS \( J_t \), and the bottom curve is for IRS \( J_s \). The vertical line on the right is the execution time needed for the full R-tree join. Note that regardless of the index structure used, the time to obtain an exact answer is more than an order of magnitude greater than that needed to get an accuracy within a 5% confidence interval with a 90% confidence level using either IRS \( J_t \) or IRS \( J_s \). In all cases, the IRS \( J_s \) and IRS \( J_t \) algorithms have similar performance.

In Figure 3.9 (a), we plot the number of I/Os (buffer misses) versus data set size required by IRS \( J_t \) for several confidence intervals and also the number of I/Os for the full R-tree join. In Figure 3.9 (b), we plot the I/O ratio of the full R-tree join to IRS \( J_t \). The size of the data sets and buffer sizes are varied to keep the buffer size fixed at 10% of the size of the R-trees.

Figure 3.10 (a) shows the number of actual node accesses versus data set size, and we plot the node access ratio of the full R-tree join to IRS \( J_t \) in Figure 3.10 (b). As can be seen from these results, the full R-tree join requires 4-10 times more disk accesses than IRS \( J_t \) with a 5% confidence interval. As the data set size increases, the advantage of IRS \( J_t \) over the full R-tree join increases. Thus, for larger data sets, as expected in a real GIS, the benefit of our proposed approach will be even greater. Figure 3.11 shows the results for an experiment where the size of the synthetic data sets varies while the
3.5 Experiments

![Graphs showing confidence intervals and I/Os for synthetic and real data sets with different buffer sizes and tree structures.](image)

(a) Synthetic data sets w/3% buffer: STR  
(b) Real data sets w/3% buffer: STR  
(c) Synthetic data sets w/10% buffer: R*-tree  
(d) Real data sets w/10% buffer: R*-tree  
(e) Synthetic data sets w/10% buffer: R-tree  
(f) Real data sets w/10% buffer: R-tree

Figure 3.8: Confidence intervals and I/Os for synthetic $U\cdot800K \bowtie S\cdot800K$ and real data sets
3.5 Experiments

Figure 3.9: I/O comparison of IRS \( J \) and a R-tree join for varying size of synthetic data sets with 10% buffer size: \( R \)(uniform) \( 
\bowtie \bigcirc \) \( S \)(skewed) (indexed by R-tree)

Figure 3.10: Node access comparison of IRS \( J \) and a R-tree join for varying size of synthetic data sets with 10% buffer size: \( R \)(uniform) \( 
\bowtie \bigcirc \) \( S \)(skewed) (indexed by R-tree)

buffer size is fixed at 1200 pages. Again, the results show that IRS \( J \) provides an I/O performance improvement of more than an order of magnitude relative to the R-tree join.

In Table 3.4, the number of I/Os and R-tree join I/O ratio to IRS \( J \) are presented for the real data sets with buffer sizes 5% and 10% of the size of the R-trees, respectively. The
3.5 Experiments

Figure 3.11: I/O comparison of IRS \( J \), and a R-tree join for varying size of synthetic data sets with a fixed buffer size 1200: \( R(\text{uniform}) \bowtie S(\text{skewed}) \) (indexed by R-tree)

data sets are indexed by R-trees. For a buffer size of 10%, the full R-tree join requires 14 times more I/Os than IRS \( J \), with a 5% confidence interval. Figure 3.12 shows results of our experiments varying the buffer size from 300 to 1800 pages. The experimental results again show that IRS \( J \) provides a better performance than the full R-tree join in terms of I/Os. We also examine IRS \( J \) on the data sets indexed by the R*-tree and STR-tree. Similar results to the data sets indexed by the R-tree are obtained.

<table>
<thead>
<tr>
<th></th>
<th>Buffer Size</th>
<th>C.I.=10%</th>
<th>7%</th>
<th>5%</th>
<th>3%</th>
<th>2%</th>
<th>R-join</th>
</tr>
</thead>
<tbody>
<tr>
<td>I/Os</td>
<td>5%</td>
<td>863</td>
<td>1426</td>
<td>2695</td>
<td>7412</td>
<td>13206</td>
<td>27756</td>
</tr>
<tr>
<td></td>
<td>10%</td>
<td>625</td>
<td>1099</td>
<td>1590</td>
<td>5110</td>
<td>11272</td>
<td>24728</td>
</tr>
<tr>
<td>Node Accesses</td>
<td>5%</td>
<td>4044</td>
<td>7446</td>
<td>13689</td>
<td>35645</td>
<td>80735</td>
<td>262200</td>
</tr>
<tr>
<td></td>
<td>10%</td>
<td>3219</td>
<td>6390</td>
<td>11559</td>
<td>32909</td>
<td>79795</td>
<td>262200</td>
</tr>
</tbody>
</table>

Table 3.4: I/Os and node accesses of IRS \( J \), and a full R-tree join for the real data sets:
C.I. = Confidence Interval, Buffer size=Percent of relation size (indexed by R-tree)
3.5 Experiments

![Graph](image)

(a) R-tree join I/O ratio to $IRS_J_I$

(b) R-tree join Node Access ratio to $IRS_J_I$

Figure 3.12: I/O and node access comparison of $IRS_J_I$ and a R-tree join for real data sets
w/10% buffer (indexed by R-tree)

3.5.4 Index structures and $IRS_J$ Performance

In this section, we present results from experiments performed to see how index structures affect the result of the three $IRS_J$ variants.

In Figure 3.13, the number of I/Os needed to get an estimated result with a 5% confidence interval is plotted for each of $IRS_J_I$, $IRS_J_S$, and $IRS_J_P$ using the R-tree, R*-tree, and STR-tree, respectively. Figure 3.13 (a) shows the results for synthetic data sets, $U$-800K $\bowtie S$ -800K, and Figure 3.13 (b) shows the results for the real data sets. In both cases, it is clear that a better index structure results in better overall performance for each of our three $IRS_J$ variants. For all index structures, $IRS_J_I$ and $IRS_J_S$ result in significantly fewer I/Os than $IRS_J_P$ to achieve a 5% confidence interval and $IRS_J_S$ performs slightly better than $IRS_J_I$.

In Figure 3.14, we plot the number of needed I/Os for different confidence intervals using synthetic and real data sets, respectively, for the different index structures. In Figure 3.14, (a) and (b) correspond to STR-tree, (c) and (d) correspond to R*-tree, and (e) and (f)
3.5 Experiments

![Graphs showing I/Os of IRS J for 5% confidence interval in STR, R*-tree and R-tree](image)

(a) Synthetic data sets $U-800K \bowtie S-800K$

(b) Real data sets

**Figure 3.13:** I/Os of IRS J for 5% confidence interval in STR, R*-tree and R-tree

correspond to R-tree for the synthetic data sets and the real data sets, respectively. As expected, better index structures result in better I/O performance for all confidence intervals and IRS J variants. We note that the advantage of IRS J$^t$ and IRS J$s$ relative to IRS J$p$ is greater when the R-tree is used as the index structure. In the case of relatively smaller data sets, as the case is with the real data sets, IRS J$p$ performs better than IRS J$^t$ and IRS J$s$ for very tight confidence intervals (2%) since the variance is smaller when using IRS J$p$ as expected from the equations in Section 3.4, and as Figure 3.8 showed.

In Figure 3.15, we compare IRS J$^t$ and IRS J$s$. Figure (a) shows the I/O ratio of IRS J$s$ to that of IRS J$^t$ on the synthetic data sets, $800k \bowtie 800k$, and Figure (b) shows the result of the real data sets. We plot results for the three index structures given five confidence intervals, 10%, 7%, 5%, 3%, and 2%. Each line from the top to the bottom represents I/O ratio of IRS J$^t$ to that of IRS J$s$ with the data sets indexed by the STR-tree, R*-tree, and R-tree, respectively. IRS J$s$ performed better than IRS J$^t$ on the data sets indexed by either STR-tree or R*-tree. However, it resulted in more I/Os for some cases.
3.5 Experiments

Figure 3.14: I/Os of IRSJ comparison of STR, R*-tree and R-tree
3.6 Summary

We presented the Incremental Refining Spatial Join (IRSJ) algorithms to efficiently estimate the query results to spatial joins for vector data while allowing users to control the query processing. We implemented three variants of IRSJ: IRSJt using simple random sampling (tuple-level), IRSJs using stratified random sampling, and IRSJp using cluster sampling (page-level). We provided proofs of the statistical soundness of these variants and the empirical evaluations of their performances.

Our experimental results demonstrated that the time (I/Os) required to obtain a reasonably accurate estimate was an order of magnitude smaller relative to the time needed for an exact answer obtained using a full R-tree join algorithm with both synthetic and real data sets. We also observed that the query time reduction of IRSJt relative to the full R-tree join increased with data set size. Thus, the benefit of our approach will be
3.6 Summary

even greater for larger GIS data sets. Perhaps surprisingly, our simple random sampling
and stratified random sampling performed better than cluster sampling. This is because
only one datum per page can be used when calculating the confidence interval due to data
correlation within a leaf level page which violates the needed independence. In all the
experimental results, IRS $J_s$ performed at least as good as IRS $J_t$. Experiments using three
different index structures, the R-tree, R*-tree, and STR-tree, were conducted to see how
index structures affect the performance of IRS $J$. The choice of index structures had no ef-
fec on the relative performance of the three IRS $J$ algorithms. However, as expected, the
algorithms achieved convergence more rapidly to the specified confidence interval when
STR-trees and R*-trees were used.
Chapter 4

An Interactive framework for Raster Data Spatial Joins

4.1 Introduction

In this chapter, we propose a new query processing framework for online spatial join over raster data sets. Our goal is to achieve fast query response times by estimating results to raster data joins, and to provide visualization tools that allow truly interactive data exploration of raster based GIS applications. We investigate two statistical techniques for obtaining fast estimates of data spatial join results: Probabilistic Join (PJ) and Incremental Stratified Sampling Join (ISSJ).

The main idea of the PJ algorithm is to calculate the join probability and the expected number of joined cells of two raster data sets that have the same geographic coordinates. PJ is based on the augmented quad-tree data structure and data density (ratio of number of non-zero data cells to total number of data cells) in each node of the quad-tree is used to calculate the join probability. Figure 4.1 shows examples of raster data sets; R and S represented by a $4 \times 4$ raster grid. Suppose that $R$ has 8 non-zero data cells (density: 8/16) while $S$ has 9 non-zero data cells (density: 9/16). Then $R$ and $S$ must intersect regardless
4.1 Introduction

of their shape and location. The density of each data set can be used in the calculation of the join probability and the expected number of joined cells.

The Incremental Stratified Sampling Join (ISSJ) algorithm reports the running estimates of raster data joins, directed from the incremental random sampling technique proposed in Chapter 3. Using quad-trees, overlapping blocks (sub-regions) are used to filter candidate pairs in order to speed up the joining process. Our sampling join approach is based on stratified random sampling from quad-trees and performing joins on the incremental samples to estimate the final answers of spatial joins with bounded confidence intervals.

Figure 4.1: Raster cells of Datasets R and S

This new interactive framework utilizes the two proposed statistical approaches in order to speed up the process of obtaining incremental estimations of the final joins in a reasonable time compared to the total time needed to perform the full join. Augmented quad-trees with non-zero data cells are used in the framework. We provide experimental results for synthetic data sets and real GIS data sets that demonstrate the efficacy of our approach compared with full quad-tree joins.
4.2 A Framework for Spatial Joins Over Raster Data

4.1.1 Notations

The notation used in this chapter is summarized in Table 4.1.

<table>
<thead>
<tr>
<th>Notation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$ab$</td>
<td>the line segment from point $a$ to point $b$</td>
</tr>
<tr>
<td>$</td>
<td>ab</td>
</tr>
<tr>
<td>$[a_i,a_h]$</td>
<td>the closed interval from point $a_i$ to point $a_h$</td>
</tr>
<tr>
<td>$</td>
<td>A</td>
</tr>
<tr>
<td>$p(a,b)$</td>
<td>Join Probability of the two $n$-dimensional prosms $A$ and $B$</td>
</tr>
<tr>
<td>$E$</td>
<td>the expected number of joined cells</td>
</tr>
<tr>
<td>$R$</td>
<td>the outer raster data set joined</td>
</tr>
<tr>
<td>$S$</td>
<td>the inner raster data set joined</td>
</tr>
<tr>
<td>$N$</td>
<td>the size of population (the total number of data cells in $R$)</td>
</tr>
<tr>
<td>$k$</td>
<td>the total number of strata</td>
</tr>
<tr>
<td>$ST_i$</td>
<td>stratum $i$, where $i = 1, \ldots, k$</td>
</tr>
<tr>
<td>$N_i$</td>
<td>the total number of data cells in stratum $i$ ($ST_i$), where $i = 1, \ldots, k$</td>
</tr>
<tr>
<td>$n_{init}$</td>
<td>the initial incremental sample size for a sampling step</td>
</tr>
<tr>
<td>$n_i$</td>
<td>the sample size for a sampling step</td>
</tr>
<tr>
<td>$S$</td>
<td>the incremental sample size for a sampling step</td>
</tr>
<tr>
<td>$n_i$</td>
<td>the sample size of stratum $i$ for a sampling step, where $i = 1, \ldots, k$</td>
</tr>
<tr>
<td>$s_i$</td>
<td>the incremental sample size of strata $i$ for a sampling step, where $i = 1, \ldots, k$</td>
</tr>
<tr>
<td>$I$</td>
<td>the current total number of joined cells in a sampling step</td>
</tr>
<tr>
<td>$I_i$</td>
<td>the current total number of joined cells in a sampling step for stratum $i$, where $i = 1, \ldots, k$</td>
</tr>
<tr>
<td>$C_I$</td>
<td>a confidence interval</td>
</tr>
<tr>
<td>$EV$</td>
<td>an estimate of the total number of joined cells</td>
</tr>
</tbody>
</table>

Table 4.1: Notations used in $PJ$ and $ISSJ$

4.2 A Framework for Spatial Joins Over Raster Data

In this section, we present a new interactive framework for raster data spatial joins using Probabilistic Join ($PJ$) and Incremental Stratified Sampling Join ($ISSJ$).

