A DATA INTEGRATION APPROACH TO DYNAMICALLY FUSING GEOSPATIAL SOURCES

by

Snehal Thakkar

A Thesis Presented to the
FACULTY OF THE GRADUATE SCHOOL
UNIVERSITY OF SOUTHERN CALIFORNIA
In Partial Fulfillment of the
Requirements for the Degree
MASTER OF SCIENCE
(COMPUTER SCIENCE)

December 2007

Dedication

To my parents,

for their unrelenting encouragement and love in face of never ending ambitions.

Acknowledgments

I would like to thank my thesis advisors Professor Craig Knoblock and Professor José Luis Ambite for their constant encouragement and valuable feedback on every aspect of my research. Their feedback has been instrumental in both shaping and presenting my research ideas. They have taught me how to present my ideas in research papers and in presentations. I also want to thank them for not just giving me the freedom to explore different ideas for research, but also for encouraging me to continue with the research despite several paper rejections. I would not have been able to complete this dissertation without their feedback and encouragement.

I am also grateful to Professor Cyrus Shahabi for his support when I started at USC in August 2000. In addition to helping me with my PhD research, Professor Shahabi funded my work when I was pursuing Masters degree and also put me in touch with Professor Knoblock. He also recommended me to Professor Knoblock and advised me to stay at ISI after Masters.

I also want to extend thanks to Professor John Wilson, the outside member on my committee. Professor Wilson's comments have helped in connecting my research work with the existing efforts in the GIS community. In particular, he pointed me to several related papers in the field and inspired me to integrate my work with ESRI's toolkit. I

believe that his feedback has made sure that my research will be useful to GIS researchers as well.

I would also like to thank Professor Louiqa Raschid and Professor Felix Neumann for their feedback on my work. Professor Raschid took a personal interest in my work and put me in touch with the team working on the Sahana disaster response framework. The information I received by reading experiences with the Sahana framework has helped me in understanding the requirements of such applications. I had several discussions with Professor Neumann about representing quality in a data integration systems and he also gave me pointers to several papers on quality-driven data integration. I would also like to thank Dr. Jason Chen and Dan Goldberg for their support of my work. I utilize the quality information from the experiments discussed in Jason's thesis. Dan's work on 'reverse-engineering' ArcIMS services inspired me to think about including support for ArcIMS services in my work.

I would also like to thank my co-workers at the Information Integration group at ISI. In particular, I would like to thank Maria Muslea for her help in programming the Prometheus data integration system and helping me out with programming issues. I would also like to thank Dr. Greg Barish for his help with various Theseus issues and advise about graduate school. I would also like to thank the administrative staff at ISI, in particular Alma Nava, for their help during my stay at ISI. I would also like to thank my officemates Matt Michelson, Martin Michalowski, Rattapoom Tuchinda, and Mark Carman for providing me excellent work environment, proof-reading my paper drafts, giving me feedback on research ideas, and providing encouragement even when

my research seem to be heading nowhere. I would also like to thank Wesley Kerr for 'donating' his television and playstation 2 for our entertainment.

Finally, I would also like to thank my family for there support during the long years of research. First I would like to thank Sharad uncle and Meena auntie, who treated me as their own son after coming to USA and inspired me to be the best that I could be. In addition to financial support they provided me a home and an emotional outlet that I could always count on. Next, I would like to thank my parents. They have always supported me in every imaginable way during my studies. From the time when I was a little kid, they instilled the habits to work hard and value the feedback from other people. In order to wake me up to study, they endured sleepless nights. When I grew up and came to USA, they sacrificed established careers in India and then again sacrificed their careers in Dayton, Ohio to support me in my quest. I would also like to thank my brother Hetal for his support during this journey. Throughout my life he has been my best friend. Like my parents, he too endured hard times to support my studies and also proof-read several papers for me. Finally, I would also like to thank my wife Kruti for supporting my studies and providing an outlet to discuss the happenings at work. She also took the brunt of the tension when things weren't going according to plan at work. The love, encouragement, and support of my family over the years is the true reason behind the research described in this thesis.

This research is based upon work supported in part by the National Science Foundation under Award No. IIS-0324955, in part by the Air Force Office of Scientific Research

under grant numbers FA9550-04-1-0105 and FA9550-07-1-0416, and in part by the Defense Advanced Research Projects Agency (DARPA), through the Department of the Interior, NBC, Acquisition Services Division, under Contract No. NBCHD030010.

The U.S.Government is authorized to reproduce and distribute reports for Governmental purposes notwithstanding any copyright annotation thereon. The views and conclusions contained herein are those of the author and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of any of the above organizations or any person connected with them.

Table of Contents

List Of Tables

List Of Figures

Abstract

Accurate and efficient integration of geospatial data is an important problem with implications in critical areas such as emergency response and urban planning. Some of the key challenges in supporting large-scale geospatial data integration are: (1) automatically representing a large number of geospatial source available on the web by utilizing various geospatial data access standards, (2) handling different geospatial data formats and access patterns, (3) assessing the quality of the data provided by a large number of geospatial sources, and (4) automatically providing high quality answers to the user queries based on a quality criteria supplied by the user. In this thesis I describe my research on efficient and accurate integration of geospatial data from a large number of sources. In particular, I describe a representation methodology for declaratively describing the content and the quality of data provided by sources in a data integration system. I discuss methods to automatically generate the descriptions of both the content and the quality of data provided by geospatial sources. I describe a quality-driven query answering algorithm that exploits the descriptions of the content provided by the geospatial sources to generate an initial data integration plan that answers a given user query and optimizes the generated plan by utilizing the description of the quality of data provided by the sources and the quality criteria specified by the user. I also present a mapping of the generated

integration plan into a program that can be efficiently executed by a streaming, dataflow-style execution engine. I implement my techniques in a framework called Quality-driven Geospatial Mediator (QGM). My experimental evaluation in automatically representing over 1200 real-world geospatial sources shows that QGM accurately generates the descriptions of the content and the quality of geospatial sources. The empirical evaluation of QGM's query answering techniques using over 1200 real-world sources shows that QGM provides better quality data in response to the user queries compared to the traditional data integration systems and does so with lower response time.

Chapter 1

Introduction

1.1 Motivation and Problem Statement

There are a variety of government agencies, university departments, and commercial companies that provide different types of geospatial data of different quality and coverage. In fact, according to a report by the US Federal Geographic Data Committee (FGDC), geographic location is a key feature of 80-90% of all government data [42]. Many critical applications such as responding to unexpected events and urban planning, depend on the ability to efficiently access and integrate data from the available geospatial sources. In a book for University Consortium of Geographic Information Systems (UCGIS) titled, "A Research Agenda for Geographic Information Science", Onsrud et. al. [56] identify a framework to dynamically integrate geospatial data as one of the key research agendas for future work.

While it is hard to estimate the total number of available geospatial data sources in the web, a quick check on collections of available data reveals the following statistics: (1) The Geospatial Information Database project at the Navy Research Laboratory provides

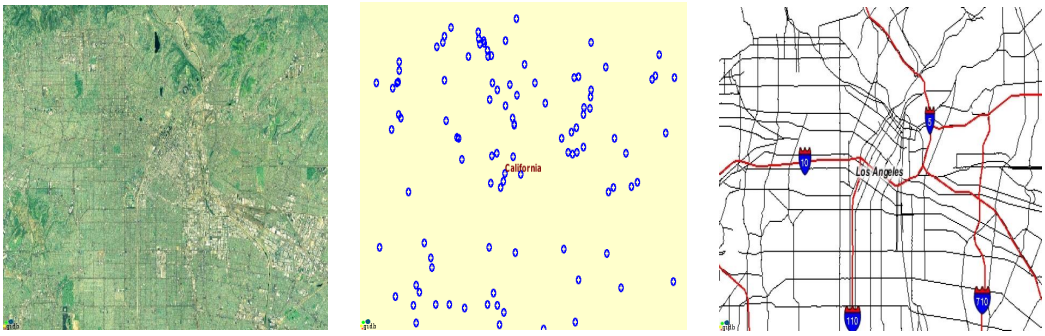

Figure 1.1: (a) Satellite Imagery, (b) Hospitals, and (c) Road Network

a list of over 1400 Web Map Servers that provide over 200,000 map data layers. (2) For vector data, a Google search for keywords 'download shapefile' produces over 344,000 result pages. These simple statistics clearly show that there are lots of available geospatial data sources on the web.

However, access to the available data is limited to a very small number of people who are good at locating geospatial data and have the correct GIS software to view the data in different formats. In fact, a number of disaster management and urban planning applications would greatly benefit from access to an integrated view of geospatial data.

Consider the example of finding information about recent wild fires in Los Angeles, CA. Given the coordinates of the fire, we would like to find satellite imagery, hospitals, and road networks in the area. We can obtain this information separately from different publicly available web sources. Figure 1.1 shows the result of querying this information from different Web Map Servers. However, this data is not easily accessible as users have to find the correct data source among thousands of available sources.

Figure 1.2: Example of Inaccuracies in Geospatial Data

Moreover, it is also important to assure the quality of integrated data. Geospatial data from different sources often has different accuracy levels due to different data collection methods and projections. Figure 1.2 shows Tigerlines vector data overlaid on the high-resolution satellite imagery from Google Maps. We can clearly see several positional inaccuracies in the vector data superimposed over the image. In several disaster management applications such inaccuracies are not acceptable. Therefore, when we integrate geospatial data, we must ensure the quality of the integrated data.

Another example of geospatial data integration is the Green Visions project [62] for addressing the need for more green space in Los Angeles. Urban planning applications similar to the Green Visions project often require ad-hoc queries such as finding the total area of public parks within half a mile of a given property parcel. In addition to selecting

the best-quality sources, a data integration system must also retrieve relevant data (parks in the given area), perform necessary filtering (distance between parks and the given parcel), and aggregate the data. In such applications the quality of the answers depends heavily on the quality of the information. Therefore, the ability of a data integration system to dynamically produce the best-quality data is a very important capability.

Large-scale geospatial data integration problems requires solving several challenges. Some of these challenges are similar to the challenges faced by traditional data integration systems. For example, geospatial data is often in different formats and schemas of geospatial sources are often different. There are also several new challenges when integrating geospatial data sources. First, there may be a very large number geospatial data sources relevant to the user query. While the traditional data integration systems may be able to generate programs to integrate a large number of sources, executing the generated programs may take a long time and may generate low quality results. Second, as the geospatial data is often very large, it can take a long time to retrieve and process geospatial data. Therefore, a geospatial data integration system must only retrieve necessary data. For example, if we have access to five different sources of road vector data, only the sources that provide data for the given area should be queried. Third, often there are many available geospatial sources that provide the same type of data with different quality. For example, if we have access to five different sources of road vector data for the given area with different quality levels, then we need to only retrieve the best-quality data. A geospatial data integration system must carefully select data sources to ensure that the best-quality data is returned to the user. Finally, the quality of geospatial data can be measured using different dimensions, such as the resolution at which the data was

collected or the date on which the data was collected. Depending on the user requirements, some dimensions may be more important than others. Therefore, we cannot use a pre-defined quality ranking of all available sources, but we must provide the user the flexibility to define his/her quality requirements.

The existing work in the field does not support efficient, accurate, and flexible integration of data from a large number of geospatial sources. The work on geospatial data integration can be broadly classified in four areas: (1) web-based GIS mapping sites and warehouses, (2) desktop GIS systems, (3) data integration applied to geospatial domain, and (4) geospatial fusion. There are a large number of web-based mapping sites, such as Google Maps[1] and Microsoft Virtual Earth.[2] These sites allow users to look at satellite imagery and road maps of different areas. It is also possible for users to display their own data on top of the mapping sites. There are also web-based geospatial data warehouses, such as Geospatial One-Stop[3], National Atlas[4], and Geography Network.[5] These sites allow users to select layers of interest and superimpose them on top of each other to create different maps. While these websites are great for accessing a large amount of geospatial data, they provide very limited querying or integration capability and put the burden of selecting the best-quality data on the user.

Desktop GIS systems, such as ESRI's ArcGIS[6], Cadcorp Systems Map Browser[7], and GiDB client[8] allow users to retrieve and visualize geospatial data. Most desktop GIS

[1] http://maps.google.com
[2] http://local.live.com
[3] http://www.geodata.gov
[4] http://www.nationalatlas.gov
[5] http://www.geographynetwork.com
[6] http://www.esri.com
[7] http://www.cadcorp.com
[8] http://dmap.nrlssc.navy.mil/dmap/

systems utilize the OpenGIS[9] standards to retrieve geospatial data from different web sources. The desktop GIS systems allow users to quickly visualize and query geospatial data. However, the user is responsible for selecting data sources and data layers.

There has been some work on applying traditional data integration techniques to the geospatial domain [1, 22, 35, 26, 70]. The major contribution of these works is to show the feasibility of applying traditional data integration techniques to the geospatial domain. However, most of this work does not focus on the quality of the geospatial data returned by the integration framework.

There exists a body of work on conflation techniques [60] to accurately fuse different types of geospatial datasets. There are several manual, semi-automatic, and a very few automatic techniques to dynamically align different geospatial datasets. I refer to this body of work as physical integration of geospatial datasets as it requires altering the data. Conflation operations often improve the quality of geospatial data. However, work on conflation operations has not been utilized in a geospatial data integration system to improve the quality of results.

This thesis is motivated by the lack of algorithms to accurately and efficiently retrieve and integrate different types of geospatial data from a large number of sources. In particular, my thesis addresses the task of quickly learning representations of a large number of geospatial sources, automatically estimating and representing quality and completeness of the sources, and using those representations to dynamically generate and execute integration plans that produce the best-quality data in response to user queries.

[9] http://www.opengis.org

1.2 Approach

This thesis provides a novel approach to efficiently and accurately integrating geospatial data from a large number of sources. Based on the issues described in the previous section, the key requirements for large-scale geospatial data integration are (1) represent geospatial sources that provide different types of geospatial data in a variety of formats, (2) dynamically determine relevant sources of data based on a user or application query, and (3) dynamically generate and execute plans that produce the best-quality integrated data based on the quality requirements of the user or application. In this section, I outline a geospatial data integration framework called Quality-driven Geospatial Mediator (QGM). QGM addresses the problem of dynamically and accurately integrating a large number of geospatial sources by declaratively representing both the content and the quality of geospatial data sources and utilizing the representations to provide high quality data in response to user queries.

Data integration systems, such as TSIMMIS [30], HERMES [1], SIMS [2], the Information Manifold [47], Disco [25], Infomaster [31], ARIADNE [45], provide a uniform query interface to multiple information sources with heterogeneous schemas and interfaces. The data integration systems utilize a set of domain relations that are utilized in formulating user queries. The set of available sources are represented as source relations. A domain expert provides a set of rules that serve as the mapping between the source relations and the domain relations. The set of domain relations, source relations, and the mapping rules together are often called the domain model. In this thesis, I describe

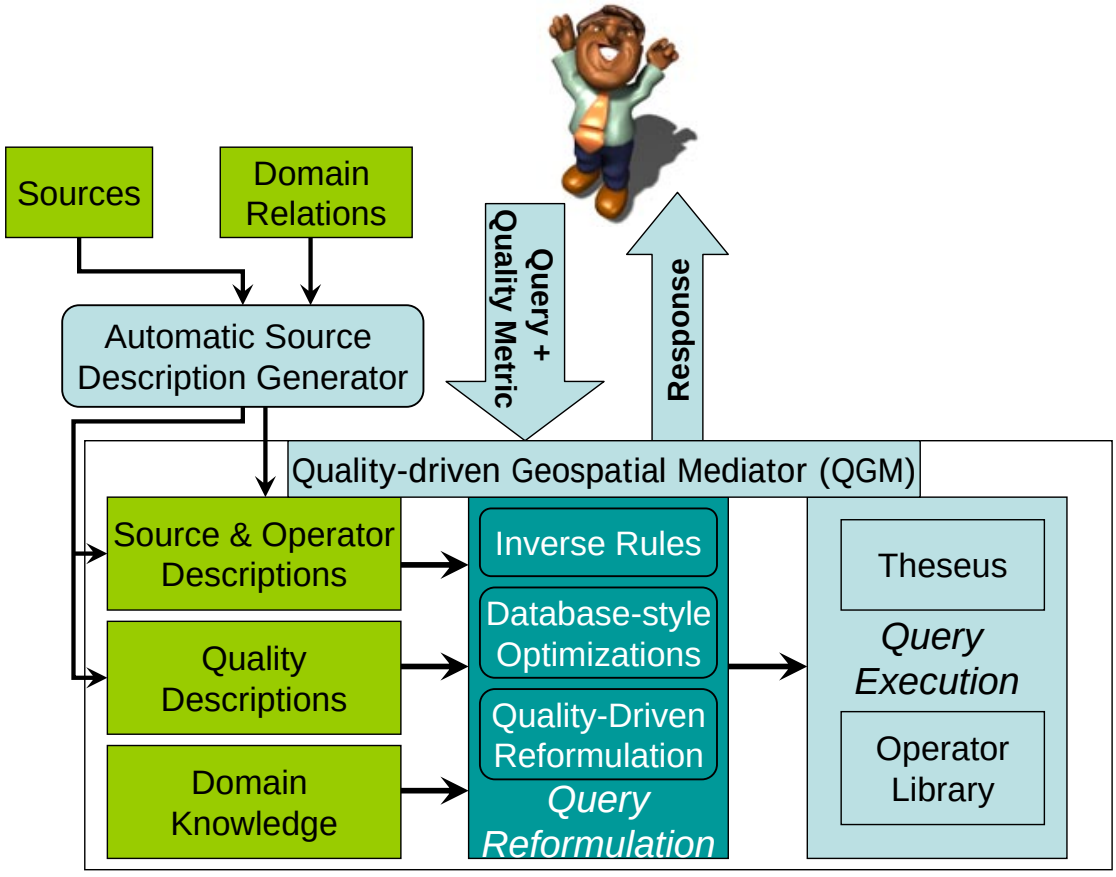

Figure 1.3: QGM Architecture

my approach to developing a general-purpose domain model for integrating geospatial sources.