4.2.1 Augmented Quad-tree

Augmented quad-tree data structure is used for $PJ$ and $ISSJ$. Specifically, we augment nodes to include the total number of non-zero data cells of the subtree below. The pro-
4.2 A Framework for Spatial Joins Over Raster Data

![Quadrant Diagrams](image)

Figure 4.2: Examples of augmented quad-tree of data sets $R$ and $S$

Proposed statistical approaches use these augmented quad-trees for obtaining information associated with the population. Figure 4.2 (a) and (b) show augmented quad-trees of the raster data set examples in Figure 4.1. The nodes of the quad-trees are displayed in counter-clockwise order starting from the north-west. In the framework, all data sets are indexed by augmented quad-trees.

4.2.2 Probabilistic Joins vs. Random Sampling

In $PJ$, the augmented value (number of non-zero data cells) of each node of two given data sets is used to calculate the join probability and the expected number of joined data cells for each pair of subregions in the two joined data sets. $PJ$ accesses nodes from the top to the bottom; hence, $PJ$ is referred to as a top-down approach.

$PJ$ does not need to access all levels of a quad-tree to calculate an estimate. It is sufficient to access only a small number of top levels. Thus, it can greatly reduce time-consuming disk I/O operations in practice. The number of levels to be accessed is a system parameter. The greater the number of levels is accessed, the more accurate the estimation can be. However, this results in a larger number of I/Os. In the experiments, in Section 4.5, we set the number of levels to 4 resulting in only 64 nodes needed in memory. Thus, it is practical to store required top level nodes of quad-trees for all joined data sets.
4.2 A Framework for Spatial Joins Over Raster Data

Although PJ provides no accuracy guarantee, our experimental results of synthetic and real data sets show the error bound is reasonably tight, e.g., a 9% error for 4th level join (Section 4.5).

In ISSJ, stratified random sampling is used to estimate the final answer of spatial joins. An accuracy guarantee is provided in the form of error bound confidence intervals. In contrast to PJ, ISSJ is performed on sampled leaf level data cells. Although fewer I/Os are required in ISSJ comparing to a full quad-tree join, obtaining a reasonable confidence interval requires a significantly greater number of I/Os compared to PJ.

4.2.3 Framework Overview

We proposed a query processing framework that combines the proposed statistical approaches. The framework consists of three main processes: probabilistic joins, result visualizations and sampling joins. The main idea is to use PJ and a visualization technique to allow users to discover “interesting” data set pairs and areas for further data exploration. Once users identify interesting data sets, they can have the system perform ISSJ in order to produce tighter running estimates of join results, or users can have the system use the full quad-tree join to obtain the exact answer.

Figure 4.3 shows the overview of the framework, where two relations $R$ and $S$ are joined. 1) Probabilistic Join (PJ): Given the user’s interesting data sets, all higher level nodes (from level 0 to level 3 in our experiments) of the two data sets’ quad-trees are loaded in memory. Then the join probability of each pair of the corresponding nodes is obtained from a look-up table. Since join probability is defined in continuous space, the system can use a lookup table for discrete values of join probability. 2) Visualization and user interface: Based on a visualized result of probabilistic joins, the user can identify “interesting join pairs”. 3) Incremental Stratified Sampling Join (ISSJ): ISSJ starts incremental sampling process with the interesting pairs. Samples (non-zero cells) are
randomly chosen from the outer relation $R$ using stratified random sampling. Spatial joining on the corresponding cells of the inner data set is performed. The number of joined cells in each step is used to calculate a running estimate and a confidence interval for the final result. Finally, the calculated running estimate and confidence interval are combined with the intermediate result into a query result through a visualization process. Then the query result is reported to the user. The user can stop the query process if the given confidence interval is sufficient or if the user sees satisfying trends from the visualized actual join locations (intermediate result); otherwise, each step of the process is repeated in an incremental manner to calculate new estimates until a desired confidence interval is achieved. Thus, the time to get join estimates needs to be compared to the time required for the full quad-tree join.
4.3 Probabilistic Join (PJ)

Given a set $X$ and two randomly chosen subsets $A$ and $B$ of $X$, we want to calculate the probability that $A \cap B \neq \emptyset$, denoted by $p$. There is an easy answer in the finite case.

Let $|X| = n$, $|A| = a$, $|B| = b$. Then $p = 1 - \left(1 - \left(\frac{a}{n}\right)\right)$, since this is the probability that a randomly chosen $b$-element subset of $X$ will not avoid a given $a$-element subset of $X$.

However, there is no reasonable answer in the infinite case, since we run into the well-known problems with (i) what is meant by “random” (the answer depends on how the experiment is conducted), (ii) measurability (how to determine the size of a set). Figure 4.4 illustrates the join probability $p$ and the expected number of joins $E$.

![Figure 4.4: Join probability](image)

We therefore restrict our attention to subsets of special kinds and use the obtained answers as approximations to the (unsolvable) general case.

**Theorem 4.3.1.** (Join Probability for 1-dimensional intervals)

Let $X = [0, 1]$, and let $A$, $B$ be randomly chosen intervals in $X$ of length $a$, $b$, respectively. Then, the probability $p$ that $A \cap B \neq \emptyset$ depends only on $a$, $b$, and can be calculated by:

$$p(a, b) = \frac{1}{1 - b} \int_0^{1-b} \frac{\min\{x + b, 1 - a\} - \max\{0, x - a\}}{1 - a} \, dx$$
4.3 Probabilistic Join (PJ)

Proof. Let $A$ and $B$ be $[a_l, a_h]$ and $[b_l, b_h]$, respectively, such that $a_l a_h = |A| = a$ and $b_l b_h = |B| = b$. If $A$ and $B$ are picked at random, then $a_l \in [0, 1 - a]$ and $b_l \in [0, 1 - b]$ (see Figure 4.5). Assuming that $x$ is a random variable for the value of $b_l$, we have $x \in [0, 1 - b]$.

Then $p(a, b)$, the probability that $A \cap B \neq \emptyset$ (A intersects B), is as follows:

$$p(a, b) \equiv P((a_h \geq b_l) \land (a_l \leq b_h))$$

$$\equiv P((a_h \geq x) \land (a_l \leq \min(x + b, 1 - a)))$$

$$\equiv P((a_l \geq x - a) \land (a_l \leq \min(x + b, 1 - a)))$$

$$\equiv P((a_l \geq \max(x - a, 0)) \land (a_l \leq \min(x + b, 1 - a)))$$

$$\equiv P(\max(x - a, 0) \leq a_l \leq \min(x + b, 1 - a))$$

In order to have $p(a, b) \neq 0$, we need to pick $a_l$ between $\max(x - a, 0)$ and $\min(x + b, 1 - a)$ from the continuous space in which the range of $x (b_l)$ is $[0, 1 - b]$ and the range of $a_l$ is $[0, 1 - a]$. Then we have the following equation.

$$p(a, b) = \frac{1}{(1 - a)(1 - b)} \int_0^{1 - b} \min(x + b, 1 - a) - \max(0, x - a) dx$$

Figure 4.6 an example of $1 - d$ join probability. Let $A$ and $B$ be $1 - d$ intervals in $X = [0, 1]$ of length $a = \frac{1}{4}$ and $b = \frac{1}{2}$, respectively. Then the probability $p$ that $A \cap B \neq \emptyset$
4.3 Probabilistic Join (PJ)

can be calculated by:

\[ p\left(\frac{1}{4}, \frac{1}{2}\right)_1 = \frac{1}{(1 - \frac{1}{2})(1 - \frac{1}{4})} \int_0^{1-\frac{1}{2}} (\min\{x + \frac{1}{2}, 1 - \frac{1}{4}\} - \max\{0, x - \frac{1}{4}\}) dx \] (4.1)

\[ = \frac{8}{3} \int_0^{\frac{1}{2}} (\min\{x + \frac{1}{2}, \frac{3}{4}\} - \max\{0, x - \frac{1}{4}\}) dx \]

In Figure 4.6, the area of the hatched region is \( p\left(\frac{1}{4}, \frac{1}{2}\right)_1 \), which is integrated part in the equation of Theorem 4.3.1.

![Figure 4.6: Example of 1 – d join probability](image)

Theorem 4.3.1 can now be generalized to any number of dimensions. The 2-dimensional case is as follows:

**Theorem 4.3.2. (Join Probability for 2-dimensional intervals)**

Let \( X = [0, 1]^2 \), and let \( A, B \) be rectangles in \( X \) of area \( a, b \), respectively. If the sides of \( A \) are of length \( a_1, a_2 = a/a_1 \) and the sides of \( B \) are of length \( b_1, b_2 = b/b_1 \), then we can use the 1-dimensional case to deduce that \( P(A \cap B \neq \emptyset) = p(a_1, b_1) \cdot p(a_2, b_2) \). However, we do not know \( a_1 \) and \( b_1 \). All we know is that \( a_1 \in [a, 1] \) (since the length of each side of \( A \)
4.3 Probabilistic Join (PJ)

has to be at least a) and b) $b_1 \in [b, 1]$. We therefore conclude that:

$$p(a, b)_2 = \frac{1}{(1-a)(1-b)} \int_a^1 \int_b^1 p(a_1, b_1) \cdot p\left(\frac{a}{a_1}, \frac{b}{b_1}\right) da_1 db_1$$

It is now easy to see the general formula for two n-dimensional prisms.

**Theorem 4.3.3.** (Join Probability for n-dimensional intervals) Let $A$ and $B$ be two n-dimensional prisms in $X = [0, 1]^n$ of volumes $a$, $b$, respectively. Let the lengths of sides of $A$ and $B$ be $(a_1, ..., a_n)$, $(b_1, ..., b_n)$, respectively. Then

$$p(a, b)_n = \frac{1}{(1-a)(1-b)} \int_a^1 \int_b^1 \cdots \int_{a_{n-1}}^1 \int_{b_{n-1}}^1 u db_{n-1} \cdot \cdots \cdot da_1$$

where $u = p(a_1, b_1) \cdot \cdots \cdot p(a_{n-1}, b_{n-1})p\left(\frac{a}{a_{n-1}}, \frac{b}{b_{n-1}}\right)$. The expected overlapped length (area, volume) of $A$ and $B$ can be calculated using the conditional probability, since it is assumed that the two data sets are chosen independently:

$$P(A \cap B) = P(A) \cdot P(B) = a \cdot b$$

The formulae for the join probability and the expected join numbers can be extended to multiway raster data joins where more than two data sets are involved.

**Theorem 4.3.4.** (General form of Theorem 4.3.1)

The general form of Theorem 4.3.1 can be written as follows:

$$p(a, b)_1 = \frac{1}{(1-a)(1-b)} \int_0^{1-a} \int_0^{1-b} X_{S(A,B)}(x, y) dxdy$$

(4.2)

$$X_{S(A,B)}(x, y) = \begin{cases} 
1 & \text{if } (x - b \leq y) \text{ and } (y \leq x + a) \\
0 & \text{otherwise}
\end{cases}$$

where $\max(x - b, 0) \leq y \leq \min(x + a, 1)$. 

73
4.4 Incremental Stratified Sampling Join (ISSJ)

We can expand (4.2) for the join probability of three $1 - d$ intervals.

**Theorem 4.3.5. (General form of Theorem 4.3.1 for three intervals)**

Let $X = [0, 1]$, and let $A$, $B$ and $C$ be randomly chosen $1 - d$ intervals in $X$ of length $a$, $b$, and $c$, respectively. Then, the probability $p$ that $A \cap B \cap C \neq \emptyset$ can be calculated by:

$$p(a, b, c) = \frac{1}{1 - a} \frac{1}{1 - b} \frac{1}{1 - c} \int_0^{1-a} \int_0^{1-b} \int_0^{1-c} X_{S(A,B,C)}(x,y,z) dxdydz$$

$$X_{S(A,B,C)}(x,y,z) = \begin{cases} 
1 & \text{if } (x - b \leq y) \text{ and } (y \leq x + a) \\
& \text{and } (x - c \leq z) \text{ and } (z \leq x + a) \\
& \text{and } (y - c \leq z) \text{ and } (z \leq y + b) \\
0 & \text{otherwise}
\end{cases}$$

where $\max\{x - b, 0\} \leq y \leq \min\{x + a, 1\}$ and $\max\{x - c, y - c, 0\} \leq z \leq \min\{x + a, y + b, 1\}$.

4.4 Incremental Stratified Sampling Join (ISSJ)

Sampling methods are used to estimate the final result from a subset (samples) of the data and to provide a bounded confidence interval. Query estimations and confidence intervals are statistically meaningful only if samples are retrieved at random. A weighted random sampling method, Acceptance/Rejection [49], is used in ISSJ. We study the stratified random sampling without replacement for raster data spatial joins. Each sampling is conducted in an incremental manner and the performance is evaluated with varying data sets and buffer sizes.