Figure 1.3 shows the architecture of the QGM framework. QGM is divided in four parts: (1) descriptions of sources and domain knowledge, (2) automatic source description generator, (3) query reformulation and optimization, and (4) plan execution.

Representing Geospatial Sources: The existing geospatial data integration systems only represent the content of the geospatial sources. In QGM, I declaratively represent not just the content of the geospatial sources, but also the quality of the data provided by the geospatial sources. QGM represents the quality of data by allowing a

domain expert to specify a set of relations to represent quality of data. The users can specify the quality criteria using the quality relations defined at the domain level. As a part of the source description, QGM also allows the source provider to specify quality of data provided by the source using a quality relation corresponding to the source. The declarative specification of quality allows flexible definition of quality based-on application requirements.

The existing geospatial data integration systems provide all results that meet the content criteria to the user and it is the user's responsibility to select the best-quality data from that. However, this is a very tedious task for the user to select the best-quality result from a large number of results. For example, a query for road network in city of El Segundo against the set of sources listed on Mapdex[10] returns over 600 images and over 20 vector data sources. Selecting the best-quality data from such a large collection is very time-consuming and error-prone task. Needless to say that making over 620 requests to different data sources also takes a long time. The declarative specification of quality allows QGM to filter out low quality data from the results.

Dynamically generating representations for geospatial sources: There are a very large number of geospatial data sources on the Internet. Each source provides limited types of data and has limited coverage. Therefore, many geospatial data integration applications require integrating a very large number of geospatial sources to ensure enough coverage. Manually representing so many sources is a very tedious and error-prone task. However, there are a large number of geospatial sources that provide some metadata description of its data using OpenGIS standards, ArcXML descriptions, or XML files.

[10]http://www.mapdex.org

```
- <Layer>
    <Title>UFDBstations</Title>
    <LatLonBoundingBox minx="-180.0" miny="-90.0" maxx="180.0" maxy="83.62999725341797"/>
  - <Layer>
      <Name>0:0</Name>
      <Title>Default Backdrop for UFDBstations</Title>
      <LatLonBoundingBox minx="-180.0" miny="-90.0" maxx="180.0" maxy="83.62999725341797"/>
    </Layer>
  - <Layer>
      <Name>0:1</Name>
      <Title>Countries</Title>
      <LatLonBoundingBox minx="-180.0" miny="-90.0" maxx="180.0" maxy="83.62359619140625"/>
    </Layer>
  - <Layer>
      <Name>0:2</Name>
      <Title>Countries with Backdrop</Title>
      <LatLonBoundingBox minx="-180.0" miny="-90.0" maxx="180.0" maxy="83.62359619140625"/>
    </Layer>
```

Figure 1.4: Examples of Layer descriptions

QGM can dynamically generate the representations of the data using the metadata. There are three key elements in the description of a geospatial data source: (1) what type of data does a source provide, (2) what is the area covered by the data, and (3) what is the quality of data provided by the source. If a source supports GIS standards, it must provide a capabilities file similar to one shown in Figure 1.4 that lists available data layers and their coverage. However, the names of the available data layers may not directly map to data layers in the domain ontology. QGM utilizes string matching techniques to link layer names from the sources with the layer names in the domain ontology. Moreover, sources may utilize different coordinate systems. The automatic source description generator module of QGM includes the necessary coordinate conversion operations.

As an example, consider the task of providing data to disaster management applications. Due to the unpredictable nature of disasters, we must cover the maximum possible area and layers. Therefore, we must find a large number of sources. QGM utilizes the

Google Search Engine[11] to find all available Web Map Servers. Next, QGM's automated source description generator module generates the source descriptions of all the servers. By utilizing this method, QGM generates over 293,000 source descriptions.

The third part of the description of a data sources is the description of the quality of data provided by a source. QGM samples the data from the new sources and compares it with sources of known quality and overlapping coverage to generate an estimate for some of the quality attributes of the new sources.

Quality-driven plan selection: When a user or an application asks QGM a query, it utilizes the source descriptions to generate an integration plan that answers the user query. QGM utilizes the user-supplied quality metric and the descriptions of the quality of data provided by the available sources to prune source requests that do not satisfy the quality criteria from the generated integration plan.

I evaluate my approach by presenting results of experiments on a large number of real-world geospatial datasets to show that my approach can (1) integrate geospatial data from a large number of sources and (2) provide the best-quality data for several real-world user queries and quality metrics.

1.3 Thesis Statement

This thesis demonstrates that by discovering geospatial sources available on the web, automatically learning the representations of both the content and the quality of data provided by the discovered sources, and exploiting the

[11] http://www.google.com

representations of the sources during query answering we can provide high quality geospatial data in response to user queries.

1.4 Contributions

The key contribution of this thesis is a Quality-driven Geospatial Mediator (QGM) framework to accurately and efficiently integrate geospatial sources and operations. There are four novel research contributions of the QGM framework:

- A declarative specification of both the content and the quality of geospatial sources in a data integration system.
- Algorithms to automatically generate source descriptions and estimate the quality of data provided by geospatial data sources available on the Internet.
- A quality-driven query reformulation algorithm to dynamically generate and optimize an integration plan that provides high-quality data for the given user query and quality metric.
- An approach to map the generated integration plans and source requests to a program that is efficiently executed by a streaming, dataflow-style execution engine.

1.5 Thesis Organization

The remainder of this thesis is organized as follows. Chapter 2 describes the representation of geospatial domain concepts, sources, and operations in QGM. Chapter 3 presents an approach to automatically generate descriptions for geospatial sources and estimate

the quality of the data. Chapter 4 discusses the quality-driven query answering algorithm. Chapter 5 reviews the work related to this research. Chapter 6 recaps the contributions of this thesis and presents ideas for future work.

Chapter 2

Representing Geospatial Data

A key aspect of any mediation system [1, 2, 23, 25, 30, 31, 45, 47] is an integration model that defines the domain relations, sources, their relationships, and domain rules that define relationships between different domain relations. In this chapter, I describe my approach to representing geospatial domain concepts, sources, and operations in my mediation framework called Quality-Driven Geospatial Mediation (QGM). The representation task is divided into four parts: (1) defining a set of domain relations, (2) describing the available geospatial sources and their quality, (3) defining operations, and (4) representing user queries. I describe each in turn.

2.1 Geospatial Domain Concepts

A domain model for any mediator system has a mediated schema [30] consisting of a set of relations that refer to different entities in the domain. The relations in the mediated schema provides a uniform interface for the users to pose queries to the mediator. In this section, I describe a hierarchy of domain relations referring to different types of geospatial

data and their quality. I utilize the domain concept hierarchy as a mediated schema for QGM.

2.1.1 Domain Concepts for Geospatial Data

Traditionally, a domain expert analyzes different queries and available data in the domain to determine the mediated schema. I merged domain concepts from the FGDC list of geographic concepts[1], the hierarchy of geospatial data types from the National Atlas[2], and the National Geospatial Agency (NGA) spatial data types[3] to determine different geospatial data types and corresponding domain relations.

I divide the data in geospatial domain into *Raster* and *Vector* types based on the major data types in the NGA spatial data types. An example of *Raster* data is a map of an area, while a set of line segments denoting roads in the area is an example of *Vector* data. *Raster* data represents a rectangular grid, with each element in the grid containing some data value. In contrast, *Vector* data represents features with geometric shapes, such as points, lines, or polygons.

I define twelve different categories for the *Vector* data, such as *Transportation* and *Hydrography*, by merging the categories in NGA vector data products, FGDC major categories, and categories from the National Atlas. *Raster* data is divided into three subclasses representing maps, images, and elevation data.[4] The three categories for the raster data are based on three types of raster data available from NGA. I further divide

[1] http://clearinghouse1.fgdc.gov/servlet/FGDCWizard
[2] http://www.nationalatlas.gov
[3] http://earth-info.nga.mil/publications/specs/
[4] Depending on the organization of elevation data, it can be included within raster or vector types. If the elevation data is in the form of contour lines, it is vector data, while elevation data in the form of a uniform grid is considered raster data. Therefore, in my generic model I add elevation data to both types.

each category of *Raster* and *Vector* into several subcategories. The different categories of the *Vector* data are divided into 35 subcategories, while the three categories of the *Raster* data are divided into six subcategories. Table 2.1 and Table 2.2 show all *Vector* and *Raster* data categories and subcategories, respectively.

Each subcategory consists of different data layers. A data layer refers to a specific type of data and is identified by the name of the layer, such *Rivers* or *Airports*. There are 167 different *Vector* data layers and 12 *Raster* data layers. Appendix A provides complete list of data layers in my geospatial domain model.

I developed a geospatial model that has concepts for *Vector* and *Raster* data, categories for both types of data, subcategories, and data layers. Figure 2.1 shows a partial set of domain relations from the resulting domain hierarchy.

I represent each concept in the hierarchy as a relation with a set of attributes. The relations for the *Raster* and *Vector* domain concepts are as follows:

```
Raster(type, format, size, resolution, cs, bbox, source, rasterobj)
Vector(type, format, cs, bbox, source, vectorobj)
```

For the *Raster* domain concepts, I use the following attributes: *type, format, size, resolution, cs, bbox, source*, and *rasterobj*. These attributes are based on the attributes associated with images by various image sources, such as Microsoft TerraServer. The *type* attribute contains the name of the domain relation, for example, *Raster*. The *type* attribute is used in the queries to combine the data objects with the quality information. The *format* attribute contains information about the image format, such as JPEG or GeoTIFF. The *size* attribute refers to the height and the width of the image. The *cs*

Type	Category	Subcategory
Vector	Boundaries	Administrative
		Political
		Other
	Buildings	Schools
		Churches
		Hospitals
		Other
	Cartography	Property
	Elevation	Contours
		Other
	Hydrography	Other
		Coast
		Rivers/Lakes
	Industry	Agriculture
		Buildings
		Other
		Facilities
		Commercial
	Physiography	Physiography
	Population	Population
	Transportation	Air
		Other
		Ground
		Water
	Utilities	Communication
		Pipelines
		Power
		Other
	Vegetation	Other
		Agriculture
	Weather	Temperature
		Rainfall
		Wind
		Stations
		Warnings

Table 2.1: Categories and Sub-categories of Vector Data

Type	Category	Subcategory
Raster	Image	Aerial Photos Satellite Images
	Maps	Topo Maps Transporation Maps Weather Maps
	Elevation	Elevation Grids

Table 2.2: Categories and Sub-categories of Raster Data

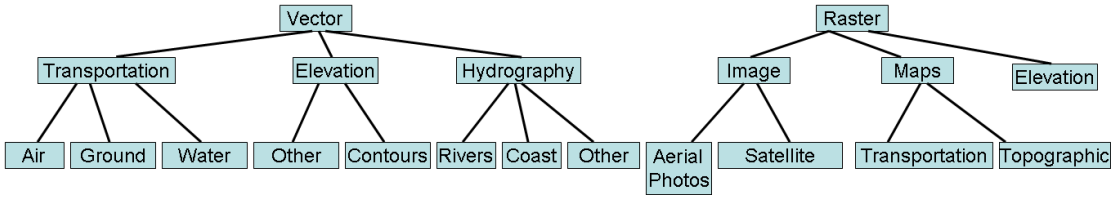

Figure 2.1: Partial QGM Domain Concepts Hierarchy

attribute represents the coordinate system in which the bounding box of the image is described. The *bbox* attribute contains the bounding box of the image. The *source* attribute provides information about the source of the image. Finally, the *rasterobj* attribute contains a pointer to the actual image.

For the *Vector* domain concept, I use the following set of attributes: *type, format, cs, bbox, source*, and *vectorobj*. The *type* attribute is used for the same purpose as the *type* attribute in the *Raster* class. The *format* attribute contains information about the format of the vector data, such as a Shapefile or GML. The *cs* attribute represents the coordinate system in which the bounding box of the vector data is described. The *bbox* attribute contains the bounding box of the vector data. The *source* attribute provides information about the source of the data. Finally, the *vectorobj* attribute contains a pointer to the actual vector data.

```
D1:Vector('Vector', format, cs, bbox, source, vectorobj):-
   Transportation(type, format, cs, bbox, source, vectorobj)
D2:Transportation('Transportation', format, cs, bbox, source,
         vectorobj):-
   GroundTransportation(type, format, cs, bbox, source, vectorobj)
```

Figure 2.2: Domain Hierarchy Rules

While I have chosen these attributes for my domain model, QGM's query answering mechanism does not rely on a specific schema for domain relations and users can choose their own schema. However, when selecting the schema for the domain relations, users must ensure that they select attributes that allow QGM to join the domain relations for different concepts with the corresponding quality relations. I selected this schema as it allows me to easily express different types of queries I found in various real-world geospatial data integration applications.

In addition to the set of domain concepts, I define the relationships between different domain concepts. I use logic rules to describe the class-subclass relationships in the domain hierarchy. Figure 2.2 shows the rules that describe the relationship between the *Vector* and *Transportation* and *Transportation* and *GroundTraponsportation* relations. The rule *D1* states that transportation data is one type of vector data, while the rule *D2* states that ground transportation data is one type of transportation vector data.

I believe that the domain concept hierarchy that I developed will be useful in most general-purpose geospatial data integration system. However, for specific applications a domain expert can specify a mediated schema of their choice to QGM or modify the model that I developed.

2.1.2 Domain Concepts for Geosptial Data Quality

In addition to describing different types of geospatial data, I also describe the quality of data using a set of domain relations with one or more attributes per quality measure. In this thesis, I utilize an extended version of the quality standards[5] proposed by the Federal Geographic Data Committee (FGDC) to describe the quality of information provided by the sources. For more specific applications, QGM allows a domain expert to define his or her choice of quality attributes.

I represent the quality of each type of data as a relation with a set of attributes. The quality relation for the *Raster* domain relation has six attributes: one attribute for the type of the image object, one attribute for the source, and four attributes corresponding to the dimensions of quality for images.

```
ImageQuality(source, type, date, originalresolution, multispectral,
             completeness)
```

The *original resolution* defines the area of the ground represented in each pixel in x and y components. An example value for the original resolution attribute is 1 meters/pixel. The *multi-spectral* attribute equals 'false' if the image in the raster object is a black-and-white image. If the image has values for all four bands (Red, Green, Blue, and InfraRed), the *multi-spectral* attribute has value 'true'. Figure 2.3 shows an example of black-and-white image, while Figure 2.4 shows an example of multi-spectral satellite image. The attribute *completeness* refers to the percentage area of the bounding box given in the source description for which the source provides raster data.

[5] http://www.fgdc.gov/standards/projects/FGDC-standards-projects/metadata/base-metadata/

Figure 2.3: Example Black-and-white Image

The quality relation for the *Vector* domain relation contains eight attributes based on the FGDC quality metadata standards.

```
VectorQuality(source, type, resolution, date, horizontalaccuracy,
              verticalaccuracy, vectorswithinaccuracybounds,
              attributecompleteness, completeness)
```

The *resolution* attribute refers to the resolution of the image that was used to collect the vector data. The *date* attribute refers to the date on which the image used to collect the vector data was taken. The *horizontal accuracy* and *vertical accuracy* refer to the

Figure 2.4: Example Multi-spectral Image

accuracy in the x and y dimension. The *vectors within accuracy bounds*[6] attribute refers to the percentage of vectors out of all vectors provided by the source that fall within the buffer around the actual location of the feature with width and height defined by the horizontal and vertical accuracy, respectively. The three attributes relate to the following quality information that is often present in the metadata provided by sources: The location of the provided features is accurate within 'n units' for 'k %' features. The 'n units' refer to the accuracy bounds for the dataset, while the 'k%' refers to the *features within accuracy bounds* attribute. Figure 2.5 shows an example of buffer around the actual vector data and the same segment obtained from a source. The *attribute*

[6]In the FGDC metadata standard this attribute is called completeness. To avoid confusion with the traditional view of completeness in the data integration literature, I use the phrase vectors within accuracy bounds.

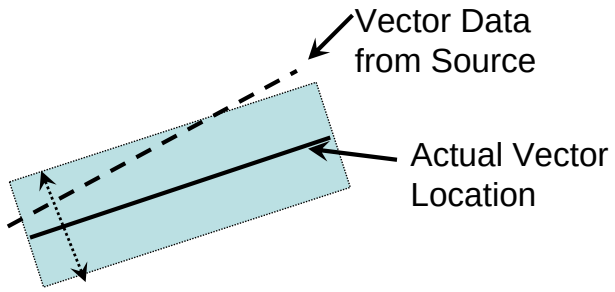

Figure 2.5: Example of Buffer Around Actual Vector Location

completeness refers to the percentage of vectors for which attribute values are available. The *completeness* attribute refers to the percentage of actual feature vectors out of all existing vectors provided by the data source.

I assume that all the subclasses of the *Raster* and *Vector* domain concepts have the same attributes for the quality measures. While I have selected the quality attributes recommended by FGDC for our motivating example, the quality representation in QGM is generic. Depending on the application requirements, users can define different dimensions of quality.

2.2 Geospatial Sources

An integral part of a domain model is a set of definitions for all available sources that relate the sources with the domain concepts. In this section, I describe my approach to defining the sources in QGM. I divide the task of defining a source into two parts. First, I describe the content of the sources with respect to the domain concepts. Second, I discuss my approach to describing the quality of data provided by the sources.

2.2.1 Describing Contents of the Sources

In this section, I show my approach to relating the contents provided by different sources with the domain concepts from the domain concepts hierarchy described in Section 2.1. I divide the task of defining contents of a source into two parts. First, I assign a relational representation for the source and second, I describe the contents of the source with respect to the domain relations.

Similar to the domain concepts, I represent each source as a relation of a set of attributes. The set of attributes for a source are all of its input attributes (if any) and all the output attributes. If values for any input attributes are required, I use the suffix *:b* after an attribute to denote a binding restriction [47] on that attribute and suffix *:f* to denote an attribute without a binding restriction.

Table 2.3 shows examples of relational representations of several sources that provide raster and feature vector data. The *Tigerlines* source[7] requires a bounding box and provides the road network data for that area. The *LARivers* data source[8] requires a bounding box and provides a vector object containing the rivers for the area covered by the bounding box. The *CasilTransportation* source[9] requires both the bounding box and the type of the transportation vector data (Roads, Railroads, or Airports), and provides a vector object containing the requested data. The *TerraServer* data source[10] requires a bounding box, size of the image, type of the image (topographic maps, b/w satellite image, or multi-spectral satellite image), resolution of the image, format of the image (JPG, GIF,

[7] http://www.census.gov/tiger
[8] http://greenvisionsplan.net/html/datasets.html
[9] http://gis.ca.gov/
[10] http://terraserver.microsoft.com

Sources
Tigerlines(bbox:b,vectorobj:f)
LARivers(bbox:b,vectorobj:f)
CasilTransportation(bbox:b,vectortype:b,vectorobj:f)
TerraServer(bbox:b, size:b, imagetype:b, resolution:b, format:b, imageobj:f)
NOAAImages(bbox:b,imageobj:f)

Table 2.3: Examples of relational representations of sources

or PNG), and provides a URL to the requested image. The *NOAAImages* data source requires a bounding box and provides a URL to a high-resolution multi-spectral satellite image.

The next step is to define the relationship between the contents of the sources and the domain concepts. I utilize the Local-As-View model [46, 47] to describe the relationships between the sources and domain relations. In the Local-As-View model, we define each source as a view over the domain relations. In the geospatial domain, sources usually provide information about one or more types of data layers in some region. Therefore, I represent each source with a set of rules to describe the contents of the source and the coverage restrictions. Each logic rule contains the source relation in the head and a conjunction of domain relations and constraints with spatial or relational selections in the body of the rule. The coverage restrictions are described using constraints.

I define the sources that provide different types of feature vector data as views over the domain relations representing different subclasses of the *Vector* domain concept. For example, the *Tigerlines* data source road vector data for the continental United States is described using the rule *S1* in Figure 2.6. The rule *S1* describes the *Tigerlines* data source over the *Roads* domain relation. We also specify the coverage restrictions using

a spatial selection operation (*coveredby*) on the *bbox* attribute to denote that the source only provides data for bounding box within the United States.

The *LARivers* data source provides vector data describing all rivers in the Los Angeles metro area. The rule *S2* describes this source as a view over the *Rivers* domain relation. The *CasilTranportation* data source provides three different types of vector data covering the state of California. Therefore, I use three rules to describe the data source. Note that while the head relation of the three rules are *CasilTransportation1*, *CasilTransportation2*, and *CasilTransportation3*, the source attribute in all rules is set to 'Casil'. I use three rules with different predicates in the head to avoid disjunctive rules. In general case disjunctive source descriptions in a Local-As-View model may result in increasing the complexity of the query reformulation process from polynomial time to co-NP [20]. The rule *S3* describes the *CasilTransportation1* as a view over the *Roads* relation, while the rules *S4* and *S5* describe the sources *CasilTransportation2* and *CasilTransportation3* as views over the *Railroads* and the *Airport* relation, respectively. Note that in each rule, the second attribute in the head relation contains the name of the data layer. This ensures that all requests to the source will only result in data referring to a particular layer, such as *Roads*.

Similar to vector data sources, we can also describe raster data sources as views over different domain concepts. For example, consider the TerraServer data source that provides topographic maps, black and white satellite imagery, and multi-spectral satellite imagery for the continental United States. We describe this source using three rules, one per each type of raster data provided by the sources. These rules are shown in Figure 2.7. Rules *S6,S7*, and *S8* describe the three types of raster data provided by TerraServer. For

```
S1:Tigerlines(bbox,vectorobj):-
     Roads(type, format, cs, bbox, source, vectorobj)^
     bbox coveredby '[[17,65],[71,-168]]'^
     vectorobj coveredby bbox^
     cs = 'EPSG:4326'^ format = 'Shape'^
     source = 'Tigerlines'^ type = 'Road'
S2:LARivers(bbox,vectorobj):-
     Rivers(type, format, cs, bbox, source, vectorobj)^
     bbox coveredby '[[33,-117],[35,-120]]'^
     vectorobj coveredby bbox^
     cs = 'EPSG:4326'^ format = 'Shape'^
     source = 'LARivers'^ type = 'River'
S3:CasilTransportation1(bbox,'Roads',vectorobj):-
     Roads(type, format, cs, bbox, source, vectorobj)^
     bbox coveredby '[[32,-115],[37,-123]]'^
     vectorobj coveredby bbox^
     cs = 'EPSG:4326'^ format = 'Shape'^
     source = 'Casil'^ type = 'Road'
S4:CasilTransportation2(bbox,'Railroads',vectorobj):-
     Railroads(type, format, cs, bbox, source, vectorobj)^
     bbox coveredby '[[32,-115],[37,-123]]'^
     vectorobj coveredby bbox^
     cs = 'EPSG:4326'^ format = 'Shape'^
     source = 'Casil'^ type = 'Railroad'
S5:CasilTransportation3(bbox,'Airports',vectorobj):-
     Airports(type, format, cs, bbox, source, vectorobj)^
     bbox coveredby '[[32,-115],[37,-123]]'^
     vectorobj coveredby bbox^
     cs = 'EPSG:4326'^ format = 'Shape'^
     source = 'Casil'^ type = 'Airport'
```

Figure 2.6: Example Source Descriptions for Vector Data Sources

all types of data, TerraServer requires a bounding box, type of the image, resolution of the image , and size of the image and returns the image. Note that the *type* attribute is not necessarily same as the value for the layer name passed to the source. For example, the values for the *imagetype* attribute passed to *TerraServer* are different from the names associated with the concepts in the domain hierarchy in rules *S6-S8*.

```
S6:TerraServer1(bbox:b, imagetype:b, size:b, resolution:b, format:b,
         imageobj):-
     TopographicMaps(type, format, size, resolution, cs, bbox,
       source, imageobj)^
     bbox coveredby '[[17,65],[71,-168]]'^
     source = 'TerraServer'^
     type = 'TopographicMap'^
     imagetype = 'Topographic Maps'
S7:TerraServer2(bbox:b, imagetype:b, size:b, resolution:b, format:b,
         imageobj):-
     BWImages(type, format, size, resolution, cs, bbox,
       source, imageobj)^
     bbox coveredby '[[17,65],[71,-168]]'^
     source = 'TerraServer'^
     type = 'BWImages'^
     imagetype = 'B/W Image'
S8:TerraServer3(bbox:b, imagetype:b, size:b, resolution:b, format:b,
         imageobj):-
     MultiSpectralImage(type, format, size, resolution, cs, bbox,
       source, imageobj)^
     bbox coveredby '[[17,65],[71,-168]]'^
     source = 'TerraServer'^
     type = 'MultiSprectralImage'^
     imagetype = 'Urban Image'
```

Figure 2.7: Example Source Descriptions for Image Data Sources

2.2.2 Representing Quality of Geospatial Sources

In addition to the source descriptions, I also declaratively specify the quality of information provided by each source.

In addition to the quality relations for the domain concepts, I define quality relations for each layer provided by the source as a relation with a set of attributes. I limit the set of attributes in the source quality relations to the attributes of the quality domain relations.

For example, consider the *Tigerlines* data source from our motivating example. As it provides only the road vector data, the source quality relation only contains attributes from the *RoadQuality* domain relation (which has the same attributes as the *VectorQuality* domain relation). However, the source and the type attributes are not present as the source and the type are already in the name of the relation.

```
TigerlineRoadQuality(resolution, date, horizontalaccuracy,
    verticalaccuracy, vectorswithinaccuracybounds, attributecompleteness,
    completeness)
```

Note that in my domain model I represent the quality at source level, i.e. any data object retrieved from the data source has the same quality. Therefore, the source quality relations do not refer to a particular data object, but they contain a reference to the source and the type of the data. If the application required QGM to associate different quality values based on some other attribute, we can change the set of attributes in the quality relations to address that.

As the source quality relations are virtual, I use facts to provide the tuples for the relations. For example, I represent the results reported in [18] for the quality of the *Tigerlines* dataset using the facts shown below.

```
TigerlineRoadQuality(1:100000K, 1/1/2000, 3.6, 3.6, 21, 95, 94)
TigerlineRoadQuality(1:100000K, 1/1/2000, 7.2, 7.2, 40, 95, 94)
TigerlineRoadQuality(1:100000K, 1/1/2000, 10.8, 10.8, 53, 95, 94)
```

Notice that, based on different values of the horizontal and vertical accuracy, we get different values for *vectors within accuracy bounds*. Moreover, higher values for the

accuracy attribute mean a larger buffer size. Typically, at higher values of the accuracy attributes, most sources will have a higher percentage of vectors within accuracy bounds. I define similar facts for all vector and raster data sources. QGM can utilize multiple facts about the quality during the query processing. For example, if the user asks to retrieve road vector datasets such that with horizontal accuracy and vertical accuracy bounds less than 8 meters, the datasets have at least 35% data within the accuracy bounds, QGM can utilize the second quality fact in the list shown above to determine that the Tigerlines dataset satisfies the quality criteria.

2.3 Representing Operations

In this section I discuss my approach to representing different types of operations and their impact of the quality data. In order to support processing data from different sources, I represent four types of operations in QGM: (1) spatial selection operations [37], (2) aggregate operations, (3) mathematical operations, and (4) coordinate conversion operations [34].

2.3.1 Spatial Selection Operations

In order to select data using spatial restrictions, QGM supports spatial selection operations. Spatial selection operations, such as *intersects*, are very important part of a geospatial data integration system as they are used in many tasks. The most common uses of the spatial selection operations are to define coverage of sources, define an area of interest for user queries, and to define constraints on geospatial features.

QGM handles the spatial selection operations in the same manner as all selection operations, such as equality order constraints. Spatial selections can be used in any constraints in the source descriptions, domain rules, or user queries. The syntax for the spatial selection operations is the same as the syntax for the equality operation. An example of a constraint with spatial selection operation are the constraints on the *bbox* attribute in the source description shown below:

```
S1:Tigerlines(bbox,vectorobj):-
     Roads(type, format, cs, bbox, source, vectorobj)^
     bbox coveredby '[[17,65],[71,-168]]'^
     vectorobj coveredby bbox^
     cs = 'EPSG:4326'^
     format = 'Shape'^
     source = 'Tigerlines'^
     type = 'Road'
```

The first constraint on the *bbox* attribute shows the *coveredby* operation applied to an attribute and a constant value, while the second constraint including the *vectorobj* and the *bbox* attributes shows the *coveredby* operation used to define the relationship between two attributes.

As a part of the generic-domain model I implemented three spatial selection operations: (1) *coveredby*, (2) *intersects*, (3) *disjoint*. I use the Geotools open source toolkit to implement all three operations.

2.3.2 Aggregate Operations

The second group of operations supported by QGM are aggregate operations. Unlike the spatial selection operations, QGM only allows the aggregate operations in domain rules and user queries. Each aggregate operation is represented using a relation similar to the source relations and domain relations.

I implemented eight aggregate operations in QGM. In this section, I describe each operation and provide an example of how I utilize the operation in QGM.

Pack: The *pack* operation is used to create an embedded relation. Once a relation has been packed, QGM can apply other aggregation operations on it. The *pack* operation operating on a relation with one attribute is represented using a relation with two arguments. The first argument refers to the attribute(s) to pack, while the second argument refers to the result of the operation. The rule shown below provides an example of *pack* operation.

```
AllRoadCompletenesss(completenessrel):-
  RoadQuality(source, type, resolution, date, horizontalaccuracy,
            verticalaccuracy, vectorswithinaccuracybounds,
            attributecompleteness, completeness)
  pack(completeness, completenessrel)^
  bbox coveredby '[[17,65],[71,-168]]'
```

In the example rule, I use the *pack* operation to create an embedded relation providing information about all values of the completeness attribute for the road vector data. The result of the operation is a relation called *completenessrel* that has exactly one attribute

and one tuple. The value of the attribute in the tuple contains a relation with one column that lists all the completeness values of the road quality relation.

In general, the *pack* operation can be used to create a nested relation with arbitrary number of attributes.

Unpack: The *unpack* operation performs exactly the opposite operation to the *pack* operation. It takes an attribute containing an embedded relation and unpacks the relation. The rule below shows and example of the *unpack* operation.

```
AllRoadCompleteness(completeness):-
  AllRoadCompletenesss(completenessrel)^
  unpack(completenessrel, completeness)
```

The first argument for the *unpack* operation contains the attribute with a relation, while the rest of the arguments are the attributes of the nested relation. In general, the *unpack* operation can be used to unpack arbitrary number of attributes.

Sum, Average, Min, Max: The *sum* operation adds all rows in the given relation. The *average* operation is similar to the *sum* operation, except that it finds the average of all values in the given relation. The *min* operation finds the minimum value from all values in the given relation. The *max* operation finds the maximum value from all values in the given relation.

All four operations are modeled as relations with two attributes. The first attribute contains a relation with one attribute, while the second attribute is the result of the operation. The first attribute must be a relation produced by using the *pack* operation.

The rule below shows an example of finding the sum of all values of the *completeness* attribute of the *RoadQuality* relation:

```
TotalCompleteness(sumofcompleteness):-
  RoadQuality(source, type, resolution, date, horizontalaccuracy,
            verticalaccuracy, vectorswithinaccuracybounds,
            attributecompleteness, completeness)
  pack(completeness, completenessrel)^
  sum(completenessrel, sumofcompleteness)
```

SkylineMin: Skyline queries [9, 57] are used to find all points in a multi-dimensional space which are not dominated by any other point. I utilize the skyline operator to define quality restrictions on the data. The general skyline operations are defined over arbitrary number of attributes. In the current implementation of QGM I implement the skyline operations with two attributes.

The *skylineMin* operation is represented as a relation with two arguments. The first argument is an attribute containing a nested relation with two attributes. The second attribute denoting the result of the operation is also a nested relation containing two attributes. However, all rows present in the relation in the second attribute are not dominated by any other tuples.

```
SkylineMinExample(mincompleteness,mindate):-
  RoadQuality(source, type, resolution, date, horizontalaccuracy,
            verticalaccuracy, vectorswithinaccuracybounds,
            attributecompleteness, completeness)
```

```
pack(completeness, date, resultrel)^
skylineMin(resultrel, skylineresultsrel)^
unpack(skylineresultsrel, mincompleteness, mindate)
```

This rules obtains all values of *completeness* and *date* attributes such that no other tuple contains lower values for both attributes using the *skylineMin* operation. First, it uses the *pack* operation to create a relation with all pairs of *completeness* and *date* values. Next, it uses the *skylineMin* operation to find all non-dominated rows. Finally, it unpacks the relation using the *unpack* operation.

SkylineMax: The *skylineMax* operation is similar to the *skylineMin* operation except that it find tuples that have higher values for the attributes instead of lower values.

2.3.3 Mathematical Operations

I also implemented four mathematical operations to assist in computation in QGM. Those operation are: (1) *add*, (2) *subtract*, (3) *multiply*, and (4) *divide*. All of these operations can appear in domain rules and user queries. All these operations are represented with a relation containing three attributes. These operations use the values of the first two attributes as inputs and the third attribute contains the result of the operation. For the *subtract* operation, the second attribute is subtracted from the first attribute. Similarly, for the *divide* operation the first attribute is divided by the second attribute.

2.4 Representing User Queries

The source descriptions, domain rules, operation descriptions, and quality relations together make up the domain model for QGM. Once we have described our domain model, users can ask QGM different queries. A user query in QGM is specified by three logic rules. The first rule specifies the join between the content and the quality criteria. The second rule specifies the quality criteria while the third rule specifies the content criteria. The head of the quality criteria rule contains a predicate with a set of attributes, while the body of the quality criteria rule contains predicates referring to quality relations defined at domain level, any operations, and necessary constraints. Similarly, the content rule contains predicates referring to the domain relations for the content, any necessary operations, and constraints. I explain more details of the query by discussing some examples.

Consider a query to obtain the most recently collected road vector data in the area defined by a bounding box '[[33,-114],[35,-119]]'. In QGM, we can describe this query in two parts. The first part describes the data necessary for the query, while the second part describes the quality requirements for the data. Each part is described using one or more logic rules.

```
Q1(vectorobj, date):-
  Q1Quality(source, type, date)^
  Q1Data(source, type, vectorobj)

Q1Quality(source, type, date):-
```

```
  RoadQuality(source, type, resolution, date, horizontalaccuracy,
              verticalaccuracy, vectorswithinaccuracybounds,
              attributecompleteness, completeness)^
  pack(date, vdaterel)^
  max(vdaterel, maxvdate)^
  date = maxvdate

Q1Data(type, source, vectorobj):-
  Roads(type, format, cs, bbox, source, vectorobj)^
  bbox = '[[33,-114],[35,-119]]'
```

The rule with the *Q1Data* relation in the head describes the data requirement of the query, while the rule with the *Q1Quality* relation represents the quality requirements for the query. The combination of *pack* and *max* operations in the *Q1Quality* rule identify the vector data with the most recent date. The result of this query is a set of URLs pointing to the vector data covering the area of interest in the query. Each result in the URL either provides more coverage of the query area compared to the other results in the set or has been collected more recently compared to other results in the set. If one source provides the most recent data for the entire area, the result of the query is a set with only one URL.

The rule describing the quality metric can also include constraints. For example, the rule below describes a quality metric to find the most recently collected road vector data with at least 75% completeness.

```
Q2Quality(source, type, date, completeness):-
  RoadQuality(source, type, resolution, date, horizontalaccuracy,
              verticalaccuracy, vectorswithinaccuracybounds,
              attributecompleteness, completeness)^
  pack(date, vdaterel)^
  max(vdaterel, maxvdate)^
  date = maxvdate ^
  completeness > 75
```

In addition, QGM also supports skyline queries [9, 57]. Given a set of points in a multi-dimensional space the skyline query returns a set of points such as that for each point there is no point that is better in all dimensions. Skyline queries are very useful in evaluating trade-offs between different quality measures. For example, consider the query to find most recent and most complete road vector data for the same area. I can describe this query to QGM using the following quality metric rule.

```
Q3Quality(source, type, date, completeness):-
  RoadQuality(source, type, resolution, date, horizontalaccuracy,
              verticalaccuracy, vectorswithinaccuracybounds,
              attributecompleteness, completeness)^
  pack(date, completeness, datecompresultrel)^
  skylineMax(datecompresultrel, skylineresultrel)^
  unpack(skylineresultrel, skylinevdate, skylinevcompleteness)^
  date = skylinevdate ^
```

```
  completeness = skylinevcompleteness
```

The rule contains a request to a *skylineMax* operator that finds all rows such that the value for at least one attribute in the *skylineMax* operation (*vdate* or *vcompleteness*) is not dominated by the other rows.