4.4.1 Stratified Random Sampling

Stratified random sampling is chosen because its property matches the property of quadtrees that provides systematic decomposition of a space with no overlaps between subregions. In stratified random sampling, the given region (population of all data cells) is
**4.4 Incremental Stratified Sampling Join (ISSJ)**

divided into a number of non-overlapping subregions called strata. Then each stratum contains a set of raster data cells. Stratified random sampling can result in smaller error bounds on an estimation and can reduce the sampling cost [59].

If strata are well-defined and the population is carefully divided into strata, this method may provide several advantages: it can result in smaller error bounds on the estimation and reduce the sampling cost [59]. Hence, construction of strata is important in order to get optimal results. However, implementing techniques for defining strata is not our main focus. In our algorithm, stratification is based on non-overlapping geometric forms such as rectangles (nodes at each level). We define the internal nodes of the quad-tree for a given level as strata, i.e., the second level nodes of quad-tree are used as strata in our experiments. We assume that the strata is pre-defined in our experiments. Algorithm 4.1 describes the **ISSJ** algorithm.

Samples (non-zero cells) are then randomly chosen from each stratum by conducting simple random sampling. The sample size of each stratum \( n_i, i=1,..,k \), is calculated for every sampling step, and it is proportional to the total number of non-zero cells within that stratum. Then the sampling size for a sampling step is \( n_s = \sum_{i=1}^{k} n_i \). If the value of the chosen data cell is 1, searching the corresponding joined cell of the inner data set is performed in the quad-tree of the inner data set (line 15 of Algorithm 4.1). If the value of the corresponding cell is 1, then two data cells join. For each stratum, we obtain the number of joined cells, and this number is used to calculate the estimate and confidence interval for the corresponding stratum. The sum of the joined cells of each stratum is the current intermediate result, and the estimates and confidence intervals of all strata are combined for an estimate and a confidence interval of the final answer. The user can stop the query process if the given confidence interval is sufficient, otherwise the process repeats.
4.4 Incremental Stratified Sampling Join (ISSJ)

Algorithm 4.1 ISSJ(R, S, ST)
1: ST = [ST₁, . . . , STₖ] // ST is a set of strata
2: I₁, . . . , Iₖ ← 0 // current joined cells for stratum i
3: nₛ ← 0; nₘᵢₙᵢ ← 30 // sample size; initial incremental sample size for a sampling
4: n₁, . . . , nₖ ← 0; s₁, . . , sₖ ← 0 // sample size; incremental sample size for stratum i
5: repeat
6: compute s₁, s₂, . . . , sₖ for ST₁, ST₂, . . . , STₖ using nₘᵢₙᵢ
7: S ← ∑ₖᵢ₌₀ sᵢ; nₛ ← nₛ + S
8: for i = 1 to k do
9:   nᵢ ← nᵢ + sᵢ
10:  for j = 1 to sᵢ do
11:     L ← choose a leaf from STᵢ at random
12:     cᵣ ← choose a non-zero cell from L at random
13:     if cell cᵣ’s value is 1 then
14:       Pᵣ ← the center point of the chosen cell cᵣ
15:       cₛ ← findJoinedCell(S, Pᵣ)
16:       if cell cₛ’s value is 1 then
17:         Iᵢ ← add 1
18:     end if
19:   end if
20:  remove cᵣ from L
21: end for
22: remove L from STᵢ if L is empty
23: end for
24: I ← ∑ₖᵢ₌₀ Iᵢ
25: C_I ← Compute a confidence interval w/all Iᵢ and nᵢ
26: EV ← Compute an estimate w/all Iᵢ and nᵢ
27: report EV, C_I, and I
28: until C_I is sufficient to the user or all STᵢ are empty
4.4 Incremental Stratified Sampling Join (ISSJ)

4.4.2 Estimates for Stratified Random Sampling

To provide bounds on the accuracy of ISSJ, we incrementally calculate the current estimate with a confidence interval. The estimates and confidence intervals of ISSJ are based on population proportion and the Central Limit Theorem (CLT) [33, 59]. We use the binomial probability distribution [59] for statistics of ISSJ. In ISSJ, the population is the number of non-zero cells of the outer relation R and \( \hat{p} \) is the fraction of the elements in the sample that possess the characteristic of interest ("join" in our algorithm). Hence \( \hat{p} \) is the fraction of cells in the sample that joins with the corresponding cell of the inner relation S. Confidence intervals depend on the size of samples and the distribution of the sample space (i.e., Student t-distribution).

Let \( N \) be the size of population (total number of non-zero cells of the outer data sets) and \( n_s \) be the sample size for a sampling step. If \( N_i \) is the number of non-zero cells in stratum \( i \), and \( n_i \) is the sample size for stratum \( i \), then \( N = \sum_{i=1}^{k} N_i \), and \( n_s = \sum_{i=1}^{k} n_i \), where \( k \) is the number of strata. Let \( I_i \) be the total number of cells that join the corresponding cells of \( S \) in stratum \( i \). The following equations are used for a sampling step for ISSJ:

Estimator of the population proportion, where \( \hat{p}_i = \frac{I_i}{n_i} \) and \( \hat{q}_i = 1 - \hat{p}_i \):

\[
\hat{p} = \frac{1}{N} (N_1 \hat{p}_1 + N_2 \hat{p}_2 + \ldots + N_k \hat{p}_k) = \frac{1}{N} \sum_{i=1}^{k} N_i \hat{p}_i. \tag{4.3}
\]

Estimate variance of \( \hat{p} \):

\[
\hat{V}(\hat{p}) = \frac{1}{N^2} \sum_{i=1}^{k} N_i^2 \left( \frac{N_i - n_i}{N_i} \right) \left( \frac{\hat{p}_i \hat{q}_i}{n_i - 1} \right) \tag{4.4}
\]

Confidence interval:

\[
E = t_c \sqrt{\hat{V}(\hat{p})}, \tag{4.5}
\]
4.5 Experiments

where $t_c$ is the critical value for confidence level $c$ taken from a Student t-distribution.

Equations (4.3), (4.4) and (4.5) are valid for the incremental stratified sampling process. The proof of incremental equations can be found in Section 3.4.3.

4.5 Experiments

In this section, we present experimental results of the Probabilistic Join (PJ) and Incremental Stratified Sampling Join (ISSJ) with both synthetic and real GIS data sets. The performance of PJ and ISSJ are compared with each other as well as with the full quad-tree join. All experiments showed similar qualitative and quantitative comparisons.

4.5.1 Data Sets and Experimental Methodology

In our experiments, we consider both synthetic and real data sets shown in Table 4.2 and Table 4.3. We generated four sets of uniformly distributed raster data and four sets of exponentially distributed raster data (a mean of 0.3 and a standard deviation of 0.3). Our real data sets are from the 2001 and 2005 U.S. Geological Survey [67]: six data sets are chosen from Arizona, Colorado, Oregon and Wyoming in the US. These data sets are about minerals, stream sediments, water sediments, rocks, pluto sediments and unconsolidated sediments. Each data set was converted into raster format. In Table 4.2 and Table 4.3, the total number of data cells (pixels) is presented along with the total number of non-zero data cells and the data density for the synthetic and real data sets.

It is necessary that both the outer and inner data sets are indexed by augmented quad-trees, and they have the same number of data cells as well as the same size of cells. Our experiments were conducted using the following parameters: Augmented quad-trees are implemented for PJ and ISSJ while nonaugmented quad-trees are used for the full quad-tree join. The page size of the quad-tree was set to 4Kbytes, resulting in 100 nodes and
4.5 Experiments

<table>
<thead>
<tr>
<th>synthetic data sets</th>
<th>uni1</th>
<th>uni2</th>
<th>uni3</th>
<th>uni4</th>
<th>exp1</th>
<th>exp2</th>
<th>exp3</th>
<th>exp4</th>
</tr>
</thead>
<tbody>
<tr>
<td># total cells</td>
<td>65536</td>
<td>65536</td>
<td>262144</td>
<td>262144</td>
<td>65536</td>
<td>65536</td>
<td>262144</td>
<td>262144</td>
</tr>
<tr>
<td># N.E. cells</td>
<td>17325</td>
<td>28365</td>
<td>39120</td>
<td>48298</td>
<td>14256</td>
<td>24736</td>
<td>36290</td>
<td>45231</td>
</tr>
<tr>
<td>density</td>
<td>0.26</td>
<td>0.43</td>
<td>0.15</td>
<td>0.18</td>
<td>0.22</td>
<td>0.38</td>
<td>0.14</td>
<td>0.17</td>
</tr>
<tr>
<td>description</td>
<td>uniformly distributed data</td>
<td>exponentially distributed data</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 4.2: Synthetic data sets: N.E. = non empty data cells

<table>
<thead>
<tr>
<th>real data sets</th>
<th>AZ</th>
<th>CO</th>
<th>OR</th>
<th>WY</th>
</tr>
</thead>
<tbody>
<tr>
<td># total cells</td>
<td>65536</td>
<td>65536</td>
<td>65536</td>
<td>65536</td>
</tr>
<tr>
<td># N.E. cells</td>
<td>10202</td>
<td>23030</td>
<td>23821</td>
<td>42321</td>
</tr>
<tr>
<td>density</td>
<td>0.26</td>
<td>0.43</td>
<td>0.15</td>
<td>0.18</td>
</tr>
<tr>
<td>description</td>
<td>6 data sets of mineral resources from USGS</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 4.3: Real data sets: N.E. = non empty data cells

64 nodes for the non-augmented tree and augmented tree, respectively. We performed comparisons assuming an LRU replacement policy with buffer sizes of 5%, 10% and 20% of the size of one of the two relations. For all presented results, the estimates and the corresponding confidence intervals are shown with a 95% confidence level.

4.5.2 Experimental Results

First, we present the accuracy of join probability using the 1-d formula \(p_1\) and 2-d formula \(p_2\) discussed in Section 4.3. The total number of joins obtained by the 1-d and 2-d join probability were compared with the total number of actual joins. For discrete values of join probability, we created two lookup tables \((20 \times 20)\). Table 4.4 illustrates a portion of 2-d join probabilities such as entries from 2-d lookup table used in the experiments. We randomly selected two corresponding nodes from the quad-trees of two real data sets. We
4.5 Experiments

checked the occupancy rates (non-zero data cells/total data cells) in the two chosen nodes and obtained the 1-d and 2-d join probabilities from the lookup tables. Then the expected numbers of joins were calculated. We repeated this process for varying sizes of sample pairs: 5%, 10%, 20% and 50% of the total quad-tree nodes. We ran the experiment 10,000 times with each of the sample sizes and presented the average. In Table 4.5, we show the results of a join, unconsolidated sediments ⊲⊳ minerals in CO. The table entries are actual error values, thus, for example, an error of 0.1060 is a 10.60% error. Clearly, the 2-d join probability provides better approximation of the actual join.

<table>
<thead>
<tr>
<th>$P$</th>
<th>0.2</th>
<th>0.4</th>
<th>0.6</th>
<th>0.8</th>
<th>1.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.2</td>
<td>0.7683</td>
<td>0.9277</td>
<td>0.9903</td>
<td>1.0</td>
<td>1.0</td>
</tr>
<tr>
<td>0.4</td>
<td>0.9277</td>
<td>0.9937</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
</tr>
<tr>
<td>0.6</td>
<td>0.9903</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
</tr>
<tr>
<td>0.8</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
</tr>
<tr>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
</tr>
</tbody>
</table>

**Table 4.4:** Example of a lookup table for 2-d join probability

<table>
<thead>
<tr>
<th>sample size</th>
<th>actual join</th>
<th>2-d (error)</th>
<th>1-d (error)</th>
</tr>
</thead>
<tbody>
<tr>
<td>5 %</td>
<td>54</td>
<td>48 (0.1060)</td>
<td>39 (0.2778)</td>
</tr>
<tr>
<td>10 %</td>
<td>109</td>
<td>99 (0.0917)</td>
<td>78 (0.2844)</td>
</tr>
<tr>
<td>20 %</td>
<td>218</td>
<td>197 (0.0963)</td>
<td>155 (0.2889)</td>
</tr>
<tr>
<td>50 %</td>
<td>545</td>
<td>494 (0.0936)</td>
<td>389 (0.2862)</td>
</tr>
</tbody>
</table>

**Table 4.5:** 1-d vs. 2-d Probabilistic Joins

To evaluate the quality of the “big picture” visualization obtained by PJ, we calculated the expected number of joins using the 4th level tree nodes. When using the 4th levels of two quad-trees, only 64 subregions are joined. As a result, users can obtain the approximate result visualization in near instantaneous time with a truly interactive manner. We
4.5 Experiments

<table>
<thead>
<tr>
<th>join data sets</th>
<th>real data sets</th>
<th>synthetic data sets</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>AZ</td>
<td>CO</td>
</tr>
<tr>
<td>average diff.</td>
<td>0.0060</td>
<td>0.0087</td>
</tr>
<tr>
<td>minimum of max. diff.</td>
<td>0.0047</td>
<td>0.0038</td>
</tr>
<tr>
<td>maximum of max. diff.</td>
<td>0.1208</td>
<td>0.0973</td>
</tr>
<tr>
<td>average max. diff.</td>
<td>0.0329</td>
<td>0.0237</td>
</tr>
<tr>
<td>average error of estimates</td>
<td>0.1105</td>
<td>0.0729</td>
</tr>
</tbody>
</table>

Table 4.6: PJ (4th level) vs. actual joins

Present results showing the difference between the PJ method and the full quad-tree join method (see Table 4.6). For the real data sets we compared PJ and ISSJ for all 15 possible pairwise joins of the 6 data sets. We grouped the synthetic data sets into two: group 1 (uni1, uni2, exp1, exp2) and group 2 (uni3, uni4, exp3, exp4). We computed all possible 6 pairwise joins of each of the two groups. In Table 4.6, we present the average differences in the join density. The minimum and maximum of maximum difference and the average maximum difference are also presented. Finally, we calculated the average error in the expected number of joins of all the pairwise joins. As can be seen, PJ is reasonably accurate in all the cases of both real and synthetic data sets. With real data sets, PJ resulted in less accuracy due to the scattered clusters found in the data sets. As shown in Figure 4.9, for the data we explored, these modest inaccuracies have little effect on the overall visual join-result appearance.