The users are not limited to retrieving just one type of data. Users can also retrieve multiple types of vector or raster data. An example of this is shown in a rule below which retrieves multi-spectral, high-resolution satellite imagery and road vector data for one area.

```
Q4Data(vsource, vtype, isource, iname, vectorobj,imageobj):-
  Roads(vtype, vformat, cs, bbox, source, vectorobj)^
  Multi-spectralImage(iname, itype, iformat, size, resolution, cs, bbox,
       isource, imageobj)
  bbox = '[[33,-114],[35,-119]]'^
  size = '[600,600]'
```

When retrieving multiple datasets, the quality metric can either be independent for each dataset or may have some dependency. For example, use can ask to find vector data and imagery so that the difference between the resolution at which both are collected is minimized using the following quality query.

```
Q5Quality(vsource, vtype, isource, itype, resdiff):-
  RoadQuality(vsource, vtype, vresolution, vdate, horizontalaccuracy,
              verticalaccuracy, vectorswithinaccuracybounds,
              attributecompleteness, vcompleteness)^
```

```
  Multi-spectralQuality(isource, itype, idate, iresolution,
              multispectral, icompleteness)^
  subtract(iresolution,vresolution, resdiff)^
  pack(resdiff, resdiffrel)^
  Min(resdiffrel, minresdiff)^
  resdiff = minresdiff
```

User can also use a Skyline query to describe the quality metric. For example query to find road vector data and multi-spectral, high-resolution imagery such that the difference in the date and the resolution at which the data is collected are minimized.

```
Q6(vectorobj,imageobj, resdiff, datediff):-
  Q6Quality( vsource, vtype, isource, itype, resdiff, datediff)^
  Q6Data(vectorobj,imageobj, vsource, vtype, isource, itype)
Q6Quality(vectorobj,imageobj):-
  RoadQuality(vsource, vtype, vresolution, vdate, horizontalaccuracy,
              verticalaccuracy, vectorswithinaccuracybounds,
              attributecompleteness, vcompleteness)^
  Multi-spectralQuality(isource, itype, idate, iresolution,
              multispectral, icompleteness)^
  subtract(iresolution,vresolution, resdiff)^
  subtract(idate,vdate, datediff)^
  pack(resdiff,datediff, aggregateresultrel)^
  SkylineMin(aggregateresultrel,skylineresultrel)^
```

```
  unpack(skylineresultrel, skylineresdiff, skylinedatediff)^
  skylinedatediff = datediff ^
  skylineresdiff = resdiff
Q6Data(vectorobj, imageobj, vsource, vtype, isource, itype):-
  Roads(vtype, vformat, cs, bbox, vsource, vectorobj)^
  Multi-spectralImage(itype, iformat, size, resolution,
      cs, bbox, isource, imageobj)^
  bbox = '[[33,-114],[35,-119]]'^
  size = '[600,600]'
```

I provide more discussion on the results of different queries and quality metrics in Section 4 where I describe how QGM formulates plans to answer user queries.

Chapter 3

Large Scale Source Modeling for Geospatial Sources

In this chapter, I describe techniques to automatically model a large number of data sources available on the web and estimate the quality of data provided by the sources. First, I discuss several methods to discover geospatial sources. Second, I describe my approach to automatically generate source descriptions for the sources. Third, I describe an approach to estimate values for some of the quality attributes of the data provided by the new sources. Finally, I describe the empirical evaluation of the techniques.

3.1 Discovering Geospatial Sources

The focus of my work is on large-scale integration of geospatial sources. Therefore, I need to identify a large number of geospatial sources. I use two different methods to find geospatial sources. First, I use a search engine to find spatial data sources providing data of different types. I use terms, such as 'download shapefiles' and 'download geospatial data' to locate different types of geospatial data available on the web. For each search term, QGM's source modeling module analyzes all results for links using the Geotools

Open Source Toolkit[1] to identify links pointing to shapefiles (or zipped shapefiles). Using this technique, I have been able to locate over 16,000 different shapefiles containing different types of geospatial data. However, manually checking all the shapefiles to determine the type of data they provide, their coverage, and the quality of data they provide is not feasible due to the large number of files. Therefore, I utilize QGM's source modeling module to automatically determine the type of data provided by the shapefiles, their coverage, and the quality of data.

Second, I traverse spatial data repositories to find all sources of geospatial data. There are several geospatial data repositories on the web. I use three such repositories to identify over 300,000 geospatial data layers from over 3,700 sources. The repositories that I have used are: (1) Mapdex[2], (2) GiDB Portal[3], and (3) Geography Network[4]. Mapdex and Geography Network provide lists of thousands of ArcIMS services available on the web, while GiDB Portal provides links to a large number of Web Map Servers.[5] In the future, we can write a specialized geospatial data crawler to identify geospatial sources. I discuss this in Chapter 6.

Once I have discovered the set of sources, I utilize QGM's automatic source description generation capability to generate descriptions for the sources.

[1] http://www.geotools.org
[2] http://mapdex.org
[3] http://dmap.nrlssc.navy.mil/
[4] http://geographynetwork.com
[5] http://www.opengeospatial.org/standards/wms

3.2 Automatic Source Description Generation

In this section, I describe my approach for automatically generating source descriptions for the identified sources. I divide the description of a geospatial data source into two modules. The first module identifies the type of data that the source provides and its coverage, while the second module determines the coordinate system used by the data source.

3.2.1 Matching Sources with Domain Concepts

In this section, I describe the process of mapping information about available data layers from the discovered sources with the concepts from the domain concepts hierarchy discussed in Section 2.1. For each domain concept I have a name and a set of keywords for the data layer. For example, the *Country Boundaries* domain concept has the keywords 'political boundaries' and 'country polygons'. For each data source discovered by searching the web or crawling the repositories, QGM finds the list of data layers provided by the source and matches the names and the keywords for the discovered layers with the names and keywords of the layers in the domain concepts hierarchy.

QGM can automatically generate descriptions of three different types of geospatial data sources: (1) shapefiles, (2) Web Map Servers or Web Feature Servers, and (3) ArcIMS Servers.

The shapefile data sources usually contain only one data layer. QGM utilizes the Geotools open source toolkit to process shapefiles. For each data layer, a shapefile stores the name of the layer, optional set of keywords that describe the layer, a set of features,

Name	Description
scpop2000	South Carolina 2000 Population Cartogram
usast	usstates
counties	uscounties
indiads	India States and Districts
schools	

Table 3.1: Example of name and descriptions extracted from shapefiles

and a table with some metadata for each feature. QGM extracts the name and the keywords of the data layer provided by the shapefile and utilizes the information to match the layer with the domain concepts. Table 3.1 shows examples of layer names and descriptions extracted from shapefiles.

Web Map Servers and Web Feature Servers provide one or more layers of raster data using the standard protocol described by the OpenGIS Consortium. According to the protocol, a server must implement two required operations called *GetCapabilities* and *GetMap* and one optional operation called *GetFeatureInfo*. The *GetCapabilities* operation provides an XML specification of all the data layers provided by the source. Figure 3.1 shows a segment of the capabilities file for a Web Map Server. The tag *Name* contains the name of the data layer, while the tag *Title* provides the description. In some cases, the *Name* tag may contain an internal id for the layer, which would not be informative for the end-user. Therefore, QGM extracts information from both of these tags for all data layers provided by a server from the capabilities file and matches each layer with the domain concepts.

The third type of geospatial sources supported by QGM are the ArcIMS servers. ArcIMS Servers are similar to Web Map Servers, but they utilize ESRI's protocol instead of the OpenGIS protocol. Similar to Web Map Servers, ArcIMS servers also provide

```
− <Layer>
    <Title>Demis World Map</Title>
    <Abstract/>
    <SRS>EPSG:4326</SRS>
    <LatLonBoundingBox minx="-180" miny="-90" maxx="180" maxy="90"/>
    <BoundingBox SRS="EPSG:4326" minx="-184" miny="-90" maxx="180" maxy="90"/>
  − <Layer queryable="1">
      <Name>Bathymetry</Name>
      <Title>Bathymetry</Title>
      <BoundingBox SRS="EPSG:4326" minx="-180" miny="-90" maxx="180" maxy="90"/>
    </Layer>
  − <Layer queryable="1">
      <Name>Countries</Name>
      <Title>Countries</Title>
      <BoundingBox SRS="EPSG:4326" minx="-184" miny="-90" maxx="180" maxy="85"/>
    </Layer>
  − <Layer queryable="1">
      <Name>Topography</Name>
      <Title>Topography</Title>
      <BoundingBox SRS="EPSG:4326" minx="-184" miny="-90" maxx="180" maxy="90"/>
    </Layer>
  − <Layer queryable="0">
      <Name>Hillshading</Name>
      <Title>Hillshading</Title>
      <BoundingBox SRS="EPSG:4326" minx="-180" miny="-90" maxx="180" maxy="90"/>
    </Layer>
```

Figure 3.1: Example capabilities file from Web Map Server

one or more data layers and provide a capabilities file as an XML document. ArcIMS servers provide the specification of data layers using ArcXML[6] schema. Figure 3.2 shows a segment of the capabilities file for an ArcIMS server. The *name* attribute of the *LAYERINFO* tag provides the name of the data layer. QGM extracts information from the *name* attribute for all data layers provided by a server from the capabilities file and matches each layer with the domain concepts.

Once QGM has extracted names and descriptions of all data layers provided by the source, it matches the tokens from the name and the keywords extracted from the source with the name and the keywords of the domain concepts using the Dice similarity score

[6]http://edndoc.esri.com/arcims/9.0/

```
<ARCXML version="1.1">
 - <RESPONSE>
   - <SERVICEINFO>
     - <ENVIRONMENT>
         <LOCALE language="en" country="US"/>
         <UIFONT name="Arial" color="0,0,0" size="12" style="regular"/>
         <SEPARATORS cs=" " ts=";"/>
         <IMAGELIMIT pixelcount="2097152"/>
         <SCREEN dpi="96"/>
         <CAPABILITIES forbidden="" disabledtypes="bmp,tif" servertype="arcmapserver" returngeometry="xmlmode"/>
       </ENVIRONMENT>
     - <LAYOUTINFO pageunits="inches">
         <ENVELOPE minx="0" miny="0" maxx="8.5" maxy="11"/>
       </LAYOUTINFO>
     - <PROPERTIES>
         <FEATURECOORDSYS
         string="GEOGCS["GCS_Assumed_Geographic_1",DATUM["D_North_American_1927",SPHEROID["Clarke_1866",6378206.
         id="104000"/>
         <FILTERCOORDSYS
         string="GEOGCS["GCS_Assumed_Geographic_1",DATUM["D_North_American_1927",SPHEROID["Clarke_1866",6378206.
         id="104000"/>
         <MAPUNITS units="decimal_degrees"/>
         <BACKGROUND color="255,255,255"/>
         <ENVELOPE minx="-205.513538400732" miny="-94.11077166389" maxx="206.07539375395" maxy="91.3014948978037"
       </PROPERTIES>
     - <LAYERINFO type="featureclass" name="Ocean" id="4" visible="true">
         <FCLASS type="polygon"/>
       </LAYERINFO>
     - <LAYERINFO type="featureclass" name="Countries" id="3" visible="true">
         <FCLASS type="polygon"/>
       </LAYERINFO>
     </SERVICEINFO>
   </RESPONSE>
</ARCXML>
```

Figure 3.2: Example ArcIMS Server capabilities file

[67]. For each layer, QGM concatenates the name and the keywords and tokenizes the concatenated string to create a set of tokens. QGM also performs similar operation on the names and keywords of the domain concepts. QGM computes a matching score between each data layer and each domain concept using the dice formula shown below.

$$Dice(TokenSet1, TokenSet2) = \frac{2 * (TokenSet1 \cap TokenSet2)}{|TokenSet1| + |TokenSet2|}$$

The formula finds all matching tokens between the two sets and divides that number by the sum of the number of tokens from both sets. The tokens are considered matching if they match using any of the following transformations: (1) equality, (2) prefix, (3) suffix,

and (4) stemming. For the prefix and suffix transformations, QGM only considers tokens with at least three characters. This measure avoids matches on single characters and very small strings. Matches using all transformations are weighed equally. In future, we can improve QGM's matching technique by utilizing different weights for each transformation.

For each layer, QGM finds the match by selecting the domain concept with the highest similarity score above 0.5. The threshold allows QGM to ignore matches on words that occur very frequently or matches on just one term such as 'polygon'. If no domain concept matches a layer from a shapefile or ArcIMS service, QGM matches that layer to the *Vector* domain concept, while a layer from a Web Map Server with no matching domain concept is matched with the *Raster* domain concept.

For each data layer provided by a source, QGM also needs to determine the coverage of the source, i.e. the area for which the source provides data for the layer. Depending on the type of the source, QGM obtains the coverage differently. If the data is stored in the shapefile format, the metadata component of the shapefile contains the bounding box of the area covered by the source. QGM utilizes the Geotools open source toolkit to obtain the coverage. If a source is a Web Map Server or a Web Feature Server, QGM obtains the coverage from the *BoundingBox* or *LatLonBoundingBox* tags (see Figure 3.1). In case of the ArcIMS sources the coverage information is provided in the *ENVELOPE* tag (see Figure 3.2).

Once QGM has determined the coverage of all data layers provided by a source and a matching domain concept for each layer, QGM can generate a description for the source. For each data layer provided by the source, QGM generates one logic rule describing the source as a view over the domain concept matching with the layer. For example, QGM

generates the following rule for a shapefile data source that provides roads in the area specified by the bounding box '[[17,65],[71,-168]]' (which covers the United States) .

```
usroadssshapefile(bbox,vectorobj):-
     Roads(vtype, format, cs, bbox, source, vectorobj)^
     bbox coveredby '[[17,65],[71,-168]]'^
     vectorobj coveredby bbox^
     cs = 'EPSG:4326'^
     format = 'Shape'^
     source = 'usroads'^
     vtype = 'Road'
```

The source is defined as view over the *Roads* domain concept as the data layer provided by the source matches with the *Roads* domain concept. The constraint on the *bbox* attribute is defined based on the coverage information extracted from the source. I discuss the constraint on the coordinate system(*cs*) attribute in Section 3.2.2.

In case of Web Map Servers, Web Feature Servers, or ArcIMS servers, QGM generates one rule per data layer provided by the source. While the head of all the rules is different to avoid disjunctive rules, the source attribute in all rules is set to the name of the source. The request to obtain data of any layer from the Web Map Servers, Web Feature Servers, or ArcIMS Servers must contain the name of the data layer. Therefore, QGM encodes that information in the logic rules for the source description. For the example Web Map Server capabilities file shown in Figure 3.1, QGM generates the rules shown in Figure 3.3.

```
DemisWMS1(bbox, name, size, resolution, format, imageobj):-
     Raster(itype, iformat, size, resolution, cs, bbox,
       source, imageobj)^
     name = 'Bathymetry'^
     bbox coveredby '[[-90,-180],[90,180]]'^
     source = 'DemisWMS'^
     itype = 'Raster'

DemisWMS2(bbox, name, size, resolution, format, imageobj):-
     CountryMaps(type, format, size, resolution, cs, bbox,
       source, imageobj)^
     bbox coveredby '[[-90,-180],[90,180]]'^
     source = 'DemisWMS'^
     name = 'Countries'^
     type = 'CountryMaps'

DemisWMS3(bbox, name, size, resolution, format, imageobj):-
     TopoMaps(type, format, size, resolution, cs, bbox,
       source, imageobj)^
     bbox coveredby '[[-90,-180],[90,180]]'^
     source = 'DemisWMS'^
     name = 'Topography'
     type = 'TopoMaps'

DemisWMS4(bbox, name, size, resolution, format, imageobj):-
     Raster(type, format, size, resolution, cs, bbox,
       source, imageobj)^
     name = 'Hillshades'^
     bbox coveredby '[[-90,-180],[90,180]]'^
     source = 'DemisWMS'^
     type = 'Raster'
```

Figure 3.3: Example Source Descriptions for Web Map Servers

```
ArcIMS1(bbox, name, vectorobj):-
     Oceans(vtype, format, cs, bbox, source, vectorobj)^
     bbox coveredby '[[33,-117],[35,-119]]'^
     vectorobj coveredby bbox^
     cs = 'EPSG:4326'^
     format = 'ArcXML'^
     source = 'ArcIMS1'^
     vtype = 'Oceans'^
     name = 'Oceans'
ArcIMS2(bbox, name, vectorobj):-
     Countries(vtype, format, cs, bbox, source, vectorobj)^
     bbox coveredby '[[33,-117],[35,-119]]'^
     vectorobj coveredby bbox^
     cs = 'EPSG:4326'^
     format = 'ArcXML'^
     source = 'ArcIMS1'^
     vtype = 'Countries'^
     name = 'Countries'
```

Figure 3.4: Example Source Descriptions for ArcIMS Servers

Each rule corresponds to different layers described in the capabilities file. In the example layers shown in Figure 3.1, QGM finds matching data layers for two of the four layers. The first layer titled 'Bathymetry', QGM does not find a suitable match in the domain concepts hierarchy, so it matches it with the *Raster* domain concept. The second and third layers are matched with the *CountryMaps* and the *TopoMaps* domain concepts. The fourth layer titled 'Hillshades' is also matched with the *Raster* concept as it does not match with any concept. In all rules, QGM puts a constant value for the name attribute in each rule to match with the name of the layer specified in the Web Map Server's capabilities file.