Next, we present the performance of ISSJ compared to the augmented full quad-tree join. Figure 4.7 shows the result using the synthetic and real data sets (minerals ⊲⊳ unconsolidated sediments from Colorado). The estimates and confidence intervals are plotted versus the number of samples (non-zero data cells) processed as well as the exact answer. Figure 4.7 (a) and (b) show the estimated values of the final joins calculated by ISSJ for the synthetic and real data sets, respectively. Figure 4.7 (c) and (d) show how
4.5 Experiments

Figure 4.7: Estimates, confidence intervals and I/Os of $ISSJ$: synthetic (uni1 $\bowtie$ exp1) and real data sets (unconsolidated sediment $\bowtie$ mineral in CO)
fast the confidence intervals converge. By showing the deviations from the actual joins, we demonstrate that IS S J provides good estimates of the final answer. In Figure 4.7 (e) and (f), we show how fast an accurate estimation could be calculated compared to the time required for the full quad-tree join. For example, it took about 1900 I/Os to reach an estimate with a 5% confidence interval while 8,000 I/Os were required for the exact answer by using the full quad-tree join.

We next show how accurately the proposed approaches provide a “big picture” of the actual join. Figure 4.8 (a), (b) and (c) show three data sets for the state of Colorado: chemical sediments (P), minerals (Q) and water sediments (S). The results of PJ and ISS J for P ⊳ Q and Q ⊳ S are presented as well as that of the actual join. The result from the top to the bottom corresponds to: ISS J with a 10% confidence interval (a), ISS J with a 5% confidence interval (b), actual joins (c) and finally PJ of the 4th level nodes (d). PJ and ISS J with a 5% confidence interval provide a reasonably accurate approximation of the actual join.

![Chemical sediments (P)](image1)
![Minerals (Q)](image2)
![Water sediments (S)](image3)

Figure 4.8: Real data sets: Mineral resources in Colorado in the U.S.

In Figure 4.10, we present I/O comparisons between PJ and ISS J with varying the confidence intervals, as well as with the full nonaugmented quad-tree join (QT). All possible pairwise joins from the six data sets of CO and AZ were run and the number
4.5 Experiments

Figure 4.9: \( ISSJ \) vs. actual joins vs. \( PJ \) for real data sets in CO
4.5 Experiments

![Graphs showing number of I/Os for different methods](image)

**Figure 4.10:** Number of I/Os of PJ, ISSJ and the full quad-tree join

of I/Os were plotted for buffer sizes of 5%, 10% and 20% of the size of one data set quad-tree. We plot the average total number of I/Os of each method averaged over all 15 pairwise joins. The results for PJ are on the left, then ISSJ for confidence interval bounds of 10, 7, 5, 3, 2 and 1%, and finally the results for the full quad-tree join on the right. Note that the performance difference varying buffer sizes is very small since there is few re-visiting of the leaf nodes, and thus little opportunity to benefit from buffer caching exits.

The PJ algorithm resulted in up to two orders of magnitude fewer I/Os than QT for both data sets. The ISSJ algorithm obtained a very reasonable confidence interval (e.g. 5%) with far fewer I/Os compared to QT. PJ is significantly faster than ISSJ but does not provide correctness bounds. However, PJ does provide a good overall picture for the data explored.
4.6 Summary

In this chapter, we presented two statistical approaches for estimating spatial joins on quad-tree indexed raster data, namely, *Probabilistic Join (PJ)* and *Incremental Stratified Sampling Join (ISSJ)*. *PJ* allows users to obtain approximate answers in near instantaneous time, and *ISSJ* is a random sampling technique that provides query estimates bounded by confidence intervals. We then proposed a new interactive framework that combines the proposed statistical approaches in order to enhance online query processing. The new framework allows fast interactive data exploration and the opportunity for the user to then drill down with full spatial joins if desired.

We provide experimental results for synthetic data sets and real GIS data sets that demonstrate the efficacy of our approach compared with full quad-tree joins. The speedup relative to the full join increases with data set size. Experimental evaluation showed that *PJ* resulted in an approximate visualization of a reasonably accurate answer, e.g., a 9% error using the 4th levels of quad-trees, with near instantaneous response time. The *ISSJ* algorithm, while not as fast as the *PJ* algorithm, provided results with bounded confidence intervals up to an order of magnitude faster than a full quad-tree join.
Chapter 5

Spatial Range Queries on Web Data using $k$-Nearest Neighbor Search

5.1 Introduction

In this chapter, we present a set of spatial range query algorithms that efficiently retrieve georeferenced data through restrictive Web interfaces. Our focus is on finding all objects within a given rectangular shaped query region $R$ using a minimal number of $k$-NN searches as supported in various Web applications. The complexity of this problem is unknown. Therefore only approximation results are provided. In reality, there can be more than one Web site involved in the given range query. Then, the general problem is to find all objects within the query region $R$ using $k$-NN searches on each of these Web data and to integrate the results from the sources into one final query result. The examples in Figure 5.1 (a) and (b) show examples of two different $k$-NN interfaces, $k=3$ in (a) and $k=6$ in (b). The optimal solution should find a minimal set of query locations that minimizes the number of $k$-NN searches to completely cover $R$.

We consider all possible $k$-NN interfaces in our approach. Any Web applications supporting $k$-NN search can be classified based on the value of $k$.
5.1 Introduction

(a) A range query using 3-NN search (b) A range query using 6-NN search

**Figure 5.1:** Range queries on Web data

1. The value of $k$ is fixed depending on Web applications. In many Web applications, the user has no knowledge about the data set at the server and no control over the value of $k$. These applications return a fixed number of $k$ nearest objects to the user’s query point. Different applications may have different values of $k$. A typical value of $k$ in real applications ranges from 5 to 20 [14].

2. The value of $k$ can be any positive integer determined by users. Some Web applications accept user-defined $k$ values for their $k$-NN searches.

3. The value of $k$ can be any positive integer less than or equal to a certain value. Some Web applications support $k$-NN searches with user-defined $k$ values but allow only a bounded range of $k$ values.

To simplify the discussion, we made the following assumptions for practical solutions: 1) a range query will be supported by a single Web data source; 2) a Web application supports only one value of $k$; 3) a $k$-NN search takes a query point $q$ as an input parameter.

Based on the classification of $k$-NN interfaces on the Web, we propose four algorithms: the Quad Drill Down (QDD) and Dynamic Constrained Delaunay Triangulation.
5.1 Introduction

(DCDT) algorithms for the case where we have no knowledge about the distribution of the data sets with the most restrictive \(k\)-NN interface (i.e., fixed values of \(k\)), the Density-Based (DB) algorithm for the case where we have some available statistical knowledge about the distribution of the data set with the most flexible \(k\)-NN interface (i.e., a value of \(k\) is a user input), and finally, the Hybrid (HB) of QDD and DB. The efficiencies of the algorithms are measured by statistical analysis and empirical results. We present efficiencies of the proposed algorithms measured through empirical experiments. The statistical analysis of two algorithms, QDD and DB, are also discussed.

5.1.1 Notations

The notations in Table 5.1 are used throughout this chapter and Figure 5.2 illustrates an example of range query using 5-NN search.

<table>
<thead>
<tr>
<th>Notation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>(R)</td>
<td>a query region (rectangle)</td>
</tr>
<tr>
<td>(P_{ll})</td>
<td>the lower-left corner point of (R)</td>
</tr>
<tr>
<td>(P_{ur})</td>
<td>the upper-right corner point of (R)</td>
</tr>
<tr>
<td>(q)</td>
<td>the query point for a (k)-NN search</td>
</tr>
<tr>
<td>(R_{q})</td>
<td>half of the diagonal of (R); the distance from (q) to (P_{ll})</td>
</tr>
<tr>
<td>(C_{Rq})</td>
<td>circle inscribing (R); the circle of radius (R_{q}) centered at (q)</td>
</tr>
<tr>
<td>(P_{k})</td>
<td>the set of (k) nearest neighbors obtained by a (k)-NN search ordered ascendingly by distance from (q), i.e., (P_{k} = {p_1, p_2, p_3, p_4, p_5}) in Figure 5.2</td>
</tr>
<tr>
<td>(r)</td>
<td>the distance from (q) to the farthest point (p_k) in (P_k)</td>
</tr>
<tr>
<td>(C_r)</td>
<td>(k)-NN circle; the circle of radius (r) centered at (q)</td>
</tr>
<tr>
<td>(C_{\epsilon r})</td>
<td>tighter bound (k)-NN circle; the circle of radius (\epsilon \cdot r) centered at (q)</td>
</tr>
<tr>
<td>(\epsilon)</td>
<td>a constant factor, where (0 &lt; \epsilon &lt; 1)</td>
</tr>
</tbody>
</table>

Table 5.1: Notations for range queries using \(k\)-NN search
5.2 Range Queries using fixed values of $k$: QDD and DCDT Algorithms

In many Web applications, the client has no knowledge of the data set at the server and no control over the value of $k$ (i.e., $k$ is fixed). Considering a typical value of $k$ in real applications, which ranges from 5 to 25 [14], in general, multiple queries are evaluated to completely cover a reasonably large $R$.

Our approach to the general range query problem on the Web is as follows: 1) divide the query region $R$ into subregions, $R = \{R_1, R_2, ..., R_m\}$, such that any $R_i \in R$ can be covered by a single $k$-NN search, 2) obtain the resulting points from the $k$-NN search on each of the subregions, 3) combine the partial results to provide a complete result to the original range query $R$. Then, the main focus is how to divide $R$ so that the total number of $k$-NN searches can be minimized. We consider the following three approaches:

1. A naive approach: divide the entire $R$ into equi-sized subregions (grid cells) such that any cell can be covered by a single $k$-NN search.

2. A recursive approach: conduct a $k$-NN search in the current query region, divide
the current query region into subregions with same sizes if the \( k \)-NN search fails to cover \( R \), and call \( k \)-NN search for each of these subregions. Repeat the process until all subregions are covered.

3. A greedy and incremental approach: divide \( R \) into subregions with different sizes. Then select the largest subregion for the next \( k \)-NN search. Check the covered region by the \( k \)-NN search and select the next largest subregion for another \( k \)-NN search.

If \( r > R_q \), then the \( k \)-NN search must have returned all the points inside \( C_{R_q} \). Therefore, we can obtain all the points inside the region \( R \) after pruning out any points of \( P_k \) that are outside \( R \) but inside \( C_r \). Then, the range query is complete. Otherwise, we need to perform some additional \( k \)-NN searches to completely cover \( R \). For example, if we use a 5-NN search for the data set shown in Figure 5.2, then the resulting \( C_r \) is smaller than \( C_{R_q} \). For the rest of this section, we focus on the latter case that is a more realistic scenario.

Note that it is not sufficient to have \( r \geq R_q \) in order to retrieve all the points inside \( R \) when \( q \) has more than one \( k^{th} \) nearest neighbor since one of them will be randomly returned as \( q \)'s \( k^{th} \) nearest neighbor. The condition, \( r > R_q \), is required by assuming four points in the four corners of \( R \) in Figure 5.2. As a result, we use \( r > R_q \) as a terminating condition in our algorithms.

In a naive approach, \( R \) is divided into a grid where the size of cells is small enough to be covered by a single \( k \)-NN search. However, it might not be feasible to find the exact cell size with no knowledge of the data sets. Even in the case that we obtain such a grid, this approach is inefficient due to large amounts of overlapping areas among \( k \)-NN circles and wasted time to search empty cells considering that most real data sets are not uniformly distributed. Therefore, we provide a recursive approach and an incremental approach in the following subsections.
5.2 Range Queries using fixed values of \( k \): QDD and DCDT Algorithms

5.2.1 Quad Drill Down (QDD) Algorithm

We propose a recursive approach, the Quad Drill Down (QDD) algorithm, as a solution to the range query problem on the Web using the properties of the quad-tree. A quad-tree is a tree whose nodes either are leaves or have four children, and it is one of the most commonly used data structures in spatial databases [57]. The quad-tree represents a recursive quaternary decomposition of space wherein at each level a subregion is divided into four equal sized subregions (quadrants). The properties of the quad-tree provide a natural framework for optimizing decomposition [70]. Hence, the quad-tree can give us an efficient way to divide the query region \( R \). We adopt the partitioning idea of the quad-tree, but we do not actually construct the tree. The main idea of QDD is to divide \( R \) into equal-sized quadrants, and then recursively divide quadrants further until each subregion is fully covered by a single \( k \)-NN search so that all objects in it are obtained.