For the ArcIMS Server capabilities file shown in Figure 3.2, QGM generates the rules shown in Figure 3.4. Both rules are similar to the rules generated for the Shapefile data

source. The key difference is that the second attribute of the head relation contains the name of the layer in the data source. QGM puts a constant value for the attribute in each rule to match with the name of the layer specified in the ArcIMS Server's capabilities file.

3.2.2 Managing Coordinate Systems

Geospatial sources on the web often use different coordinate systems to store geospatial data. As a user may not be familiar with different coordinate systems, he or she may want the results of the queries in certain coordinate system. Therefore, a geospatial data integration system that integrates data from a large number of geospatial sources, must be able to transform data from one coordinate system to another. In this section, I describe QGM's approach to handling sources that provide data in different coordinate systems.

As described in Section 2.3, QGM supports a coordinate conversion operation, that accepts a geospatial data object, its coordinate system, and a target coordinate systems and transforms the geospatial data to the target coordinate system. I utilize the coordinate conversion operation to address different coordinate systems.

In QGM I can use different strategies for handling coordinate systems by providing different source descriptions and domain rules to handle coordinate conversions. I investigated three different methods: (1) keep all datasets in their own coordinate systems and only change coordinate systems to meet user requirements, (2) change all datasets to one coordinate system and change coordinate systems based on user requirements, and (3) internally keep only the coverages of datasets in one coordinate system and change coordinate systems as needed.

The key problem with the first approach is that every query requires converting all coverages into the coordinate system specified in the query. Once the coordinate conversions are performed, QGM can determine which sources are relevant to the query. Next, QGM retrieves the data from the relevant sources and performs necessary coordinate conversion operations. In this approach, QGM performs many coordinate system transformations for each query resulting in a slow response time.

The second approach requires that QGM has complete control over sources, which is not the case for Web Map Server or ArcIMS Servers. We can materialize the data provided by the sources, but the materialization process can take a very long time and many resources. Therefore, this approach is not practical.

The third method is a compromise between the first two methods. By keeping the coverage information in one coordinate system, QGM is able to determine if a source provides data for the query area without performing any coordinate conversion operations. If a source provides relevant data in the query area, QGM retrieves the data and performs a coordinate transformation. Also, converting coverages into one coordinate system is not as time-consuming as converting all data records. This method results in the lowest number of coordinate conversions and the fastest query response time. Therefore, in my geospatial domain, I use this method.

This method requires that when QGM generates a source description for a new source, it determines the coordinate system of the source, determines the coverage of the source, converts the coverage into the common coordinate system, and inserts proper coordinate system transformation operations in the source description. For my sample domain, I utilize the *EPSG:4326* coordinate system as the common coordinate system. *EPSG:4326*

is a common geographic projection that refers to WGS84 latitude/longitude coordinates in degrees with Greenwich as the central meridian. However, this parameter is part of QGM's configuration and can be changed easily to suit different applications.

Different types of geospatial sources store the coverage information differently. The geospatial data stored in shapefiles often contains a projection file that records the coordinate system of the data source. Web Map Servers provide the coordinate system information in the *SRS* tag of the capabilities file (see Figure 3.1), while the ArcIMS Servers specify the coordinate system in the *FEATURECOORDSYS* and *FILTERCOORDSYS* tags (see Figure 3.2). QGM extracts the coordinate system information from the sources.

If the coordinate system of the source does not match with the QGM's coordinate system, QGM must generate a model of the source in the common coordinate system to ensure that it can determine if a source is relevant to the user query without performing any coordinate conversions. Moreover, it must make sure that any source requests made to the source specifies the required inputs to the source, such as a bounding box, in the source's coordinate system and any data objects retrieved from the source are converted to QGM's common coordinate system. QGM performs this task by declaring a virtual source predicate is identical to the original source except that the values for the bounding box and the vector object attributes are translated to QGM's common coordinate system. QGM defines the virtual source predicate using a domain rule. For example, consider a shapefile data source that provides roads for some area and uses the *EPSG:4269* coordinate system, which is similar to the *EPSG:4326*, but uses the *NAD83* specification for the globe. Below

is a domain rule for that specifies the virtual source predicate for this source and the relationship between the virtual source predicate and the original source:

```
VirtualRoadsShapefile(cbbox, cvectorobj):-
     RoadsShapeSrc(bbox,vectorobj)^
     CoordinateTransformation(ccs, cs, cbbox, bbox)^
     CoordinateTransformation(cs, ccs, vectorobj, cvectorobj)^
     cvectorobj coveredby cbbox^
     cs = 'EPSG:4269'^
     ccs = 'EPSG:4326'
```

The domain rule contains the original source relation (*RoadsShapeSrc*) and two coordinate conversion operations in the body of the rule. The first coordinate conversion operation converts the bounding box attribute (*cbbox*) from QGM's common coordinate system to the source's coordinate system. The second coordinate conversion operation ensures that any data object retrieved from the source is converted into QGM's coordinate system.

Once QGM has defined the virtual source predicate, it generates a Local-As-View description using the techniques described in Section 3.2.1. Describing the virtual source predicate using a Local-As-View rule allows QGM to specify exact constraints describing the contents and the coverage. For the example virtual source, QGM generates the following rule:

```
VirtualRoadsShapefile(bbox,vectorobj):-
```

```
Roads(vtype, format, cs, bbox, source, vectorobj)^

vtype = 'Roads'^

format = 'Shape'^

cs = 'EPSG:4326'^

bbox coveredby '[[33,-117],[35,-119]]'^

vectorobj coveredby bbox^

source = 'RoadsShapeSrc'
```

Note that the Local-As-View rule does not contain any coordinate conversion operations and the coverage restrictions are specified using the common coordinate system. QGM's ability to push all coordinate conversion operations to the domain rules allows it to determine relevant sources to answer a user query without performing any coordinate conversions.

3.3 Automatically Estimating the Quality of Geospatial Data

In addition to automatically generating a source description for new sources, QGM can also automatically estimate the values for some of the quality attributes by analyzing data provided by the new sources. The *Vector* and the *Raster* data sources have different quality attributes associated with them. Therefore, I divide the automatic estimation of quality into two parts. Section 3.3.1 describes automatic estimation of quality of data provided by vector data sources, while Section 3.3.2 describes automatic estimation of quality of data provided by raster data sources.

3.3.1 Estimating Quality Attributes of Vector Data Sources

In this section I discuss QGM's approach of automatically estimating values for quality attributes for vector data sources. QGM can automatically estimate the values of the following attributes: (1) completeness, (2) horizontal accuracy, (3) vertical accuracy, and (4) features within accuracy bounds. Manually populating values for those four attributes would require analyzing a large number of features from a data source and comparing them with information from a source with known quality of data or a high resolution satellite image. Therefore, QGM's ability to automatically estimate values for those attributes saves users from the painful task of manually populating values for those attributes. The values for other attributes such as the date data was collected or the resolution at which the data was collected is often available in text documentation provided by a data source. In future, we can utilize information retrieval techniques to extract values for those attributes from the text documentation.

Automatic estimation of quality attributes for vector data sources in QGM relies on some assumptions: (1) QGM has access to a source of known quality providing the same layer as the new source and with overlapping coverage and (2) the quality of data provided by the new source is uniform, i.e., quality is similar in all areas covered by the source.

QGM begins the quality estimation process by identifying a source with known quality that provides the same data layer and has overlapping coverage with the new source for each data layer provided by the new source. If QGM cannot find a qualifying source, it cannot automatically estimate some of the quality attributes by utilizing any of the known sources. If no qualifying sources exists, QGM adds the source with unknown quality to the

list of available sources with unknown quality of data. After QGM successfully assesses the quality of data provided by any new source, it checks to see if the expanded coverage due to the new source allows QGM to evaluate any new sources. For example, consider that QGM needs to assess the quality of road network data covering the city San Diego provided by some data source. However, the only source with known quality road network data only covers the city of Los Angeles. As there is no overlap, QGM adds the road network source covering the city of San Diego to the list of sources with unknown quality of data. Next, QGM needs to assess the quality of road network data covering entire Southern California region provided by some data source. As the Southern California region overlaps with the city of Los Angeles, QGM can assess the quality of road network data for the source covering the Southern California region. Once QGM finishes assessing the quality of data provided by the source, it checks the list of sources with unknown quality and discovers that it can now assess the quality of the source that provides the road network data for the city of San Diego.

If QGM finds a source with overlapping data, it samples the data from the overlapping coverage from both sources and estimates the values for the quality attributes for the new source. As geospatial data sources often provide a very large amount of data and do not allow querying the entire data source all at once, QGM must sample some data to estimate the quality.

As a part of my research, I evaluated three different patterns to sample geospatial data. Each pattern involves dividing the overlapping area into a grid and retrieving data from some cells in the grid. The size of the grid cells and the number of cells we select depends on the percentage of data we select to sample. As geospatial data may not be

distributed uniformly, it is important to select cells that represent the distribution of the data provided by a source. In my research I selected three different patterns to sample the data.

The first pattern I evaluated was to select all cells in one or more columns or rows in the center of the overlapping area. The rationale behind selecting this pattern was that by selecting all cells in a row or column, we would get cells at the edges as well as cells in the middle. As a geospatial source may have sparse features along the cells toward the edge of the coverage and may have higher feature density in the center, this sampling pattern gives us a more representative sample. Depending on the cell size and the percentage of sample data, QGM selects to sample one or more rows or columns. For example, if each cell covers about 1/100th of the area covered by a source and the user asks QGM to sample 20% data, QGM will select two columns (or rows) in the center of the area as shown in Figure 3.5 (a). The filled areas indicate the selected cells. If QGM needs to sample data from more cells, it will randomly select cells from a third column.

The second pattern was to select all cells in the diagonal pattern. The rationale here was similar to the rationale for the first pattern. However, I expected the diagonal pattern to work better in cases where source may have one or more areas of densely populated features located toward the corners of the coverage areas. Similar to the first pattern the number of cells along the diagonals that QGM samples data from depends on the cell size and the percentage of data QGM needs to sample. For example, if each cell covers about 1/100th of the area covered by a source and the user asks QGM to sample 20% data, QGM will select 20 cells from the two major diagonals as shown in Figure 3.5

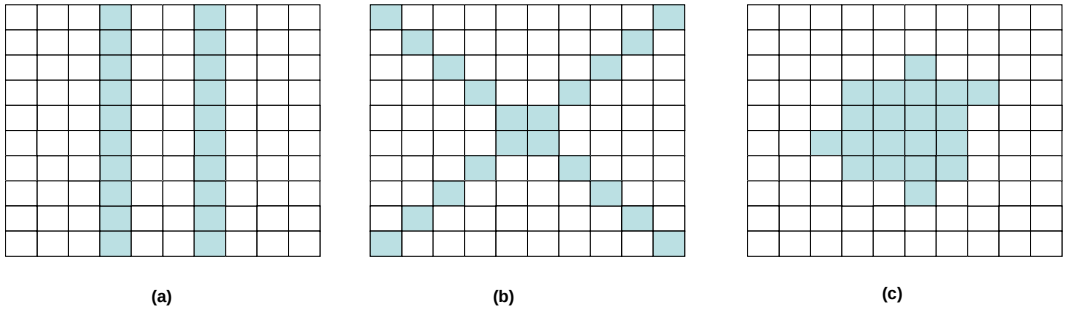

Figure 3.5: Three different sampling patterns for geospatial data

(b). If QGM needs to select more cells, it will randomly select a minor diagonal from neighboring diagonals and randomly select necessary cells from them.

Finally, the third pattern was to select the cells in the center of the grid. The rationale behind selecting this pattern was that we may get maximum number of features by sampling in the center of the area as geospatial source often have high density of features toward the center of the coverage area. Similar to the first two patterns, the number of cells selected to sample depends on the cell size and the percentage of data QGM needs to sample. For example, if each cell covers about 1/100th of the area covered by a source and the user asks QGM to sample 20% data, QGM needs to sample data from 20 cells located at the center of the grid. QGM begins sampling data from the four cells at the center of the grid. Next, it samples data from the twelve neighbor cells of the four center cells. The next set of neighboring cells include 21 cells adjunct to the 16 selected cells. However, QGM can only select four of those cells as it only needs to sample data from 20 cells. Therefore, QGM randomly selects four of the 21 neighbor cells. Figure 3.5 (c) shows an example of cells that would be selected for this example.

Once QGM obtains the sampled data from both the reference source and the new source, it computes the completeness and the accuracy of the new source.

3.3.1.1 Estimating Completeness

In this section, I explain how QGM estimates the completeness of a new source given the sampled data from the new source and a reference source and the completeness of the new source. The completeness of a geospatial source refers to the percentage of existing features that the source provides. For example, if there are three hospitals in a given area and the source provides two of them, it is considered to be 66.7% complete. As QGM does not know about all existing features in the area, it estimates all existing features using a reference source. For example, if the reference source is 50% complete and it provides two features in an area, QGM estimates that there are four features in the area. If a source provides point or polygon data, QGM estimates the completeness for the new source by comparing the number of features provided by the new source with the estimated number of features in the area using the following formula:

$$C_{new} = \frac{\#\text{ of features}_{new}}{\#\text{ of features}_{reference}} * C_{reference}$$

The formula takes into account the fact that the reference source may not be complete. The term $\#offeatures_{new}$ refers to the number of features sampled from the new data source, while the term $\#offeatures_{reference}$ refers to the number of features sampled from the reference data source. The term $C_{reference}$ refers to the completeness of the reference source, while the term C_{new} refers to the completeness of the new source. First,

the formula computes the completeness of the new source compared to the reference source and multiplies that number with the completeness of the reference source to obtain the completeness of the new source.

In case of polylines, the number of features is not a good indicator as different sources may use different granularities to define features. For example, a freeway may be described using many segments or may be approximated using a small number of segments. In both cases the freeway is represented in the source, so the completeness should be similar. Therefore, if a source provides polylines, QGM utilizes a slightly different formula that takes into account the length of the polylines, instead of the number of polylines.

$$C_{new} = \frac{\sum(\text{length of all polylines}_{new})}{\sum(\text{length of all polylines}_{reference})} * C_{reference}$$

As geospatial sources may store a road as a single polyline or a set of line segments, using the number of features does not work well with polyline sources. Therefore, QGM utilizes the sum of the lengths of all features instead of the total number of features.

The completeness estimates generated by QGM depend heavily on the accuracy of the completeness value of the reference set. If the actual value for the completeness attribute of the reference set is lower than the value for the completeness of the reference set provided to QGM, it will underestimate the total number of features. For example, if the actual completeness of the reference set is 25% and the value given to QGM is 50%, QGM will underestimate the total number of features. This will result in QGM computing a higher value of the completeness attribute for the new set. Similarly, if the value of the actual completeness of the reference set is higher than the value provided

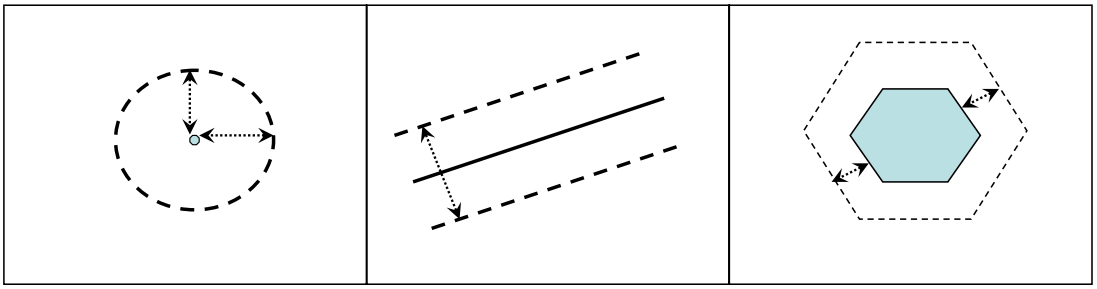

Figure 3.6: Examples of buffers for points, lines, and polygons

to QGM it will compute a lower value of the completeness attribute for the new source. Assuming that we obtain representative samples from both the reference source and the new source, the error in the estimated value of the completeness is double the error in the completeness value for the reference source. Intuitively, this is due to the fact that the error in completeness of the reference source effects the calculation of the error in the completeness of the new source in two ways. First, it effects the total number of features retrieved from the reference source and second, it impacts QGM's estimation of total number of actual features.

3.3.1.2 Estimating Accuracy

In this section, I describe QGM's approach to estimating the accuracy of the data provided by the new source. The accuracy of the data source is measured using three attributes: (1) *horizontalaccuracy*, (2) *verticalaccuracy*, and (3) *vectorsinaccuracybounds*. The first two attributes are used to define a buffer around the actual location of a feature, while the third attribute defines the percentage of roads that fall in the buffer. Figure 3.6 shows examples of buffers for points, polylines, and polygons.

Algorithm 3.3.1: COMPUTEVECTORACCURACY($RefSrc, NewSrc$)

procedure COMPUTEVECTORACCURACY(RefSrc,NewSrc)
$horizacc \leftarrow horizacc(RefSrc)$
$vertacc \leftarrow vertacc(RefSrc)$
$withinaccbnds \leftarrow withinaccbnds(RefSrc)$
$RefSrcFeatures \leftarrow SampleFeatures(RefSrc)$
$NewSrcFeatures \leftarrow SampleFeatures(NewSrc)$
$Buffers \leftarrow CreateBuffer(RefSrcFeatures, horizacc, vertacc)$
if $typeOf(NewSrc) == Points$
then
$newwithinaccbnds = NoOfPointsWithin(Buffers, NewSrcFeatures)$
if $typeOf(NewSrc) == Polylines$
then
$newwithinaccbnds = LengthOfPointsWithin(Buffers, NewSrcFeatures)$
if $typeOf(NewSrc) == Polylines$
then
$newwithinaccbnds = AreaOfPolygonsWithin(Buffers, NewSrcFeatures)$
return $(newwithinaccbnds, horizacc, vertacc)$

Figure 3.7: Algorithm to Compute Accuracy Values for New Source

As QGM does not have access to the actual location of a feature, it utilizes the features from the reference set to approximate the actual location of the feature and computes the values for the *horizontalaccuracy*, *verticalaccuracy*, and *vectorsinaccuracybounds* attributes for the new source using the algorithm shown in Figure 3.7.