Algorithm of QDD

Algorithm 5.1 describes QDD. First, a \( k \)-NN search is invoked with a point \( q \) which is the center of \( R \). Next, the \( k \)-NN circle \( C_r \) is obtained from the \( k \)-NN result. If \( C_r \) is larger than \( C_{R_q} \), it entirely covers \( R \), then all objects in \( R \) are retrieved. Finally, a pruning step is necessary to retrieve only the objects that are inside \( R \) – a trivial case. However, if \( C_r \) is smaller than or equal to \( C_{R_q} \), the query region is partitioned into four subregions by equally dividing the width and height of the region by two. The previous steps are repeated for every new subregion. The algorithm recursively partitions the query region into subregions until each subregion is covered by a \( k \)-NN circle. The pruning step eliminates those objects that are inside the \( k \)-NN circle but outside the subregion. An example of QDD is illustrated in Figure 5.3 (when \( k=3 \)), where twenty one \( k \)-NN calls are required to retrieve all the points in \( R \).
5.2 Range Queries using fixed values of \( k \): \( QDD \) and \( DCDT \) Algorithms

![Diagram of a Quad Drill Down range query](image)

(a) A \( QDD \) Range Query

(b) A Quad-tree representation of \( QDD \)

**Figure 5.3**: A Quad Drill Down range query
Algorithm 5.1 QDDRangeQuery($P_{ll}, P_{ur}$)

1: $q \leftarrow \text{getCenterOfRegion}(P_{ll}, P_{ur})$
2: $R_q \leftarrow \text{getHalfDiagonal}(q, P_{ll}, P_{ur})$
3: $\text{Knn}[] \leftarrow \text{add kNNSearch}(q)$
4: $r \leftarrow \text{getDistToK}^{th}\text{NN}(q, \text{Knn}[])$
5: if $r > R_q$ then
   6: $\text{result}[] \leftarrow \text{pruneResult}(\text{Knn}[], P_{ll}, P_{ur})$
6: else
5: 8: clear $\text{Knn}[]$
9: 10: $P_0 \leftarrow (q.x, P_{ll}.y)$
11: 12: $P_1 \leftarrow (P_{ur}.x, q.y)$
13: 14: $P_2 \leftarrow (P_{ll}.x, q.y)$
15: 16: $P_3 \leftarrow (q.x, P_{ur}.y)$
13: 14: $\text{Knn}[] \leftarrow \text{add QDDRangeQuery}(P_{ll}, q)$ // q1
14: 15: $\text{Knn}[] \leftarrow \text{add QDDRangeQuery}(P_0, P_1)$ // q4
15: 16: $\text{Knn}[] \leftarrow \text{add QDDRangeQuery}(P_2, P_3)$ // q2
16: 17: $\text{Knn}[] \leftarrow \text{add QDDRangeQuery}(q, P_{ur})$ // q3
17: $\text{result}[] \leftarrow \text{Knn}[]$
18: end if
19: return $\text{result}[]$

Statistical Analysis of QDD

Suppose that we have a perfectly balanced quad-tree. Each node corresponds to each region in which a range query is invoked at its center point. Let $D$ be the diagonal of the initial query region $R$ and $d_k$ be the minimum of the distances between any point in the data set and its $k$th nearest neighbor. The regions represented by nodes become smaller and smaller while going down the tree. Finally, when the diagonal of each region becomes
5.2 Range Queries using fixed values of $k$: QDD and DCDT Algorithms

less than or equal to $d_k$, $R$ can be completely covered by $k$-NN circles so that all the points in $R$ can be retrieved. When does the diagonal of every region become less than or equal to $d_k$? Let $L$ be the tree level when the diagonal of the region is less than or equal to $d_k$ (i.e., $L$ is the leaf level in the quad-tree). Then, $L = \log_4(D/d_k)$. Let $N_{knn}$ be the total number of nodes of the perfect quad tree. Then we have

$$N_{knn} \leq 4^0 + 4^1 + 4^2 + \ldots + 4^L \leq \sum_{i=0}^{L} 4^i \quad (5.1)$$

By solving (5.1), we get

$$N_{knn} \leq \sum_{i=0}^{L} 4^i = \sum_{i=0}^{L} 4^i = \frac{1}{3} \left(4 \left(\frac{D}{d_k}\right)^2 - 1\right) \quad (5.2)$$

The performance of the Naive approach is defined by $N_{knn}$ because the Naive needs to be perfectly balanced, i.e., each cell should be the same size.

However, the quad-tree representation of QDD does not need to be perfectly balanced, and the total number of nodes can be smaller than $N_{knn}$ (see the example in Figure 5.3.b). In QDD, each level of the quad tree has a different size of a query region, i.e., the size of its quadrant. If there are $j$ levels from level 0 to level $j - 1$, then $j$ different sizes of the query region are created. In order to analyze the performance of QDD, let us assume that the distribution of the datasets are known.

First, we do random range queries to get the data distribution of each size of range query. The data distribution for the given query region is achieved. We then calculate the probability for each level of the quad-tree, which is the probability that its quadrant cannot be covered by a single $k$-NN search. Non-zero probability means that the subregion should be further divided. Finally, we estimate the total number of nodes of the quad-tree. Figure 5.4 shows a data distribution of the number of points in a query region.
5.2 Range Queries using fixed values of $k$: $QDD$ and $DCDT$ Algorithms

Let $E$ be the expected value of the number of points in $C_{Rq}$, where $E \approx N \frac{\text{the area of } C_{Rq}}{A_{global}}$. Let $P_B$ be the probability that $C_{Rq}$ contains more than $k$ points, i.e., $P_B = P(E > k)$. Let $P_{Bi}$ be the probability that $C_{Rq}$ contains more than $k$ at the $i^{th}$ level of the tree. Then the number of nodes at the $i^{th}$ level is $(\prod_{j=0}^{i-1} P_{Bj}) \cdot 4^i$, and the number of nodes at the 0th level is obviously 1. Let $N'_{knn}$ denote the expected number of nodes in a quad-tree representation of $QDD$; it is quantified as:

$$N'_{knn} = 1 + P_{B0} \cdot 4^1 + \ldots + P_{B0}P_{B1} \ldots P_{BL-1} \cdot 4^L = 1 + \sum_{i=1}^{L} (\prod_{j=0}^{i-1} P_{Bj}) \cdot 4^i \quad (5.3)$$

Since $P_{Bi} \leq 1$ for every level $i$, $\prod_{j=0}^{i-1} P_{Bj}$ becomes smaller and smaller while the tree level increases, and finally approaches 0. Therefore, $N'_{knn} \leq N_{knn}$.

Table 5.2 shows the analysis of $QDD$ for the 4K synthetic uniform data set. The quad tree for the 4K data set has six levels from level 0 to 5. Each probability $P_{Bi}$ for level $i$ is calculated based on the data distribution of the given range query. We compare $N'_{knn}$ with $N_{knn}$ as well with the results of the $QDD$ algorithm. For example, when $k=15$, a quadrant at level 1 is not covered by a single $k$-NN search with a 99.96% probability, and one at level 2 is not covered by a $k$-NN search with a 6.35% probability. The perfect quad-tree
5.2 Range Queries using fixed values of $k$: $QDD$ and $DCDT$ Algorithms

with four levels has 85 nodes in total. Therefore, the total number of $k$-NN calls for the \textit{Naive} approach is $N_{knn} = 85$ while $N'_{knn} = 26$ for this case. The number of $k$-NN calls with $QDD$ for this example was 29 as presented in Section 5.5. Overall, $N'_{knn}$ had 62\% of average reduction compared to $N_{knn}$. We also observed that the maximum difference between $QDD$ and $N'_{knn}$ was 9\%. This statistical analysis describes the behavior of the $QDD$ algorithm.

$QDD$ discards the returned $k$-NN result when the $k$-NN circle does not cover the given query region and repeatedly calls additional four $k$-NN searches for the new four sub-regions. Thus, we can consider skipping some levels of the $QDD$ process to reduce unnecessary calls. If $R_c$ is the current query region of a $k$-NN search, and $P_s$ is the probability that the $k$-NN circle completely covers $R_c$, then $1 - P_s$ is the probability that the $k$-NN circle fails to cover the region. From statistical analysis based on the results of the previous $k$-NN searches, we estimate the probability $P_s$ that the $k$-NN search covers $R_c$. To skip the current call of $k$-NN search, the total expected number of $k$-NN searches for the current level and the next level must be greater than four calls; four $k$-NN searches are required if we skip the current level and directly go to four subregions for the next level. Hence we obtain the condition of $P_s$ by the following equation: $P_s + (1 - P_s) > 4$.

<table>
<thead>
<tr>
<th>Level</th>
<th>Query Size</th>
<th>$P_{Bi}(%)$</th>
<th>$k=5$</th>
<th>10</th>
<th>15</th>
<th>20</th>
<th>30</th>
<th>40</th>
<th>50</th>
</tr>
</thead>
<tbody>
<tr>
<td>Level 0</td>
<td>3%</td>
<td>$P_{B0}$</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>Level 1</td>
<td>1/4% 3%</td>
<td>$P_{B1}$</td>
<td>100</td>
<td>100</td>
<td>99.96</td>
<td>96.27</td>
<td>50.00</td>
<td>3.73</td>
<td>0</td>
</tr>
<tr>
<td>Level 2</td>
<td>1/4% 3%</td>
<td>$P_{B2}$</td>
<td>54.48</td>
<td>45.52</td>
<td>6.35</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Level 3</td>
<td>1/4% 3%</td>
<td>$P_{B3}$</td>
<td>36.94</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Level 4</td>
<td>1/4% 3%</td>
<td>$P_{B4}$</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>number of $k$-NN searches</td>
<td>$N_{knn}$</td>
<td>341</td>
<td>85</td>
<td>85</td>
<td>85</td>
<td>21</td>
<td>21</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>$N'_{knn}$</td>
<td>108</td>
<td>51</td>
<td>26</td>
<td>21</td>
<td>13</td>
<td>6</td>
<td>5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$QDD$</td>
<td>115</td>
<td>59</td>
<td>29</td>
<td>25</td>
<td>16</td>
<td>7</td>
<td>3</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\textbf{Table 5.2:} Statistical analysis of $QDD$: 4K uniform synthetic data set
5.2 Range Queries using fixed values of $k$: $QDD$ and $DCDT$ Algorithms

Therefore, $QDD$ can skip a $k$-NN search for $R_s$ iff $P_s < \frac{1}{4}$.

5.2.2 Dynamic Constrained Delaunay Triangulation ($DCDT$) Algorithm

In this section, we propose the Dynamic Constrained Delaunay Triangulation ($DCDT$) algorithm – a greedy and incremental approach to solve the range query on Web data using the Constrained Delaunay Triangulation ($CDT$)$^1$. $DCDT$ uses triangulations to divide the query range and keeps track of covered triangles by $k$-NN circles using the characteristics of constrained edges in $CDT$. $DCDT$ greedily selects the largest uncovered triangle for the next $k$-NN search while $QDD$ follows a pre-defined order. To overcome redundant $k$-NN search on the same area, $DCDT$ includes a propagation algorithm to cover the maximum possible area within a $k$-NN circle. Thus, no $k$-NN search will be wasted because a portion of a $k$-NN circle is always added to the covered area of the query range.

![Constrained edges](image1.png)  
![CDT](image2.png)

(a) Constrained edges  
(b) CDT

**Figure 5.5:** Example of Constrained Delaunay triangulation

$^1$The terms, $CDT$ and $DCDT$, are used for both the corresponding triangulation algorithms and the data structures that support these algorithms in this chapter.
5.2 Range Queries using fixed values of $k$: QDD and DCDT Algorithms

Given a planar graph $G$ with a set of $n$ vertices, $P$, in the plane, a set of edges, $E$, and a set of special edges, $C$, $CDT$ of $G$ is a triangulation of the vertices of $G$ that includes $C$ as part of the triangulation [18], i.e., all edges in $C$ appear as edges of the resulting triangulation. These edges are referred as to constrained edges, and they are not crossed (destroyed) by any other edges of triangulation. $CDT$ is also called as an obstacle triangulation or a generalized Delaunay Triangulation. Figure 5.5 illustrates a graph $G$ and the corresponding $CDT$. In this example, Figure 5.5 (a) shows a set of vertices, $P = \{P_1, P_2, P_3, P_4, P_5, P_6, P_7\}$, and a set of constrained edges, $C = \{C_1, C_2\}$. Figure 5.5 (b) shows the result of $CDT$ that includes the constrained edges of $C_1$ and $C_2$ as part of the triangulation.