First, QGM retrieves the *horizontalaccuracy*, *verticalaccuracy*, and *vectorsinaccuracybounds* attributes for the reference set. Next, QGM utilizes the values of the accuracy attributes to generate a buffer around locations of all features retrieved from the reference set in the sampled data. Next, QGM determines the percentage of features from the sample data retrieved from the new source that fall within the generated buffer. If

the source provides point data, QGM counts the number of points that are within the buffer. If the source provides polylines, QGM computes the total length of all parts of the polylines that are within the buffer. If the source provides polygons, QGM computes the total area of all parts of the polygons that are within the buffer. The computed value is the value for the *vectorswithinaccuracybounds* attribute for the new source, while the values of the *horizontalaccuracy* and the *verticalaccuracy* are the same as the values for the *horizontalaccuracy* and the *verticalaccuracy* attributes for the reference set as those values were used to construct the buffer around the features.

One of the limitations of this approach is that if the reference set is very inaccurate, QGM cannot produce a good estimate of the three quality attributes for the new source. In particular, if we try to estimate the quality of a very accurate dataset using an inaccurate dataset, QGM will only be able to estimate that the accurate dataset is as good as the reference set. This is due to the fact that the values of the vertical and horizontal accuracy bounds will be the same for both sources. Moreover, the value computed for the features within accuracy bounds for the more accurate source depends on the location of the features reported by the inaccurate source as well. It is not possible to provide an error bound for QGM's estimation of accuracy as it depends on a good representative sample from both sources, error in the accuracy values for the reference source, and type of positional inaccuracies in the locations of features provided by the reference source. However, my experiments with real-world data described in Section 3.4 show that the error in estimating the accuracy is less than 10% for sources that provide point, polyline, or polygon data.

3.3.2 Estimating Quality of Raster Datasets

In this section I discuss QGM's process of estimating the quality of raster data sources. For the raster sources QGM can automatically estimate the value of the completeness attribute and learn more accurate coverage of the data sources. This is very important on the Internet as Web Map Servers and ArcIMS Servers often overstate their coverage. This is due to the fact that OpenGIS protocol requires sources to define its coverage of a given data layer using only one bounding box. If a source provides raster data for two disjoint areas, it usually will return a rectangle covering both areas as its coverage. The value for the completeness attribute refers to the percentage of reported coverage area for which the source provides raster data. QGM utilizes the methods described in this section to estimate the area for which the source actually returns the raster data by sampling data from the source.

Figure 3.8 shows QGM's algorithm to estimate the completeness and an accurate coverage for raster sources. Figure 3.9 shows an example of the process of estimating the true coverage for a raster source. QGM first samples the data from the raster sources. In the sampling process QGM divides the area covered by a data source into a uniform grid with cells of the given cell size. QGM sends a source request to obtain an image of 600 pixels width and 600 pixels height at the given resolution in center of each cell. For most data sources and resolution values the image request does not cover entire cell as most sources cover very large areas resulting in large cell sizes. For example TerraServer's reported coverage is a rectangle around the United States. The grid in Figure 3.9 (a)

Algorithm 3.3.2: EstimateRasterQuality($Src, CellSize, resolution$)

```
completeness ← 0
procedure EstimateRasterQuality(Src, CellSize, resolution)
 CellList ← SampleData(Src, CellSize, resolution)
 actualCoverage ← ComputeCoverage(Src, CellList)
 return (actualCoverage, completeness)

procedure SampleData(Src,CellSize)
 CellList ← CreateGrid(Src, CellSize)
 for each cell ∈ CellList
 {cell.Image ← getImage(cell, resolution)
 return (CellList)

procedure ComputeCoverage(Src,CellList)
 srcReportedCoverage ← coverage(Src)
 voronoiDiagram ← CreateVoroNoi(CellList)
 actualCoverage ← coverage(voronoiDiagram)
 completeness ← area(srcReportCoverage)/actualCoverage
 return (actualCoverage)
```

Figure 3.8: Algorithm to Compute Accuracy Values for New Source

shows an example grid and image requests. The areas in gray indicate the areas covered by the image requests.

If the source returns an image QGM stores the image, otherwise it stores label *no image* for that cell. In Figure 3.9 (b), the areas with no image are indicated by white cells. Next, QGM utilizes Fortune's Sweepline algorithm [29] to create a Voronoi diagram representation of all the labeled cells returned from the sampling process. Figure 3.9 (c) shows the example Voronoi diagram. Next, QGM merges the adjunct Voronoi cells containing images into one cell. Finally, QGM computes a minimum bounding rectangle

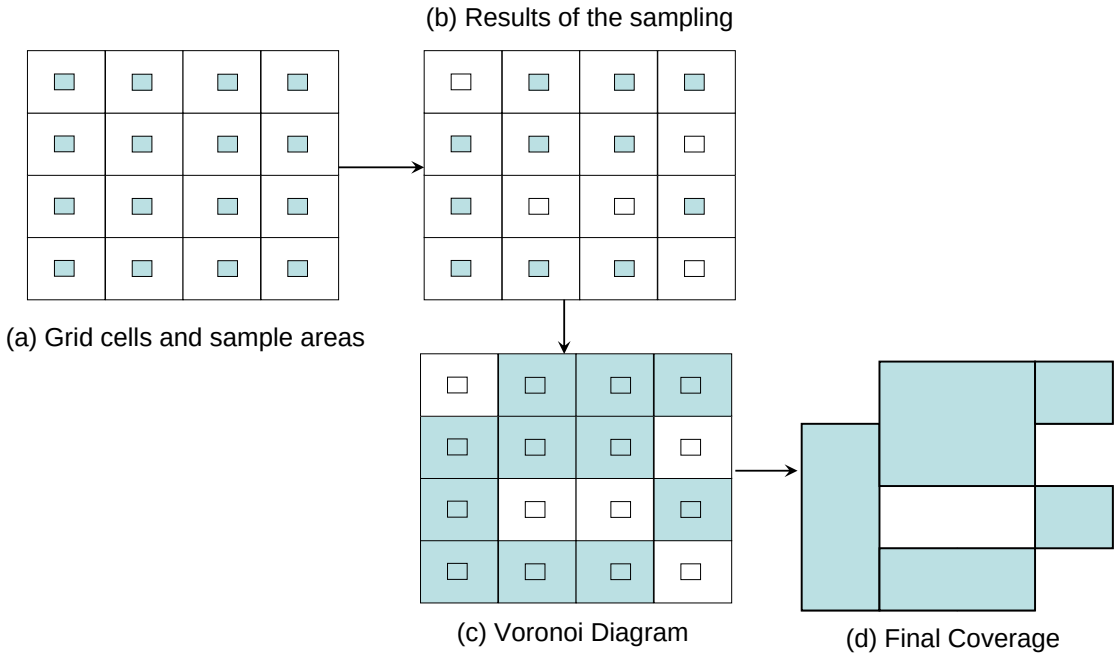

Figure 3.9: Example of coverage estimation process for raster data

around each Voronoi cell with images. The union of the minimum bounding rectangles is the more accurate coverage of the source, while the area of the more accurate coverage divided by the area of the reported coverage is the completeness of the source for the given resolution. Figure 3.9 (d) shows an example of the estimated coverage.

Once QGM has more accurate coverage information, it updates the constraint on the bounding box attribute to specify the more accurate bounding box instead of source's reported bounding box. For example, if a source's report coverage was '[[33,-117],[35,-120]]' and the more accurate coverage is '[[33,-117],[34,-119]]', QGM will update the constraint on the bounding box attribute from *'bbox coveredby '[[33,-117],[35,-120]]"* to *'bbox coveredby '[[33,-117],[34,-119]]"*. If the more accurate coverage contains more than one disjoint rectangles QGM describes the coverage by having one description per polygon.

In our given example if the more accurate coverage was union of the polygons '[[33,-117],[34,-119]]' and '[[34,-119],[35,-120]]', QGM will modify source's description have two similar rules with coverage constraints *'bbox coveredby '[[33,-117],[34,-119]]"* and *'bbox coveredby '[[34,-119],[35,-120]]"*.

In the current implementation, I configured QGM to compute the more accurate coverage and completeness for four different resolutions: (1) 1 meter/pixel, (2) 5 meters/pixel, (3) 10 meters/pixel, and (4) 50 meters/pixel. QGM then computes a source's completeness by averaging the values of completeness computed for each resolution. Moreover, QGM utilizes the union of the coverages computed by all resolutions to update the bounding box constraint.

3.4 Experimental Evaluation

In this section, I describe the empirical evaluation using real-world datasets to support the automatic source description generation and quality estimation techniques described in Sections 3.2 and 3.3.

The performance of the automatic labeling technique can be divided into assigning the correct matching domain concept for a data layer provided by a source and correctly identifying the coverage of the data source and its coordinate system. As the coverage and the coordinate system are provided by a data source, QGM always identifies them correctly. Therefore, I only measure the performance of QGM in identifying the correct matching domain concept for a data layer. I describe the experimental evaluation for the automatic labeling technique in Section 3.4.1. I describe my evaluation of the quality

estimation process in Section 3.4.2. For the quality estimation techniques, I evaluate QGM's performance in estimating the completeness and the accuracy of different sources.

All the experiments involved real-world data sources and were conducted on a server running Windows 2003 Server operating system with dual Xeon processors running at 1.8MHz with 3 GB memory.

3.4.1 Experimental Evaluation of Automated Labeling Technique

In this section, I show that the automatic source description generating techniques generate an accurate representation for real-world geospatial data sources. I used the methods described in Section 3.1 to identify 253 shapefile data sources, 152 Web Map Servers, and 107 ArcIMS servers. All those sources together provide 1248 data layers. I manually matched each layer provided by the sources with a domain concept by looking at the name of the layer, the description of the layer, and the data provided for the layer by the source. While I can identify a lot more sources using the techniques described in Section 3.1, I kept this number small as I had to manually match each layer to a domain concept to identify the ground truth. Table 3.2 shows the top 15 domain concepts and the number of layers assigned to each domain concept using the manual labeling.

Next, I tasked QGM with automatically generating representations of these data sources. I compare the labels generated by QGM with the manual labels to measure the accuracy of the automatic labeling. Table 3.3 outlines the precision, recall, and F-measure values for different domain concepts. Precision, recall, and F-measure are used to measure performance of information retrieval systems, such as search engines. In my case, the term precision refers to the total number of correct matches returned by QGM

Category	Subcategory	Layer	Number of Layers
Transportation	Ground	Roads	361
Imagery	Aerial Photos	Orthophoto	167
Raster			119
Vector			57
Hydrography	Rivers/Lakes	Rivers	53
Transportation	Ground	Bridge/Overpasses	47
Transportation	Tracks	Cart Tracks	46
Transportation	Ground	Ramp Lines	46
Maps	Topo Maps	Topographic Maps	43
Transportation	Air	Airports	36
Buildings	Schools	Schools	31
Boundaries	Political	Administrative Areas	27
Boundaries	Political	Counties	22
Boundaries	Administrative	Flood Zones	18
Boundaries	Administrative	Census Blocks	18

Table 3.2: Major categories with manual labeling

divided by the total number of matches returned by QGM. The term recall refers to the total number of correct matches returned by QGM divided by the total number of possible correct matches. The F-measure is the harmonic mean of the precision and recall values. The formulas below provide equations to compute values for the three terms.

$$Precision = \frac{\#CorrectMatches}{\#TotalMatchesMade}$$

$$Recall = \frac{\#CorrectMatches}{\#PossibleMatches}$$

$$F - Measure = \frac{2 * Precision * Recall}{Precison + Recall}$$

The experimental results indicate that major errors occur in the *Orthophoto*, *Raster*, and *Vector* layers. The errors in the *Raster* and the *Vector* layers are due to QGM not being able to find a suitable match on the layers. I believe this can be fixed by adding

Layer	Precision	Recall	F-measure
Roads	98.02	95.84	96.92
Orthophoto	82.44	64.67	72.48
Raster	70.63	84.87	77.10
Vector	71.62	92.98	80.92
Rivers	94.55	98.11	96.30
Cart Tracks	100.00	97.83	98.90
Bridge/Overpasses	100.00	91.49	95.56
Ramp Lines	95.45	91.30	93.33
Topographic Maps	95.35	95.35	95.35
Airports	97.30	100.00	98.63
Schools	70.73	93.55	80.56
Administrative Areas	71.05	100.00	83.08
Counties	87.50	95.45	91.30
Flood Zones	100.00	100.00	100.00
Census Blocks	84.21	88.89	86.49
Totals	**88.21**	**89.66**	**88.93**

Table 3.3: Results of automatic labeling

new layers to QGM's domain concepts hierarchy. In general, I found that data layers about different weather conditions that have very long name or descriptions, such as 'total precipitation in last fifteen days', caused errors. This was due to the fact that the threshold for matching 0.5 was too high for those layers. We can address this problem by removing stop words from the layer names, by utilizing TF-IDF weighting of different tokens, and by utilizing the actual data in labeling the layers. I discuss the possible improvements in Chapter 6.

The errors in labeling the *Orthophoto* layer were due to the fact that a lot of satellite image layers were labeled 'OrthoImage' and QGM incorrectly labeled them as 'OrthoPhoto'. Similarly, some 'OrthoPhoto' layers were incorrectly labeled 'OrthoImage'. When I labeled the layers, I was able to utilize the metadata to determine if the image was taken by a plane (Orthophoto) or by a satellite (OrthoImage) by looking at the text

description of the metadata. I believe this problem can be addressed in future by developing a specialized information extraction to extract information about the data from the textual metadata.

Overall, I believe that the experimental results show that relatively simple similarity measure, such as the Dice similarity results in 88.93% F-measure in labeling a large sample of real-world datasets.

3.4.2 Experimental Evaluation for Quality Estimation

In this section, I describe the results of estimating the quality of geospatial data sources using QGM. The goal of the experiments is to show that QGM can generate a reasonable estimate of quality for the datasets it discovers on the web. I divide the evaluation into two parts. First I discuss the evaluation of estimating the quality of vector data sources and second I show QGM's performance at estimating completeness of raster data sources.

3.4.2.1 Quality Estimation of Vector Data Sources

In this section I describe the evaluation of QGM's vector data quality estimation. The goal of the experiments is to show that QGM can accurately estimate the quality of vector data sources available on the web. In order to test this hypothesis, I provided QGM with the Navteq rivers, roads, hospitals, schools, lakes, and parks datasets as sources of known quality. I selected the Navteq datasets as I had access to Navteq data covering the entire United States and the Navteq data has high quality.

Next, I asked QGM to estimate quality of 126 shapefiles and 276 data layers from ArcIMS servers. I limited the shapefiles and ArcIMS data layers to the layers containing

rivers, roads, hospitals, schools, lakes, and parks. Moreover, I did not consider the sources that provided data outside of United States as I did not have access to reference datasets outside of the United States.

For all layers, I downloaded all the data and computed the values for the *completeness*, *horizontal accuracy*, *vertical accuracy*, and *vectors within accuracy bounds* attributes for all the new data layers by comparing all of their data with the corresponding data from the Navteq datasets. Next, I use the three sampling techniques described in Section 3.3 to sample the data from the new sources and compute the values for the four attributes using the sampled data.

For each technique, I sampled 5%, 10%, 20%, and 25% data. The values for the *horizontal accuracy* and the *vertical accuracy* attributes only determine the size of the buffer to compute the values for the *vectors within accuracy bounds* attribute. Moverover, the values for the *horizontal accuracy* and *vertical accuracy* do not depend on the sampling technique. Therefore, I compare the values of the *completeness* attribute and the *vectors within accuracy bounds* attribute computed by the complete data sets and sampled datasets.

QGM's performance in point datasets was almost the same for both schools and hospitals datasets. For the polyline datasets, QGM performed slightly better with road network datasets compared to the rivers datasets. However, the difference in performance for all techniques was less than 1.5%. For polygon datasets, QGM's performance was about the same for 8 lake datasets and 4 park datasets. Therefore, I only show results at the level of point, polyline, and polygon datasets.

Type	Sample Size %	# of Layers	Avg. Completeness With 100% Sampling	% Error with Sampling		
				Diag.	Center	Column
Points	5	93	91.76	17.22	18.67	17.86
Points	10	93	91.76	17.54	21.53	15.19
Points	20	93	91.76	14.27	18.85	13.20
Points	25	93	91.76	13.68	16.94	12.04
Polylines	5	297	38.09	30.18	32.42	26.65
Polylines	10	297	38.09	24.69	29.84	24.71
Polylines	20	297	38.09	20.74	29.57	18.20
Polylines	25	297	38.09	19.68	28.58	17.94
Polygons	5	12	68.12	19.54	28.01	24.70
Polygons	10	12	68.12	25.61	28.53	24.19
Polygons	20	12	68.12	24.97	27.75	24.26
Polygons	25	12	68.12	23.68	27.47	23.14

Table 3.4: Completeness estimation using different sampling methods

Table 3.4 shows the results of estimating the completeness of the layers provided by the new sources. The 'Sample Size % ' column shows the percentage of data in the sample set. The 'Avg. Completeness With 100% Sampling' column lists the average value of completeness for the relevant data layers using all the data provided by the source. The last three columns show the average error using different sampling methods. The diagonal sampling and vertical slice (column) sampling perform better than the center cell sampling method. This is mainly due to the fact that in a lot of sources the data is concentrated in the center cells, while the outer cells often contain more missing features. Therefore, the center cells sampling method tends to overestimate the completeness of the sources. The vertical slice and the diagonal methods sample data from cells on the edges as well as cells in the center. Therefore, they get better estimates of the completeness.