We define the following data structures to maintain the covered and uncovered regions of $R$:

- **pList**: a set of all vertices of $G$; initially $\{P_{ll}, P_{lr}, P_{ur}, P_{ur}\}$
- **cEdges**: a set of all constrained edges of $G$; initially empty
- **tList**: a set of all uncovered triangles of $G$; initially empty

$pList$ and $cEdges$ are updated based on the current covered region by a $k$-NN search. $R$ is then triangulated (partitioned into subregions) using $CDT$ with the current $pList$ and $cEdges$. For every new triangulation, we obtain a new $tList$. $DCDT$ keeps track of covered triangles (subregions) and uncovered triangles; covered triangles are bounded by constrained edges (i.e., all three edges are constrained edges) and uncovered triangles are kept in $tList$, which are sorted in descending order by the area of the triangles. $DCDT$ chooses the largest uncovered triangle in $tList$ and calls $k$-NN search using the centroid, which always lies inside the triangle as opposed to the center of the triangle, as the query point. Our algorithm uses a heuristic approach that picks the largest triangle from the list of uncovered triangles. The algorithm terminates when no more uncovered triangles exist.
5.2 Range Queries using fixed values of $k$: QDD and DCDT Algorithms

Algorithm of DCDT

We describe the details of DCDT shown in Algorithm 5.2 and Algorithm 5.3. DCDT invokes the first $k$-NN search using the center point of the query region $R$ as the query point $q$. Figure 5.6 (a) shows an example of a 3-NN search. If the resulting $k$-NN circle $C_r$ completely covers $R$ ($r > R_q$), then we prune and return the result – a trivial case. If $C_r$ does not completely cover $R$, DCDT needs to use $C'_r$, which is a little smaller circle than $C_r$, for checking covered regions from this point for further $k$-NN searches. We have previously discussed the possibility that the query point $q$ has more than one $k^{th}$ nearest neighbor, and one of them is randomly returned. In that case, DCDT may not be able to retrieve all the points in $R$ if it uses $C_r$. Hence we define $C'_r = \epsilon \cdot C_r$, where $0 < \epsilon < 1$ ($\epsilon = 0.99$ in our algorithm).

DCDT creates an $n$-gon inscribed into $C'_r$. Choosing the value of $n$ depends on the tradeoff between computation time and the coverage of area; $n = 6$ (a hexagon) is used in our experiments as shown in Figure 5.6 (b). All vertices of the $n$-gon are added into $pList$. To mark the $n$-gon as a covered region, the $n$-gon is triangulated and the resulting edges are added as constrained edges into $cEdges$ (getConstrainedEdges() line 12 of Algorithm 5.2). The algorithm constructs a new triangulation with the current $pList$ and $cEdges$, then a newly created $tList$ is returned (constructDCDT() line 13 of Algorithm 5.2).

DCDT selects the largest uncovered triangle from $tList$ for the next $k$-NN search. With the new $C'_r$, DCDT updates $pList$ and $cEdges$ and creates a new triangulation. For example, if an edge lies within $C'_r$, then DCDT adds it into $cEdges$; on the other hand, if an edge intersects $C'_r$, then the partially covered edge is added into $cEdges$. Figure 5.7 shows an example of how to update and maintain $pList$, $cEdges$ and $tList$. $\triangle_1 (\triangle_{max})$ is the largest triangle in $tList$, therefore $\triangle_1$ is selected for the next $k$-NN search. DCDT uses the
5.2 Range Queries using fixed values of $k$: QDD and DCDT Algorithms

![Diagram](image)

(a) Query region $R$ and center point $q$
(b) Marking a hexagon as covered region

**Figure 5.6:** An example of first $k$-NN call in query region $R$

centroid of $\Delta_1$ as the query point $q$ for the $k$-NN search. $C'_r$ partially covers $\Delta_1$: vertices $v_5$ and $v_6$ are inside $C'_r$ but vertex $v_3$ is outside $C'_r$. $C'_r$ intersects $\Delta_1$ at $k_1$ along $v_3v_6$ and at $k_2$ along $v_3v_5$, hence, $k_1$ and $k_2$ are added into $pList$ ($getCoveredVertices()$ line 3 of Algorithm 5.3). Let $R_c$ be the covered area (polygon) of $\Delta_1$ by $C'_r$ (Figure 5.7 (c)). Then $R_c$ is triangulated and the resulting edges, $k_1k_2$, $k_1v_6$, $k_2v_5$, $v_5v_6$ and $k_1v_5$ are added into $cEdges$. The updated $pList$ and $cEdges$ are used for constructing a new triangulation of $R$ (Figure 5.7 (c)).

As shown in Figure 5.7 (c), the covered region of $\Delta_1$ can be a lot smaller than the coverage area of $C'_r$. In order to maximize the covered region by a $k$-NN search, DCDT propagates to (visits) the neighboring triangles of $\Delta_1$ (triangles that share an edge with $\Delta_1$). DCDT marks the covered regions starting from $\Delta_{max}$, and recursively visiting neighboring triangles ($findNeighbors()$ line 8-11 of Algorithm 5.3). Figure 5.8 shows an example of the propagation process. In Figure 5.8 (a), $\Delta_{max}$ is completely covered and its neighboring triangles $\Delta_{11}$, $\Delta_{12}$ and $\Delta_{13}$ (1-hop neighbors of $\Delta_{max}$) are partially covered in Figure 5.8 (b), (c) and (d), respectively. In Figure 5.8 (e), the neighboring triangles of $\Delta_{max}$’s 1-hop neighbors, i.e., 2-hop neighbors of $\Delta_{max}$, are partially covered. Finally, based on the
Algorithm 5.2 DCDTrangeQuery($P_{ll}, P_{ur}, \epsilon$)

1: $pList() \leftarrow \{P_{ll}, P_{rl}, P_{lr}, P_{ur}\}$; \hspace{1em} $cEdges() \leftarrow \{\}$; \hspace{1em} $tList() \leftarrow \{\}$;
2: $Knn() \leftarrow \{\}$ \hspace{1em} // all objects retrieved by $k$-NN search
3: $q \leftarrow$ getCenterOfRegion($P_{ll}, P_{ur}$)
4: $Knn() \leftarrow$ add kNNSearch($q$)
5: $r \leftarrow$ getDistTo$K$thNN($q, Knn()$)
6: $C_r \leftarrow$ circle centered at $q$ with radius $r$
7: $R_q \leftarrow$ getHalfDiagonal($q, P_{ll}, P_{ur}$)
8: if $r \leq R_q$; $C_r$ does not cover the given region $R$ then
9: $C'_r \leftarrow$ circle centered at $q$ with radius $\epsilon \cdot r$
10: $N_g \leftarrow$ create an n-gon inscribing $C'_r$ with $q$ as the center
11: $pList() \leftarrow$ add {all vertices of $N_g$}
12: $cEdges() \leftarrow$ getConstrainedEdges({all vertices of $N_g$})
13: $tList() \leftarrow$ constructDCDT($pList(), cEdges()$)
14: while $tList$ is not empty do
15: \hspace{1em} mark all triangles in $tList$ to unvisited
16: $\Delta_{max} \leftarrow$ getMaxTriangle()
17: $q \leftarrow$ getCentroid($\Delta_{max}$)
18: $Knn() \leftarrow$ add kNNSearch($q$)
19: \hspace{1em} $r \leftarrow$ getDistTo$K$thNN($q, Knn()$)
20: \hspace{1em} $C'_r \leftarrow$ circle centered at $q$ with radius $\epsilon \cdot r$
21: \hspace{1em} checkCoverRegion($\Delta_{max}, C'_r, pList(), cEdges()$)
22: $tList() \leftarrow$ constructDCDT($pList(), cEdges()$)
23: \end while
24: \end if
25: $result() \leftarrow$ pruneResult($Knn(), P_{ll}, P_{ur}$)
26: return $result()$
5.2 Range Queries using fixed values of $k$: QDD and DCDT Algorithms

![Diagram](image1)

(a) step1  (b) step2  (c) step3

**Figure 5.7:** Example of pList, cEdges and tList: covering $A_{max}$

---

**Algorithm 5.3** checkCoverRegion($\Delta$, $C'_r$, pList[], cEdges[])

1: if $\Delta$ is marked unvisited then
2: mark $\Delta$ as visited
3: $localPList[] \leftarrow$ getCoveredVertices($\Delta$, $C'_r$)
4: $localCEdges[] \leftarrow$ getConstrainedEdges($localPList[]$)
5: if $localCEdges$ is not empty then
6: $pList[] \leftarrow$ add $localPList[]$
7: $cEdges[] \leftarrow$ add $localCEdges[]$
8: $neighbors[] \leftarrow$ findNeighbors($\Delta$)
9: for every $\Delta_i$ in $neighbors[]$, $i = 1, 2, 3$ do
10: checkCoverRegion($\Delta_i$, $C'_r$, pList[], cEdges[])
11: end for
12: end if
13: end if

---

103
5.2 Range Queries using fixed values of $k$: QDD and DCDT Algorithms

returned result of $\text{checkCoverRegion}()$ shown in Figure 5.8 (e), DCDT constructs a new triangulation as shown in Figure 5.8 (f). Algorithm 5.3 describes the propagation process.

![Diagram](image)

**Figure 5.8:** Example of checkCoverRegion()

**Cover Regions: checkCoverRegion()**

In this section we define the cases of covered region $R_c$ and discuss the details of Algorithm 5.3. Let $C_r$ be the returned $k$-NN circle centered by $q$ and $\triangle T$ be the given triangle that we consider covering using $C_r$. Let $a$, $b$ and $c$ be the three vertices of $\triangle T$. The following cases are defined for covered region $R_c$, and the corresponding examples are illustrated in Figure 5.9.

Case 1: All three vertices, $a$, $b$ and $c$, are inside $C_r$ so that $\triangle T$ is completely covered by a
5.2 Range Queries using fixed values of $k$: QDD and DCDT Algorithms

Figure 5.9: Examples of the cases for covered region $R_c$
5.2 Range Queries using fixed values of \( k \): QDD and DCDT Algorithms

\( k \)-NN search: \( R_c \) is \( \triangle_{abc} \).

Case 2: Two vertices (say \( a \) and \( b \)) are inside \( C_r \) but the third vertex (say \( c \)) is outside \( C_r \).

Then \( C_r \) has one intersection (say \( b' \)) with \( bc \) and one intersection (say \( a' \)) with \( ca \): \( R_c \) is the polygon with the edges, \( \overline{ab}, \overline{a'a}, \overline{bb'} \) and \( \overline{a'b'} \).

Case 3: One vertex (say \( a \)) is inside \( C_r \) but two other vertices (say \( b \) and \( c \)) are outside \( C_r \);

Case 3.1: \( C_r \) has one intersection (say \( b' \)) with \( ab \) and one intersection (say \( c' \)) with \( ca \): \( R_c \) is \( \triangle_{ab}c' \).

Case 3.2: \( C_r \) has one intersection (say \( b' \)) with \( ab \), one intersection (say \( c' \)) with \( ca \), and either one intersection (say \( k_1 \)) or two intersections (say \( k_1 \) and \( k_2 \)) with \( bc \): \( R_c \) is the polygon with the edges, \( \overline{ab'}, \overline{b'k_1}, \overline{k_1k_2}, \overline{k_2c'} \) and \( \overline{c'a} \).

Case 4: All three vertices are outside \( C_r \);

Case 4.1: The center of \( C_r \) is inside \( \triangle_T \) and \( C_r \) has either no intersection or only one intersection with all edges of \( \triangle_T \): create an \( n \)-gon inscribed into \( C_r \) and then \( R_c \) is \( n \)-gon.

Case 4.2: \( C_r \) has two intersections with at least two edges of \( \triangle_T \): Let these intersection points be \( k_1, k_2, \ldots, k_i \) where \( 4 \leq i \leq 6 \). \( R_c \) is the polygon with the edges, \( k_1k_2, k_2k_3, \ldots, k_ik_1 \).

Case 4.3: \( C_r \) has two intersections with one edge (say \( ab \)) of \( \triangle_T \). Let these two intersections be \( k_1 \) and \( k_2 \) and let \( m \) be the center point of \( k_1k_2 \):

Case 4.3a: If the center of \( C_r \) is inside \( \triangle_T \), then add \( k_1k_2 \), then create an \( n \)-gon inscribed into \( C_r \) and find the intersections of the \( n \)-gon and \( ab \). Let \( S \) be a set of points that includes: 1) the points of \( n \)-gon that are inside \( \triangle_T \), 2) the intersection points of the \( n \)-gon and \( ab \), \( S = \{k_1, k_2, \ldots, k_i\} \). \( R_c \) is the polygon with edges in \( S \).

Case 4.3b: If the center of \( C_r \) is outside \( \triangle_T \), then find an intersection point of \( C_r \) and the line that is perpendicular to \( k_1k_2 \) and passes the point \( m \). Let this point be \( m' \). Then \( R_c \) is \( \triangle_{k_1k_2m'} \).

After we define \( R_c \), \( T_c \) is obtained by triangulating \( R_c \). Then all vertices of \( R_c \) are
5.2 Range Queries using fixed values of $k$: $QDD$ and $DCDT$ Algorithms

(a) 1st $k$-NN call (12%)  
(b) 5th $k$-NN call (52%)  
(c) 9th $k$-NN call (92%)

Figure 5.10: Example of $DCDT$ w/k=15 (coverage rate)

added into $pList$, and all edges of $T_c$ are added into $cEdges$. The updated $pList$ and $cEdges$ are used for creating a new triangulation.

Notice that $DCDT$ covers a certain portion of $R$ at every step, and therefore the covered region grows as the number of $k$-NN searches increases. On the contrary, $QDD$ discards the returned result from a $k$-NN call when it does not cover the whole subregion, which results in re-visiting the subdivisions of the same region with 4 additional $k$-NN calls (see line 5-16 in Algorithm 5.1). Also note that the cost of triangulation is negligible as compared to $k$-NN search because triangulation is performed in memory and incrementally.

$DCDT$ repeats the processes until no more uncovered regions exist. The resulted $k$-NN circle $C_r$ can either completely cover $R$ or partially cover $R$. The example in Figure 5.10 (a) shows the first step of $DCDT$. The shaded triangles are covered-subregions and all the edges of these triangles are now considered as constrained edges. Figure 5.10 (b) and (c) illustrate examples of $DCDT$ result after the 5th $k$-NN search and the 9th $k$-NN search, respectively.
5.3 Range Queries using flexible values of $k$: DB Algorithm

Some Web applications can support a $k$-NN search with any $k$ value. By carefully determining the $k$ value, a range query can be evaluated with a single $k$-NN search if all the objects in the given query region can be found from the results of the very first $k$-NN search.

The main idea of the density-based range query is to find a good approximation of the $k$ value using some statistical knowledge of the Web data, i.e., the density of the data sets. Our estimation method for $k$ is based on the density of the data set. Assuming that the density information of Web data is available, let $N$ be the total number of objects in a web source and $A_{\text{global}}$ be the size of the minimum bounding rectangle (MBR) that includes all those objects. Then, two estimation methods are considered: global and local estimation.

The global estimation method is as follows: The global density is $\frac{N}{A_{\text{global}}}$, and $R_q$ is the half diagonal of $R$ as described in Figure 5.2. We then estimate the number of objects ($k$) that are inside $C_{R_q}$ with the parameters, $A_{\text{global}}$, $N$, and $R_q$:

$$k = \pi R_q^2 \cdot \frac{N}{A_{\text{global}}}$$

$k$ is used to call the first $k$-NN search to the Web data. From the $k$-NN result, $C_r$ ($k$-NN circle) is expected to match $C_{R_q}$. If $C_r \leq C_{R_q}$, then we need extra $k$-NN searches. Thus, we need to reestimate the $k$ value using the local estimation.

The local estimation method is as follows: Let $A_{\text{local}}$ be the area of the current $C_r$, Area($C_r$)=$\pi r^2$ and $k$ be the value used by the previous $k$-NN search. The local density is $\frac{k}{A_{\text{local}}}$. Then the new estimation of $k$ is:

$$k' = \pi R_q^2 \cdot \frac{k}{A_{\text{local}}}$$

$k'$ is used to call the next $k$-NN search to the Web data.
5.3 Range Queries using flexible values of $k$: DB Algorithm

5.3.1 Algorithm of DB

The DB algorithm works as follows. First, we estimate the initial $k$ using the global density of the data set. The center point of the query region $R$ is the query point $q$. If $C_r$ is not larger than $C_{R_q}$, the value of $k$ is under-estimated because $C_r$ cannot cover the entire $R$. Then, it is necessary to increase the $k$ value. A new $k'$ value is then calculated using the local density. This local estimation of $k'$ continues until $C_r$ becomes larger than $C_{R_q}$. If $C_r$ is larger than $C_{R_q}$, the value of $k$ is over-estimated so that the result of $k$-NN could include all the objects inside $C_{R_q}$. Finally, pruning is required to eliminate the objects outside $R$. Algorithm 5.4 describes the DB range query.

The range query execution is broken down into four basic steps: 1) Client sends a request to the Web server with the query point $q$, 2) Server executes the $k$-NN search using $q$, 3) Server returns the result of the $k$-NN search to client, 4) Client prunes the result (client may invoke additional requests to server). Let $C_q$, $C_{knn}$, $C_{res}$, and $C_{pru}$ represent the cost associated with each of these four steps. Let $C_n$ be the total cost of the range query, where $n$ is the total number of $k$-NN searches required to complete a range query. Then, $C_n$ is defined as follows: $C_n = C_{pru(n)} + \sum_{i=1}^{n} [C_{q(i)} + C_{knn(i)} + C_{res(i)}].$

Suppose that the first estimated $k$ value is not sufficient to cover a query region $R$ and one more $k$-NN search is required. Then the query cost is $C_2 = C_{pru(2)} + (C_{q(1)} + C_{knn(1)} + C_{res(1)}) + (C_{q(2)} + C_{knn(2)} + C_{res(2)})$. Let $a_1$, $a_2$, $a_3$, and $a_4$ be some constant factors for $C_q$, $C_{knn}$, $C_{res}$, and $C_{pru}$, respectively. Supposing that $k'$ is the second estimated $k$, then we rewrite the cost $C_2 = a_1 + 2a_2 \log N + a_2 (k + k') + a_3 (k + k') + a_4 (k + k')$.

On the other hand, if we double the $k$ value for the first $k$-NN search, the query cost is $C_1 = a_1 + a_2 \log N + a_2 (k + k) + a_3 (k + k) + a_4 (k + k)$. Since $k < k'$, we have $C_1 < C_2$. This proves that the number of $k$-NN calls affects the query cost more than the value of $k$.

In some cases, the number of points inside $C_r$ can be greater than the initial $k$ value,
5.3 Range Queries using flexible values of $k$: DB Algorithm

Algorithm 5.4 DB rangeQuery($P_{ll}$, $P_{ur}$, $A_{global}$, $N$)

1: $\epsilon \leftarrow$ a constant factor
2: $P_{c}.x \leftarrow (P_{ll}.x + P_{ur}.x)/2$
3: $P_{c}.y \leftarrow (P_{ll}.y + P_{ur}.y)/2$
4: $R_q \leftarrow \sqrt{(P_{ur}.x - P_{c}.x)^2 + (P_{ur}.y - P_{c}.y)^2}$
5: $q \leftarrow P_c$
6: $k_{init} \leftarrow$ estKvalue($R_q$, $A_{global}$, $N$)
7: $Knn[] \leftarrow$ kNNSearch($q$, $\epsilon \cdot k_{init}$)
8: $r \leftarrow$ maxDistkNN($q$, $Knn[]$)
9: $k \leftarrow k_{init}$
10: while $r \leq R_q$ do
11: \hspace{1em} clear $Knn[]$
12: \hspace{1em} $A_{local} \leftarrow \pi \cdot r^2$
13: \hspace{1em} $k \leftarrow$ estKvalue($R_q$, $A_{local}$, $k$)
14: \hspace{1em} $Knn[] \leftarrow$ kNNSearch($q$, $\epsilon \cdot k$)
15: \hspace{1em} $r \leftarrow$ maxDistkNN($q$, $Knn[]$)
16: end while
17: $result[] \leftarrow$ pruneResult($Knn[]$, $P_{ll}$, $P_{ur}$)
18: return $result[]$

resulting in extra $k$-NN searches. For those cases, the range query could be evaluated in a single $k$-NN search if the initial $k$ value were over-estimated well. Consequently, we introduce the over-estimation constant, $\epsilon$, for a loosely bounded $k$ value. To obtain a loosely bounded $k$ value, the $k$ value is multiplied by $\epsilon$. The $\epsilon$ value can be achieved from empirical experiments and statistics based upon the density and distribution of data set. In our experiments in Section 5.5, we demonstrate how the $\epsilon$ value affects the performance of the DB algorithm.
5.3 Range Queries using flexible values of $k$: DB Algorithm

5.3.2 Statistical Analysis of DB

The proposed algorithms assume no knowledge (or little) about the data set where range queries are performed because this information is generally unknown in reality. However, the data distribution in a query region does affect the performance of the algorithms. One can devise the best solution with a complete knowledge of the data set. To evaluate the impact of data distribution on the algorithms, this section performs a statistical analysis. First, we generate a number of data points assuming two typical distributions of their locations, uniform and exponential. Then, while varying the size of range queries, we analyze how many points will be distributed in a query region and how it will impact the performance of the algorithms.

We created various sized synthetic data sets (1K, 2K, 4K, 8K, 16K points) using both uniform and exponential distribution of their locations. The size of query region varied such that $C_{R_q}$ of the given query region varied: 3%, 5%, and 10% of the entire region of data set. For each experiment, 100K range queries were issued on randomly generated query points. Then, the average number of points in a query region was measured.

![Data distribution of the number of points in a query region](image)

**Figure 5.11:** Data distribution of the number of points in a query region
5.4 Hybrid (HB) Algorithm

Figure 5.11 shows the distributions of the number of points in a query region with the query sizes of 3%, 5%, and 10% for the 4K synthetic data sets. The x-axis represents the number of points inside $R$, and the y-axis represents the probability of each number of points. The distribution of the uniform data set is normal while that of the exponential data set shows a skewed distribution. For example, the 3% query size in the uniform data set retrieved 120 points on average and the maximum number of points was 156.

Our global estimation method used in the $DB$ algorithm decides the initial $k$ value using the global estimation (i.e., $k = 3 \cdot 4000/100 = 120$), which matches the result from the statistical analysis well. Then, if $DB$ uses $\epsilon = 1.3$ (i.e., $k = 120 \times 1.3 = 156$), it can retrieve all the points in $R$ in a single $k$-NN search. Even though the exponential data set resulted in a skewed distribution, two or three $k$-NN searches can be enough to retrieve all points in $R$. For example, Approximately 75% of the range queries resulted in less than or equal to the estimated value of $k$ in $DB$. In our experimental result of $DB$ in Section 5.5, the value of $k$ was well over-estimated in most cases with the exponential data set. This proves that $DB$ works well regardless of the distribution of data set. This was also confirmed in the implementation of $DB$ as presented in Section 5.5.

5.4 Hybrid (HB) Algorithm

Some Web applications support $k$-NN queries with bounded values of $k$. When the range of $k$ values is bounded, not only a good estimation of the $k$ value must be considered but also a good placement of query point $q$. $HB$ combines the basic approaches of $QDD$ and $DB$ algorithms that are discussed in Section 5.2 and Section 5.3, respectively.

$HB$ first estimates the value of $k$ based on the global density. If additional $k$-NN search calls are required, the new $k$ value is calculated based on the local density. The estimation of the $k$ value continues until either the $k$-NN circle covers the query region $R$
5.4 Hybrid (HB) Algorithm

**Algorithm 5.5** HB rangeQuery($P_{ll}$, $P_{ur}$, $A_{global}$, $N$)

1: $\varepsilon \gets$ a constant factor; $k_{max} \gets$ a system factor
2: $q \gets$ getCenterOfRegion($P_{ll}$, $P_{ur}$)
3: $R_q \gets$ getHalfDiagonal($q$, $P_{ll}$, $P_{ur}$)
4: $k_{init} \gets$ estKvalue($R_q$, $A_{global}$, $N$, $\varepsilon$)
5: $Knn[] \gets$ kNNSearch($q$, $k_{init}$)
6: $r \gets$ maxDistkNN($q$, $Knn[]$); $k \gets k_{init}$
7: while $r < R_q$ and $k < k_{max}$ do
8:   clear $Knn[]$
9:   $A_{local} \gets \pi \cdot r^2$
10:  $k \gets$ estKvalue($R_q$, $A_{local}$, $k$, $\varepsilon$)
11:  $Knn[] \gets$ kNNSearch($q$, $k$)
12:  $r \gets$ maxDistkNN($q$, $Knn[]$)
13: end while
14: if $r < R_q$ then
15:   clear $Knn[]$
16:   $P_0 \gets (q.x, P_{ll}.y)$
17:   $P_1 \gets (P_{ur}.x, q.y)$
18:   $P_2 \gets (P_{ll}.x, q.y)$
19:   $P_3 \gets (q.x, P_{ur}.y)$
20:   $Knn[] \gets$ add HB rangeQuery($P_{ll}$, $q$, $A_{local}$, $k$)
21:   $Knn[] \gets$ add HB rangeQuery($P_0$, $P_1$, $A_{local}$, $k$)
22:   $Knn[] \gets$ add HB rangeQuery($P_2$, $P_3$, $A_{local}$, $k$)
23: end if
24: return $result[] \gets Knn[]$
25: else
26: return $result[] \gets$ prunResult($Knn[]$, $P_{ll}$, $P_{ur}$)
27: end if
5.5 Experiments

or the $k$ value reaches the maximum value allowed. If the $k$ value reaches the maximum before $R$ is covered, $R$ is partitioned into four equal-sized subregions. The algorithm keeps recursively partitioning the query region into subregions until every subregion is covered by its $k$-NN circle. The implementation of HB is straightforward from DB and QDD, and the details of the HB algorithm is described in Algorithm 5.5.

5.5 Experiments

In this section, we evaluate QDD and DCDT using both synthetic and real GIS data sets. Our synthetic data sets use both uniform (random) and skewed (exponential) distributions. For the uniform data sets, $(x, y)$ locations are distributed uniformly and independently between 0 and 1. The $(x, y)$ locations for the skewed data sets are independently drawn from an exponential distribution with mean 0.3 and standard deviation 0.3. The number of points in each data set varies: 1K, 2K, 4K, and 8K points. Our real data set is from the U.S. Geological Survey in 2001: Geochemistry of consolidated sediments in Colorado in the U.S. [67]. The data set contains 5,410 objects (points).

Our performance metric is the number of $k$-NN calls required for a given range query. We performed range queries using the proposed algorithms with various data sets and measured the average number of $k$-NN searches to support a range query while varying the size of range queries and the distribution of the data sets. In our experiments, we varied a range query size between 1% and 10% of the entire region of the data set. Different $k$ values between 5 and 50 were used for the QDD algorithm, and the DB algorithm used varying $\epsilon$ values from 1.0 to 2.0. Each size of range queries was conducted for 100 trials and the average values were reported. We present only the most illustrative subset of our experimental results due to space limitation. Similar qualitative and quantitative trends were observed in all other experiments.
5.5 Experiments

5.5.1 Results of QDD and DCDT

In this section, we present the performance of the QDD and DCDT algorithms.

First, we compared the performance of QDD and DCDT with the theoretical minimum, i.e., the necessary condition (N.C.) \( \lceil \frac{n}{k} \rceil \), where \( n \) is the number of points in \( R \).