Table 3.5 shows the results of estimating the value for the *vectors within accuracy bounds* attribute for the layers provided by the new sources. The 'Avg. Vec. in Bounds With 100% Sampling' column lists the average value of the *vectors within accuracy bounds*

Type	Sample Size%	# of Layers	Avg. Vec. in Bounds With 100% Sampling	% Error with Sampling		
				Diag.	Center	Column
Points	5	93	95.6	12.27	8.69	11.71
Points	10	93	95.6	9.83	8.27	8.96
Points	20	93	95.6	7.95	7.20	7.18
Points	25	93	95.6	7.71	7.14	7.23
Polylines	5	297	80.28	9.8	8.14	8.63
Polylines	10	297	80.28	8.68	7.73	6.81
Polylines	20	297	80.28	8.95	8.50	6.98
Polylines	25	297	80.28	8.67	8.26	6.84
Polygons	5	12	82.19	10.63	10.28	10.53
Polygons	10	12	82.19	9.81	11.36	9.68
Polygons	20	12	82.19	10.12	9.64	9.41
Polygons	25	12	82.19	9.97	9.83	9.43

Table 3.5: Results of Vectors within accuracy bounds estimation

attribute for the relevant data layers using all the data provided by the sources. The last three columns show the average error using different sampling methods. The key observation here is that the center cells sampling method does better with less data compared to the other methods. This is expected as the center area usually contains more feature. Therefore, the center sampling results in larger number of features. In estimating the percentage of vectors that fall within the accuracy bounds the larger sample produces more accurate result. However, in some datasets that cover a big city and its suburbs, such as Los Angeles, the center sampling method tends to overestimate the value for the *vectors within accuracy bounds* as the features in the center of the area tend to have more accurate positioning compared to features at the edge of the coverage.

Overall, QGM's quality estimation can measures the completeness attribute with about 20% error and features within accuracy bounds attribute with about 10% error. Depending on the positional inaccuracies in the location of features provided by the reference source and the new source, QGM may overestimate or underestimate values for

both attributes. While the error seems high, the estimated values are good enough to prune sources that provide low quality data for most user queries. All sampling techniques perform better with points compared to polylines or polygons. This is mainly due to the fact that point datasets usually had fewer features and were more complete.

3.4.2.2 Completeness Estimation of Raster Data Sources

In this section I describe evaluation of QGM's techniques to estimate the completeness of raster data sources. In order to evaluate QGM's performance, I randomly selected 50 data layers provided different raster data sources. The different types of data layers included satellite images, aerial photos, topographic maps, and transportation maps. For each selected data layer, I asked QGM to measure the completeness and an estimate of the true coverage of the source. Computing the actual coverage of raster data source would require making a very large number of requests to the available sources using different resolutions. For example, finding true coverage for TerraServer would require asking it for images covering all areas in entire United States at different resolutions. As I do not have access to true coverage of the data source, I could not compare QGM's estimates to ground truth. Therefore, I used the following method to measure QGM's performance. I randomly selected 15 bounding boxes that intersected with reported coverage of at least one selected data layer. I fixed the image size parameter to 600 pixels by 600 pixels. An image covering any of the fifteen bounding boxes and with the selected image size would have resolution 1 meters per pixel. Similarly, I selected 15 bounding boxes for the resolutions 5 meters/pixel, 10 meters/pixel, and 50 meters/pixel.

For each of the 60 bounding boxes, I asked QGM to first use the reported coverages to select the sources that provide raster data for the area. If the reported coverage of the source intersected with the bounding box, QGM selected the source as providing raster data for the area. Next, I asked QGM to retrieve the raster data from the selected sources for the given bounding box. I counted the number of blank images returned by the sources and the number of images containing data. As long as at least some portion of the image contained data, I counted that image as containing data.

Next, I asked QGM to use the estimated coverages of sources to select the sources that provide raster data for the area. If the estimated coverage of the source intersected with the bounding box, QGM selected the source as providing raster data for the area. Next, I asked QGM to retrieve the raster data from the selected sources. I counted the number of blank images returned by the source and the number of images containing data. Based on the number of image returned by utilizing the reported coverages and the number of images returned by utilizing the estimated coverages I compute the precision and recall values for the estimated coverages and the reported coverages. The precision was computed by dividing total non-empty images returned by total images returned. The recall was computed by total non-empty images returned divided by total possible non-empty images. I computed the total possible non-empty images as the total non-empty images returned based on the reported coverage. This is accurate as all sources overstate their coverage. Table 3.6 shows the results of the evaluation.

The report coverage has 100% recall for all resolutions as it does not miss any images. However, it has lower precision values as it results in many blank images. On an average over 25% images generated by using the reported coverage are empty. This is a very

Resolution	Reported Coverage			Estimated Coverage		
1 meter/pixel	Precision	Recall	F-measure	Precision	Recall	F-measure
1	72.15	100.00	83.82	94.12	84.21	88.89
5	82.43	100.00	90.37	91.38	86.89	89.08
10	81.82	100.00	90.00	92.86	86.67	89.66
15	81.58	100.00	89.86	90.00	87.10	88.52
Total	79.23	100.00	88.41	91.94	86.22	88.99

Table 3.6: Results of Completeness Estimation for Raster Data

significant number as the users not only have to filter out empty images from the results, but a data integration system also spends time to obtain the blank images from the sources. When using the estimated coverages, the recall was little lower as it missed some images due to incorrect estimation of coverages. However, there were very few empty images when utilzing the estimated coverages. Therefore, the precision was much higher.

In addition, the coverage estimation technique also found supported resolutions for sources. For example, the precision when using 1 meters/pixel resolution for reported coverage was low as every query tried to retrieve topographic maps from TerraServer, which provides topographic maps for United States. However, the minimum resolution for the topographic maps from TerraServer is 2 meters/pixel. Therefore all requests returned empty images. The estimated coverage for topographic maps from TerraServer at resolution 1 meter/pixel is empty indicating that the source does not provide images at that resolution. By utilizing the techniques discussed in Section 3.3.2, QGM automatically determines this information and avoids sending extra queries.

Overall, the experimental evaluation suggests that QGM's completeness estimation for raster datasets results in missing some images containing data. However, it results in significant reduction in the number of empty images resulting in higher F-measure score

compared to using the reported coverages. In future we can improve the performance of the system by doing smarter sampling as I discuss in Section 6.

Chapter 4

Query Answering

In this chapter, I describe QGM's process of generating, optimizing, and executing an integration plan in response to user queries. As I mentioned earlier QGM's goal is to efficiently and accurately integrate geospatial data from a large number of sources. Therefore, it is important for QGM to efficiently answer user queries to ensure low response time. Moreover, as QGM may have access to sources of varying quality it is also important to only provide high quality data.

QGM's query answering algorithm is shown in Figure 4.1. I divide the algorithm into three parts. The first part is to utilize the domain model and the query given by the user to generate a plan graph containing requests to all sources and operations pertaining to the user query and quality criteria. QGM generates the plan graph by first inverting the source descriptions using the Inverse rules algorithm [20]. The inversion of the rules is a linear time operation in terms of the number of sources and the number of predicates in the source descriptions. Therefore, QGM can quickly invert a large number of source descriptions. Moreover, as the results of the inversion on depend only on the domain model and not on the user query, QGM can invert the rules once and reuse the results

for future queries. Next, QGM generates a datalog program containing only the relevant rules to the user query by expanding the query rule. This is a very important step in the plan graph generation as it results in selection of only the relevant sources to the user query. For example, in my experiments QGM had access to over 1250 sources of geospatial data and as a result of this step QGM generated a datalog program containing requests to about 63 relevant sources to the user query. The generated datalog program is still not executable as it may not satisfy all the binding restrictions of the sources. Therefore, QGM converts the datalog program to a plan graph and makes changes to satisfy the binding restrictions.

The second step in the query answering process is to optimize the generated plan graph. QGM performs two optimizations. First, it identifies and reuses common subexpressions in the generated graph. Second, it utilizes the quality criteria to prune the source requests that do not satisfy the quality requirements from the generated plan. While the number of source requests QGM can prune depend on the quality criteria, in my experiments on an average QGM pruned just over half of the source requests from the generated plans. So, if a plan contained requests to twelve sources, after pruning based on quality results would result in a plan with six source requests. As geospatial sources often have high response times, removal of half of the sources from the generated plan results in large reduction in the response time. The third step in the query answering algorithm is to translate the generated integration plan into a program that can be executed using a streaming, dataflow-style execution engine called Theseus [5], and execute the plan. The Theseus execution engine streams data between different operations and executes

Algorithm 4.0.1: QGMQueryAnswering($DomainModel, Query$)

```
procedure QGMQueryAnswering(DomainModel,Query)
 PlanGraph ← GeneratePlan(DomainModel, Query)
 OptPlanGraph ← OptimizePlan(PlanGraph)
 Results ← ExecutePlan(OptPlanGraph)
 return (Results)

procedure GeneratePlan(DM,Q)
 DatalogPrg ← InverseRules(DM, Q)
 DatalogPrgWRRs ← IdentifyRelevantRules(DatalogPrg, DM, Q)
 PlanGraphWoBindings ← ConvertToPlanGraph(DatalogPrgWRRs)
 PlanGraph ← HandleBindingRestrictions(PlanGraphWBindings)
 return (PlanGraph)

procedure OptimizePlan(PlanGraph)
 PlanGraphWoCSE ← IdentifyCommonSubExs(PlanGraph)
 QualRes ← EvaluateQualityQuery(PlanGraphWoCSE)
 OptPlanGraph ← PruneByQuality(QualRes, PlanGraphWoCSE)
 return (OptPlanGraph)

procedure ExecutePlan(OptPlanGraph)
 TheseusPlan ← ConvertToTheseusPlan(OptPlanGraph)
 Results ← ExecuteTheseusPlan(TheseusPlan)
 return (Results)
```

Figure 4.1: QGM's Algorithm to Answer Queries

independent operations in parallel resulting in much lower execution time compared to a datalog evaluation engine.

Section 4.1 discusses an example user query and available sources that I use to clarify different algorithms in this chapter. Section 4.2 describes the process of generating a plan graph to answer the user query. Section 4.3 describes the process of optimizing the

generated plan graph. Section 4.4 describes the plan execution. Section 4.5 describes the experimental evaluation of the techniques discussed in this chapter.

4.1 Motivating Example

In order to clearly explain the plan generation and execution process, I describe a scenario where QGM has access to the following data sources:

- VS1: provides road vector data for the area '[[33.5,-117],[34,-118]]'
- VS2: provides road vector data for the area '[[33,-116.5],[33.5,-118]]'
- VS3: provides road vector data for the area '[[33,-117.5],[34,-118]]'
- VS4: provides road vector data for the area '[[33,-117],[34,-118]]'
- VS5: provides road vector data for the area '[[43,-77],[44,-78]]'
- VS6: provides park vector data for the area '[[33,-117],[34,-118]]'
- IS1: provides multi-spectral satellite image for the area '[[33,-116],[34,-118]]'
- IS2: provides B/W satellite image for the area '[[33,-116],[34,-118]]'
- IS3: provides topographic maps for the area '[[33,-116],[34,-118]]'
- IS4: provides multi-spectral satellite image for the area '[[43,-76],[44,-78]]'

Figure 4.2 shows graphical representation of the coverage of all data sources.

I assume that all sources use the 'EPSG:4326' coordinate system. All vector data sources provide data in the shapefile format. I assume that all data sources provide 100%

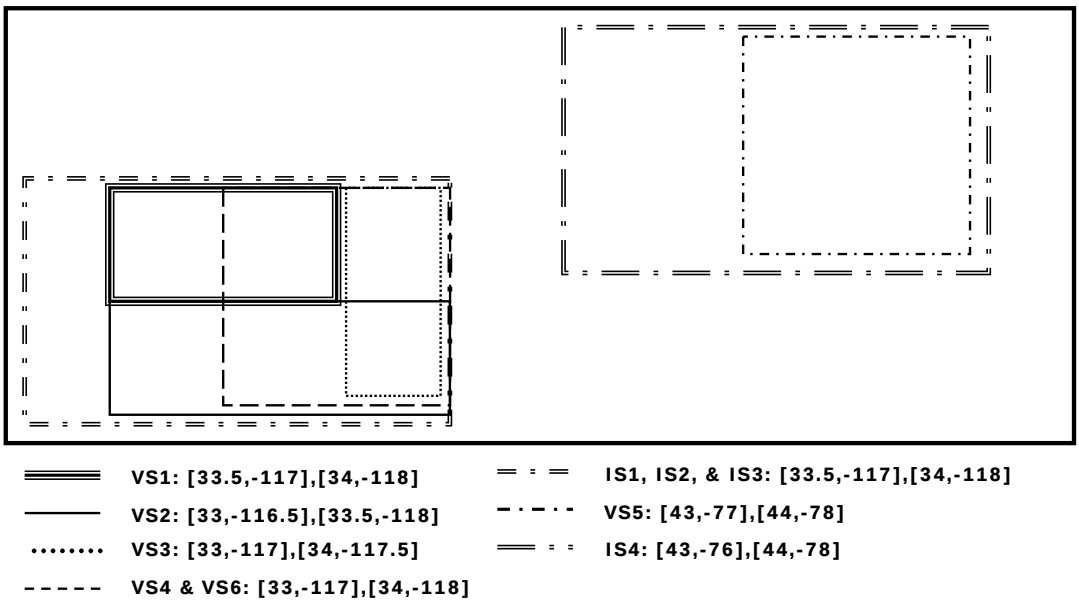

Figure 4.2: Coverage of Available Data Sources

Source	Date	Accuracy (meters)	vectorswithinaccuracybounds (%)	Resolution(m/p)
VS1	1/1/2001	3.6	67	26
VS2	1/1/2001	3.6	82	26
VS3	1/1/2006	3.6	68	35
VS4	2/1/2006	3.6	89	12
VS5	2/1/2006	7.2	56	50
VS6	2/1/2006	10.6	97	100

Table 4.1: Quality Information for Vector Sources

complete data for the area they cover and have 100% attribute completeness. The values for the other attributes of quality are shown in Table 4.1. For the *resolution* attribute, the lower value suggests better quality as the lower values imply that the data was collected at better granularity.

I assume that all image sources provide data in *JPEG* format. Table 4.2 shows the quality of data provided by the image sources. Similar to the vector data sources, I assume that all image sources are complete in this example.

Source	Date	Resolution (m/p)	Multi-spectral
IS1	1/1/2005	0.3	true
IS2	1/1/2001	10	false
IS3	4/1/2003	8	false
IS4	5/12/2001	16	true

Table 4.2: Quality Information for Image Sources

Figure 4.3 shows the source descriptions of the vector data sources *VS1* through *VS6*. The rules describe the sources as views over the *Roads* and *Parks* domain relations. The complete list of all domain relations in the geospatial domain model that I use are in Appendix B. Figure 4.4 shows the descriptions of the four available *Raster* data sources.

In addition to the descriptions of the content of the sources, we also provide QGM with the description of the quality of the data provided by the available sources. The rules that describe the quality of data for the vector data sources are shown in Figure 4.5. The rules for the source quality relations *VS1Quality* through *VS5Quality* are defined as views over the *RoadQuality* relation as those sources provide road network data. The rule for the *VS6ParkQuality* is defined as a view over the *ParkQuality* relation as the source *VS6* provides data about parks.

Similar to the vector data sources, we also provide QGM the descriptions of quality of data provided by the image sources using the rules shown in Figure 4.6. All the source quality relations corresponding to the image sources are defined as a views over the corresponding quality relations defined at the domain level.

Finally, we also provide QGM with the rules that describe the domain concept hierarchy. Figure 4.7 shows the relevant rules for the domain concept hierarchy. The relevant

```
S1:VS1(bbox:b,vectorobj):-
     Roads(type, format, cs, bbox, source, vectorobj)^
     bbox coveredby ‘[[33.5,-117],[34,-118]]’^
     vectorobj coveredby bbox^
     cs = ‘EPSG:4326’^ format = ‘Shape’^
     source = ‘VS1’^ type = ‘Road’
S2:VS2(bbox:b,vectorobj):-
     Roads(type, format, cs, bbox, source, vectorobj)^
     bbox coveredby ‘[[33,-116.5],[33.5,-118]]’^
     vectorobj coveredby bbox^
     cs = ‘EPSG:4326’^ format = ‘Shape’^
     source = ‘VS2’^ type = ‘Road’
S3:VS3(bbox:b,vectorobj):-
     Roads(type, format, cs, bbox, source, vectorobj)^
     bbox coveredby ‘[[33,-117.5],[34,-118]]’^
     vectorobj coveredby bbox^
     cs = ‘EPSG:4326’^ format = ‘Shape’^
     source = ‘VS3’^ type = ‘Road’
S4:VS4(bbox:b,vectorobj):-
     Roads(type, format, cs, bbox, source, vectorobj)^
     bbox coveredby ‘[[33,-117],[34,-118]]’^
     vectorobj coveredby bbox^
     cs = ‘EPSG:4326’^ format = ‘Shape’^
     source = ‘VS4’^ type = ‘Road’
S5:VS5(bbox:b,vectorobj):-
     Roads(type, format, cs, bbox, source, vectorobj)^
     bbox coveredby ‘[[43,-77],[44,-78]]’^
     vectorobj coveredby bbox^
     cs = ‘EPSG:4326’^ format = ‘Shape’^
     source = ‘VS5’^ type = ‘Road’
S6:VS6(bbox:b,vectorobj):-
     Parks(type, format, cs, bbox, source, vectorobj)^
     bbox coveredby ‘[[33,-117],[34,-118]]’^
     vectorobj coveredby bbox^
     cs = ‘EPSG:4326’^ format = ‘Shape’^
     source = ‘VS6’^ type = ‘Park’
```

Figure 4.3: Example Source Descriptions for Vector Data Sources

```
S7:IS1(bbox:b, size:b, resolution:b, imageobj):-
     MultiSpectralImage(type, format, size, resolution, cs,
       bbox, source, imageobj)^
     bbox coveredby '[[33,-116],[34,-118]]'^
     format = 'JPG'^
     source = 'IS1'^
     type = 'MultiSpectralImage'
S8:IS2(bbox:b, size:b, resolution:b, imageobj):-
     BWImage(type, format, size, resolution, cs, bbox,
       source, imageobj)^
     bbox coveredby '[[33,-116],[34,-118]]'^
     format = 'JPG'^
     source = 'IS2'^
     type = 'BWImage'
S9:IS3(bbox:b, size:b, resolution:b, imageobj):-
     TopographicMaps(type, format, size, resolution, cs, bbox,
       source, imageobj)^
     bbox coveredby '[[33,-116],[34,-118]]'^
     format = 'JPG'^
     source = 'IS3'^
     type = 'TopographicMaps'
S10:IS4(bbox:b, size:b, resolution:b, imageobj):-
     MultiSpectralImage(type, format, size, resolution, cs,
       bbox, source, imageobj)^
     bbox coveredby '[[43,-76],[44,-78]]'^
     format = 'JPG'^
     source = 'IS4'^
     type = 'MultiSpectralImage'
```

Figure 4.4: Source Descriptions for Raster Data Sources

rules define the *SatelliteImage* domain concept as a superclass of the *BWImage* and *MultiSpectralImage* concepts. It also defines the *SatelliteImageQuality* domain concept as a superclass of the *BWImageQuality* and *MultiSpectralImageQuality* concepts.