Figure 5.12 (a) and (b) show the comparisons of QDD, DCDT and N.C. with uniformly distributed and exponentially distributed 4k synthetic data sets, respectively. For both DCDT and QDD, the number of \( k \)-NN calls rapidly decreased as the value of \( k \) increased in the range 5-15 and 5-10 for the uniformly and exponentially distributed data, respectively. Then, it slowly decreased as the value of \( k \) became larger, approaching to those of N.C. when \( k \) is over 35. The results for the real data set have similar trends to those of the exponentially distributed synthetic data set (Figure 5.13). Note that \( k \) is determined not by the client but by the Web interface, and a typical \( k \) value in real applications ranges between 5 and 20 [14]. In both synthetic and real data sets, DCDT needed a significantly smaller number of \( k \)-NN calls compared to QDD. For example, with the exponential data set (4K), DCDT showed 55.3% of average reduction in the required number of \( k \)-NN searches compared to QDD. As discussed in Section 5.2, DCDT covers a portion of the query region with every \( k \)-NN search while QDD revisits the same subregion when a \( k \)-NN search fails to cover the entire subregion. DCDT still required more \( k \)-NN calls than N.C. On the average, DCDT required 1.67 times more \( k \)-NN calls than N.C. However, the gap became very small when \( k \) was greater than 15.

Next, we conducted range queries with different sizes of query regions: 3%, 5%, and 10% of the entire region of data set. Table 5.3 shows the average percentage reduction in the number of \( k \)-NN calls between DCDT and QDD for 4K uniformly and exponentially distributed synthetic data sets. On the average, DCDT resulted in 50.4% and 58.2% of reduction over QDD for the uniformly and exponentially distributed data set, respectively.
5.5 Experiments

The results show that the smaller the value of $k$ is, the greater the reduction rate.

In Figure 5.14, we plotted the average coverage rate versus the number of $k$-NN calls required for DCDT on a 4K synthetic uniformly distributed data set. Figure 5.14 (a) shows the average coverage rate of the DCDT algorithm while varying the value of $k$. For example, when $k=7$, 50% of the query range can be covered by the first 8 $k$-NN calls while the entire range is covered by 40 $k$-NN calls. The same range query required 24 calls for
Table 5.3: The average percentage reduction in the number of $k$-NN calls (%)

![Graph](image)

(a) 3% range query with 4K synthetic data set  
(b) 4K synthetic data set with $k=10$

**5.5.2 Results of the DB Algorithm**

With the statistical knowledge of the data set, $DB$ can decide the best $k$ value to minimize the number of $k$-NN calls. As a result, $DB$ completed any range query in one or two $k$-NN calls in all experiments with both synthetic and real data sets. The impact of $\epsilon$
5.5 Experiments

Figure 5.15: Percentage of $k$-NN calls of $DB$

Figure 5.16: Accuracy of $DB$

was quantified by measuring the required number of $k$-NN calls while varying $\epsilon$. As $\epsilon$ increased from 1.0 to 2.0, more range queries were completed in a single $k$-NN call. Figure 5.15 (a) shows what percentage of range queries were evaluated in one $k$-NN call (or in two) while varying $\epsilon$. For example, when $\epsilon=1.0$, 57% of range queries were covered by one $k$-NN search while 43% were by two $k$-NN searches. When $\epsilon \geq 1.2$, all range
5.6 Summary

queries were completed in one $k$-NN call. Overall, the exponential data set required a higher number of $k$-NN calls than the uniform data. The real data set showed a similar trend as the exponential data set.

$DB$ performed all range queries in one or two $k$-NN calls with over-estimated values of $k$. Because $DB$ uses a loosely bounded $k$ value, the obtained $k$-NN circle $C_r$ is larger than $C_{R_q}$ in most cases. To quantify how accurately $DB$ estimates $C_r$ with regard to $C_{R_q}$ ($C_r/C_{R_q}$ or $r/R_q$) using $\epsilon$, we measured the ratio of $r$ to $R_q$ while varying $\epsilon$ (Figure 5.16). The ratio was approximately 1.26 and 1.98 for the uniform and exponential data sets, respectively, when $\epsilon = 1.5$ while the ratio was 1.48 for the uniform data set and 2.34 for the exponential data set when $\epsilon = 2.0$. The ratio for the real data set are presented in Figure 5.16 (b): 1.51 with $\epsilon = 1.5$ and 1.77 with $\epsilon = 2.0$. For both the synthetic and real data sets, the ratio $r$ to $R_q$ decreased as the size of the range query increased.

5.6 Summary

In this chapter, we introduced the problem of evaluating spatial range queries on Web data by using only $k$-Nearest Neighbor searches. The problem of finding a minimum number of $k$-NN searches to cover the query range appears to be hard even if the locations of the objects (points) are known.

The Quad Drill Down ($QDD$) and the Dynamic Constrained Delaunay Triangulation ($DCDT$) algorithms were proposed for the most restricted $k$-NN interface, i.e., fixed values of $k$. We showed that $QDD$ and $DCDT$ covered the entire range (completeness) while providing reasonable performance (efficiency). The efficiencies of these two algorithms were compared each other as well as with the necessary condition. $DCDT$ provides better performance than $QDD$. The Density-Based ($DB$) algorithm was proposed assuming data distribution knowledge and more flexible $k$-NN interface, i.e., a value of $k$ is a user input.
5.6 Summary

DB used available data density information, resulting in only one or two $k$-NN searches in all our experiments. Finally, we proposed the Hybrid ($HB$) algorithm that combines the $QDD$ and $DB$ algorithms and provides the possibility of customization of proposed algorithms for real world applications.
Chapter 6

Conclusions

This chapter summarizes the main results of this dissertation and presents possible future research problems.

6.1 Summary of Results

This dissertation presented issues for designing and implementing online query processing algorithms in Geographic Information Systems (GIS). The large size of the data sets and the complexity of spatial query processing make a traditional spatial query processing unsuitable for online query processing. Therefore, we introduced a new interactive approach for online spatial queries, which enables users to quickly see query results (i.e., an approximate answer and the current actual result) as soon as the query is executed and to control (i.e., stop or update) the query processing as the query runs. This new approach allows for interactive spatial joins for both vector and raster data. Moreover, the restrictive Web interfaces of spatial queries motivated us to conceptualize the Web data retrieval problem into a more general problem of supporting rectangular shaped range queries on Web data. We then proposed methods to efficiently evaluate range queries using only $k$-Nearest Neighbor ($k$-NN) queries. This dissertation enables interactivity, flexibility and
efficiency for Web-based GIS, resulting in faster and more effective decision support.

First, we designed and implemented a family of *Incremental Refining Spatial Join* (IRSJ) algorithms to support interactive spatial joins over vector data. The IRSJ algorithms return incrementally refined running estimates of the final answer with bounded confidence intervals. In addition, the IRSJ algorithms display the partial results for the data sets joined so far allowing users to gain more insight into the query results. The algorithms significantly reduced the query response time relative to a full R-tree join, but does require users to be willing to accept an estimate within a bounded confidence interval. The algorithms achieved reduced time by providing query estimates obtained from a subset of the full data join directed by sampling techniques. We investigated and evaluated different sampling methodologies: simple random sampling, stratified random sampling, and cluster sampling. Through extensive experimental evaluations using R-trees, R*-trees and STR packed R-trees, we showed that the proposed algorithms achieved the expected performance regardless of the indexing data structure. In addition, the time required to produce reasonably accurate online estimates, i.e., a 5% confidence interval at a 90% probability (confidence level), was up to an order of magnitude smaller than the time required for the exact answer using a full R-tree join.

The second problem in this dissertation studied join approximation for raster data. The traditional map overlay method for raster data joins does not provide an idea of how the final result will look until the join is completed. Hence, to enable an interactive data exploration over raster data, it is essential to allow a user to get a fast estimation, i.e., a “big picture” visualization, of the join result. We proposed two statistical approaches for estimating spatial join results on quad-tree indexed raster data: *Probabilistic Join* (PJ) and *Incremental Stratified Sampling Join* (ISSJ). PJ allows users to obtain an approximate visualization of the answer in two orders of magnitude faster time than the time required for an exact answer, while ISSJ obtains an order of magnitude improvement of
6.2 Future Research Work

query response time over the full quad-tree join. The \textit{ISSJ} algorithm provides running confidence intervals for estimates whereas the \textit{PJ} algorithm does not.

Finally, this dissertation presented methods for retrieving georeferenced data from Web applications that only support \textit{k-NN} queries. Based on the classification of the \textit{k-NN} interfaces on the Web, a set of range query algorithms were proposed to achieve the completeness and efficiency requirements of spatial range queries. Through extensive experimental performance evaluations, we provided practical possibilities of the proposed algorithms for real world applications.

6.2 Future Research Work

The following problems have arisen as a result of this dissertation:

1. Extending the incremental sampling techniques to multiway spatial joins. We conjecture that sampling from all joined data sets would obtain better data randomization and thus result in a tighter confidence interval with the same query processing time.

2. In a traditional query processing, multiple query plans for satisfying a query are examined and a good query plan is identified a priori. This may or not be the best strategy because there are many ways of obtaining query plans. Dynamic spatial joins do not decide a query plan a priori but optimize a query while the query is processing. Extending \textit{Incremental Refining Spatial Joins (IRSJ)} for supporting dynamic spatial joins, which continuously adjust the query processing in accordance with the statistical results of a subset of data sets. This adaptive approach might maximize the amount of confidence interval convergence at each update of estimates. The questions to be considered are what factors affect query processing, how to determine join orders and when join orders can be changed.
6.2 Future Research Work

3. Georeferenced data sets can be correlated based on their spatial properties. If correlation coefficients among joined data sets are involved in the calculation of join probabilities, we might obtain better approximations of raster spatial joins using Probabilistic Join (PJ).

4. Our study of Probabilistic Join (PJ) over raster data showed possibility of extending PJ to spatial joins on vector data sets. The result of 2-dimensional join probability promises a good estimation of overlapping area between two polygons. Approximating the size of overlapping area of polygons at the filtering step of spatial joins would greatly reduce the number of I/Os.

5. The computational complexity of range queries using $k$-Nearest Neighbor searches is unknown for the case where we have no control on the value of $k$. Even though the empirical results of the Dynamic Constrained Delaunay Triangulation (DCDT) algorithm are quite reasonable, it would be interesting to improve DCDT to achieve a bounded complexity. Instead of propagating to the neighbor triangle regions, which results in many computational comparisons, always using an n-gon as a covered region might improve the performance of DCDT.
Bibliography


BIBLIOGRAPHY


[22] M. Dickerson, R. Drysdale, and J. Sack. Simple algorithms for enumerating inter-
point distances and finding k nearest neighbors. *International Journal of Computa-


graphic Information Analysis. *University of California, Santa Barbara, USA*, 1989.


augmented frequent value statistics. In *Proceedings of International Conf. on Data

[31] P. J. Haas, J. F. Naughton, and A. N. Swami. On the relative cost of sampling for

sampling. In *Proceedings of International Conf. on Database and Expert Systems

[33] P. J. Hass. Large-sample and deterministic confidence intervals for online aggrega-
tion. In *Proceedings of International Conf. Scientific and Statistical Databases

[34] P. J. Hass and J. M. Hellerstein. Ripple joins for online aggregation. In *Proceedings


ONLINE QUERY PROCESSING IN GEOGRAPHIC INFORMATION SYSTEMS

An Abstract of a Dissertation
Presented to
the Faculty of Engineering and Computer Science
University of Denver

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy

by
Wan D. Bae
November 2007

Advisor: Prof. Scott T. Leutenegger
Co-advisor: Prof. Seon Ho Kim
Abstract

Geographic Information Systems (GIS) have been widely used in many applications for exploring large georeferenced data. With the recent advances in Internet technologies, a large volume of GIS data is available on the Web. As a result, an increasing number of emerging applications began to provide tools for accessing these data. However, exploring georeferenced data can be very time-consuming due to the large size of the data. Current query processing techniques deal with spatial queries in a blocking manner and hence, may not be directly applicable for data analysis in GIS. This problem is more pronounced in decision support queries where query response time is a critical issue. Since these queries are mostly used to get a “big picture” of data sets and their relationships, a more useful and effective approach is to provide approximate query answers quickly and in an interactive manner. In this way, users can obtain an idea of how the query result would look like and can therefore stop/modify the query accordingly. Another issue related to online query processing is the limited access to Web data by certain types of queries due to restrictive Web interfaces. This problem hinders online data retrieval, which necessitates for methods that utilize certain query types to provide solutions to non-supported queries.

This dissertation studies technical, practical and theoretical issues raised in designing and implementing online query processing in GIS and provides
techniques for interactive spatial joins and efficient Web data retrieval. We present a family of interactive spatial join algorithms that report incrementally refined running estimates for aggregate queries over vector data, while simultaneously displaying the actual query result tuples of the data sets sampled so far. We also present a new framework for raster data joins that allows users to get approximate answers in near instantaneous time for more interactive data exploration. Both solutions to online query processing for vector and raster data provide orders of magnitude improvement of response time over traditional GIS spatial joins. Finally, we conceptualize the online data retrieval problem as a more general problem of solving spatial range queries using only $k$-Nearest Neighbor ($k$-NN) queries. Consequently, based on the classification of $k$-NN interfaces on the Web, we propose a set of range query algorithms to completely cover the rectangular shape of a spatial range query while minimizing the number of $k$-NN queries.