It may seem that writing all the rules and source descriptions is a lot of work. However, the rules to describe the domain concepts hierarchy are only created once and once QGM

```
VS1RoadQuality(resolution, date, horiz-accuracy, vert-accuracy,
               within-acc-bounds, attr-completeness, completeness):-
  RoadQuality(source, type, resolution, date, horiz-accuracy,
     vert-accuracy, within-acc-bounds, attr-completeness, completeness)^
     source = 'VS1'^
     type = 'Road'
VS2RoadQuality(resolution, date, horiz-accuracy, vert-accuracy,
               within-acc-bounds, attr-completeness, completeness):-
  RoadQuality(source, type, resolution, date, horiz-accuracy,
     vert-accuracy, within-acc-bounds, attr-completeness, completeness)^
     source = 'VS2'^
     type = 'Road'
VS3RoadQuality(resolution, date, horiz-accuracy, vert-accuracy,
               within-acc-bounds, attr-completeness, completeness):-
  RoadQuality(source, type, resolution, date, horiz-accuracy,
     vert-accuracy, within-acc-bounds, attr-completeness, completeness)^
     source = 'VS3'^
     type = 'Road'
VS4RoadQuality(resolution, date, horiz-accuracy, vert-accuracy,
               within-acc-bounds, attr-completeness, completeness):-
  RoadQuality(source, type, resolution, date, horiz-accuracy,
     vert-accuracy, within-acc-bounds, attr-completeness, completeness)^
     source = 'VS4'^
     type = 'Road'
VS5RoadQuality(resolution, date, horiz-accuracy, vert-accuracy,
               within-acc-bounds, attr-completeness, completeness):-
  RoadQuality(source, type, resolution, date, horiz-accuracy,
     vert-accuracy, within-acc-bounds, attr-completeness, completeness)^
     source = 'VS5'^
     type = 'Road'
VS6ParkQuality(resolution, date, horiz-accuracy, vert-accuracy,
               within-acc-bounds, attr-completeness, completeness):-
  ParkQuality(source, type, resolution, date, horiz-accuracy,
     vert-accuracy, within-acc-bounds, attr-completeness, completeness)^
     source = 'VS6'^
     type = 'Park'
```

Figure 4.5: Vector Source Quality Relation Definitions

```
IS1MultiSpectralImageQuality(date, resolution,
        multispectral, completeness):-
   MultiSpectralImageQuality(source, type, date, resolution,
        multispectral, completeness)^
     type = ‘MultiSpectralImage’^
     source = ‘IS1’
IS2BWImageQuality(date, resolution,
        multispectral, completeness):-
   BWImageQuality(source, type, date, resolution,
        multispectral, completeness)^
     type = ‘BWImage’^
     source = ‘IS2’
IS3TopoMapsQuality(date, resolution,
        multispectral, completeness):-
   TopoMapsQuality(source, type, date, resolution,
        multispectral, completeness)^
     type = ‘TopographicMaps’^
     source = ‘IS3’
IS4MultiSpectralImageQuality(date, resolution,
        multispectral, completeness):-
   MultiSpectralImageQuality(source, type, date, resolution,
        multispectral, completeness)^
     type = ‘MultiSpectralImage’^
     source = ‘IS4’
```

Figure 4.6: Raster Source Quality Relation Definitions

has source descriptions for some seed sources, it generates the descriptions of other sources automatically. Therefore, users do not have to provide a lot of information to QGM.

Given these sources, QGM is tasked to retrieve the satellite imagery and the road vector data such that both the difference between the dates both datasets are collected and the difference between the resolutions at which both datasets are collected are minimized. Figure 4.8 shows the datalog representation of the query. The datalog representation is divided into three parts: (1) a rule describing the quality criteria, (2) a rule describing the content criteria, and (3) a rule combining the quality and content criteria. The rule

```
SatelliteImage('SatelliteImage' format, size,
               resolution, cs, bbox, source, imageobj):-
  BWImage(type, format, size, resolution, cs,
       bbox, source, imageobj)
SatelliteImage('SatelliteImage', format, size,
               resolution, cs, bbox, source, imageobj):-
  MultiSpectralImage(type, format, size, resolution, cs,
       bbox, source, imageobj)
SatelliteImageQuality(source, 'SatelliteImage', date, resolution,
               multispectral, completeness):-
  MultiSpectralImageQuality(date, resolution, multispectral,
        completeness)
SatelliteImageQuality(source, 'SatelliteImage', date, resolution,
               multispectral, completeness):-
  BWImageQuality(source, type, date, resolution, multispectral,
        completeness)
```

Figure 4.7: Relevant Rules to Describe Domain Concepts Hierarchy

with head *Q1Quality* describes the quality criteria. The rule with the *Q1Data* in the head shows the content criteria. The rule with *Q1* relation shows the join of the quality criteria and the content criteria.

The quality criteria rule contains domain relations describing the quality of data in the body. In addition, it also contains several operations to compute statistics necessary to compute the quality criteria. In the running example, it has a *subtract* operation to compute the difference in the dates on which the the vector data and satellite images were collected for each possible combination of road vector data and satellite image. It also has another *subtract* operation to compute the difference between the resolutions at which the image and road vector data were collected. In addition, it has a *SkylineMin* operation to compute combinations of datasets that are not dominated by other combinations in either resolution difference or date difference.

```
Q1(vectorobj,imageobj,resdiff,datediff):-
  Q1Quality(vsource, vtype, isource, itype, resdiff, datediff)^
  Q1Data( vectorobj, imageobj, vsource, vtype, isource, itype)

Q1Quality(vsource, vtype, isource, itype, resdiff, datediff):-
  SatelliteImageQuality(isource, itype, idate, iresolution,
     multispectral, icompleteness)^
  RoadQuality(vsource, vtype, vresolution, vdate,
     horiz-accuracy, vert-accuracy, withinaccuracybounds,
     attr-completeness, vcompleteness)^
  subtract(iresolution,vresolution, resdiff)^
  subtract(idate,vdate, datediff)^
  pack(resdiff,datediff, aggregateresultrel)^
  SkylineMin(aggregateresultrel,skylineresultrel)^
  unpack(skylineresultrel, skylineresdiff, skylinedatediff)^
  skylinedatediff = datediff ^
  skylineresdiff = resdiff

Q1Data(vectorobj, imageobj, vsource, vtype, isource, itype):-
  Roads(vname, vformat, cs, bbox, vsource, vectorobj)^
  SatelliteImage(iname, iformat, size, resolution, cs,
       bbox, isource, imageobj)^
  bbox = '[[33,-116],[34,-118]]'^
  size = '[600,600]'
```

Figure 4.8: Datalog Representation for Motivating Query

The content criteria rule contains a *Roads* relation and a *SatelliteImage* relation and the necessary coverage constraints to describe the content requirements of the query. The rule with *Q1* relation in the head joins the data retrieved from the content and the quality criteria based on the source and the type attributes for each type of data object. The join between the quality and the data relations is only based on the source and the type attributes as all data objects provided by a given source for a particular type of data have same values associated with all quality attributes.

Based on the descriptions of the sources, it is clear that none of the vector data sources provide road vector data covering entire area. Also, all sources provide data collected on different dates or at different resolutions. Only the first four vector data sources are relevant to the user query as the source *VS5* does not provide vector data for the bounding box described in the query and the source *VS6* does not provide road vector data. Also, only the image sources *IS1* and *IS2* are relevant to the user query out of the four image sources as the source *IS3* does not provide satellite images and the source *IS4* does not provide satellite images for the area of the query.

Table 4.3 shows the difference between the dates and resolutions of possible vector and image source combinations from the relevant vector data and image sources. Based on the resolution and date differences, we can see that QGM should include the combinations of road vector data from the sources *VS1* and *VS2* and the satellite image from the source *IS2* in the answers as these combinations have the minimum date difference (0). Based on the resolution differences in the possible combinations, we can conclude that QGM should also include the combination of the vector data from the source *VS4* and the satellite image from the source *IS2* as that combination has the lowest resolution difference. Finally, QGM should also include the combination of the vector data from the source *VS4* and the satellite image from the source *IS1* as it has lower date difference compared to the combination of *VS4* and *IS2* and it has lower resolution difference compared to all other possible combinations. Therefore, the final answer from QGM should include the following four combinations: (1) vector data from *VS1* and satellite image from *IS2*, (2) vector data from *VS2* and satellite image from *IS2*, (3) vector data

Vector Source	Image Source	Res Diff	Date Diff (Days)
VS1	IS1	36.7	1460
VS1	IS2	16	0
VS2	IS1	25.7	1460
VS2	IS2	16	0
VS3	IS1	34.3	365
VS3	IS2	25	1825
VS4	IS1	11.3	396
VS4	IS2	2	1856

Table 4.3: Possible Answers to User Query and Their Quality

from *VS4* and satellite image from *IS1*, and (4) vector data from *VS4* and satellite image from *IS2*.

4.2 Plan Graph Generation

In this section I describe QGM's process of generating a plan graph to answer the user query. The plan graph generation process is divided into three steps. First, QGM utilizes a technique called the Inverse Rules [20] to obtain the definitions of the domain relations as views over the source relations. As a part of the Inverse Rules process QGM also identifies the relevant rules and generates a datalog program containing relevant inverted rules, relevant domain rules, and the user query. The generated datalog program is not yet executable as it may contain requests to sources without satisfying the binding restrictions. The second step is to convert the datalog program into a plan graph. Finally, in third step QGM modifies the generated graph to ensure that all binding restrictions are satisfied.

4.2.1 Previous Work: Inverse Rules

The Inverse Rules algorithm was utilized by the InfoMaster information integration system [20, 31] to reformulate the user queries. The intuition behind the Inverse Rules algorithm is to obtain the definitions of the domain relations as views over the source relations by inverting the source descriptions. As QGM is expected to integrate a very large number of sources, it is important that the inversion step does not take a long time for each definition. Moreover, the rules generated as a result of the inversion are exactly the same for all queries. Therefore, QGM can reuse the rules generated using the algorithm.

The Inverse Rules algorithm inverts the source descriptions as follows. For every source description, $S(X) : -P_1(X_1), , P_n(X_n)$, where S is a source, P_1 through P_n are domain relations, X and X_i refer to set of attributes in the corresponding source or domain relation, the Inverse Rules algorithm generates n inverse rules, for $i = 1, .., n$, $P_i(X_i) : -S(X)$, where if $X_i \in X$, X_i is the same as X_i else X_i is replaced by a function symbol [20].

For example, consider the source descriptions for the example vector data sources shown in Figure 4.3. QGM utilizes the Inverse Rules algorithm to invert the rules and generates the inverted view definitions shown in Figure 4.9.

The rules *IR1* through *IR6* are the result of inverting the descriptions *S1* through *S6*. Note that QGM keeps all the constraints. All the equality constraints on the attributes of the domain relation are translated to the constants in the head of the rule, while the *coveredby* constraint on the *bbox* attribute remains in the body of the inverted rule. As a

```
IR1:Roads(‘Road’, ‘Shape’, ‘EPSG:4326’, bbox, ‘VS1’, vectorobj):-
     VS1(bbox, vectorobj)^
     bbox coveredby ‘[[33.5,-117],[34,-118]]’^
     vectorobj coveredby bbox
IR2:Roads(‘Road’, ‘Shape’, ‘EPSG:4326’, bbox, ’VS2’, vectorobj):-
     VS2(bbox, vectorobj)^
     bbox coveredby ‘[[33,-116.5],[33.5,-118]]’^
     vectorobj coveredby bbox
IR3:Roads(‘Road’, ‘Shape’, ‘EPSG:4326’, bbox, ‘VS3’, vectorobj):-
     VS3(bbox, vectorobj)^
     bbox coveredby ‘[[33,-117.5],[34,-118]]’^
     vectorobj coveredby bbox
IR4:Roads(‘Road’, ‘Shape’, ‘EPSG:4326’, bbox, ‘VS4’, vectorobj):-
     VS4(bbox, vectorobj)^
     bbox coveredby ‘[[33,-117],[34,-118]]’^
     vectorobj coveredby bbox
IR5:Roads(‘Road’, ‘Shape’, ‘EPSG:4326’, bbox, ‘VS5’, vectorobj):-
     VS5(bbox, vectorobj)^
     bbox coveredby ‘[[43,-77],[44,-78]]’^
     vectorobj coveredby bbox
IR6:Parks(‘Parks’, ‘Shape’, ‘EPSG:4326’, bbox, ‘VS6’, vectorobj):-
     VS6(bbox, vectorobj)^
     bbox coveredby ‘[[33,-117],[34,-118]]’^
     vectorobj coveredby bbox
```

Figure 4.9: Inverted Source Descriptions for Vector Data Sources

result of the inversion, QGM has rules defining the *Roads* and the *Parks* domain relations as views over the source relations.

In addition to the vector data sources, QGM also inverts all source descriptions for the *Raster* data sources. Figure 4.10 shows examples of inverting all source descriptions shown in Figure 4.4. Similar to the source descriptions for the vector data sources, QGM keeps the constraints in the source descriptions in the inverted rules. As a result of the inversion, QGM has rules defining the *MultiSpectralImage*,the *BWImage*, and the *TopographicMaps* domain relations as views over the source relations.

```
IR7:MultiSpectralImage(‘MultiSpectralImage’, ‘JPG’, size, resolution,
        ‘EPSG:4326’, bbox, ‘IS1’, imageobj):-
     IS1(bbox, size, resolution, imageobj)^
     bbox coveredby ’[[33,-116],[34,-118]]’
IR8:BWImage(‘BWImage’, ‘JPG’, size, resolution,
        ‘EPSG:4326’, bbox, ‘IS2’, imageobj):-
     IS2(bbox, size, resolution, imageobj)^
     bbox coveredby ’[[33,-116],[34,-118]]’
IR9:TopographicMaps(‘TopographicMaps’, ‘JPG’, size, resolution,
        ‘EPSG:4326’, bbox, ‘IS3’, imageobj):-
     IS3(bbox, size, resolution, imageobj)^
     bbox coveredby ’[[33,-116],[34,-118]]’
IR10:Multi-spectral(‘MultiSpectralImage’, ‘JPG’, size, resolution,
        ‘EPSG:4326’, bbox, ‘IS4’, imageobj):-
     IS4(bbox, size, resolution, imageobj)^
     bbox coveredby ’[[43,-76],[44,-78]]’
```

Figure 4.10: Inverted Source Descriptions for Raster Data Sources

The quality query passed to QGM is in terms of the domain quality relations. The domain quality relations are virtual. Therefore, the domain quality relations cannot be queried. QGM finds the definitions of the domain quality relations as views over the source quality relations by inverting the descriptions of the source quality relations.

For example, consider the description of the source quality relations shown in Figure 4.5. QGM inverts the descriptions to generate the rules shown in Figure 4.11.

Similar to the vector data quality relation descriptions, QGM also inverts the descriptions of the raster data quality relations shown in Figure 4.6. The inverted definitions for the raster sources are shown in Figure 4.12.

The inverted rules, the domain hierarchy rules, and the user query together form a datalog program to answer the user query. The next step is to analyze the generated

```
RoadQuality('VS1', 'Road', resolution, date, horiz-accuracy,
   vert-accuracy,withinaccuracybounds,attr-completeness,completeness):-
  VS1RoadQuality(resolution, date, horiz-accuracy, vert-accuracy,
   withinaccuracybounds, attr-completeness, completeness)

RoadQuality('VS2', 'Road', resolution, date, horiz-accuracy,
   vert-accuracy,withinaccuracybounds,attr-completeness, completeness):-
  VS2RoadQuality(resolution, date, horiz-accuracy, vert-accuracy,
   withinaccuracybounds, attr-completeness, completeness)

RoadQuality('VS3', 'Road', resolution, date, horiz-accuracy,
   vert-accuracy,withinaccuracybounds,attr-completeness, completeness):-
  VS3RoadQuality(resolution, date, horiz-accuracy, vert-accuracy,
   withinaccuracybounds, attr-completeness, completeness)
RoadQuality('VS4', 'Road', resolution, date, horiz-accuracy,
   vert-accuracy,withinaccuracybounds,attr-completeness, completeness):-
  VS4RoadQuality(resolution, date, horiz-accuracy, vert-accuracy,
   withinaccuracybounds, attr-completeness, completeness)

RoadQuality('VS5', 'Road', resolution, date, horiz-accuracy,
   vert-accuracy,withinaccuracybounds,attr-completeness, completeness):-
  VS5RoadQuality(resolution, date, horiz-accuracy, vert-accuracy,
   withinaccuracybounds, attr-completeness, completeness)

ParkQuality('VS6', 'Park', resolution, date, horiz-accuracy,
   vert-accuracy,withinaccuracybounds,attr-completeness, completeness):-
  VS6ParkQuality(resolution, date, horiz-accuracy, vert-accuracy,
   withinaccuracybounds, attr-completeness, completeness)
```

Figure 4.11: Inverted Descriptions for Vector Source Quality Relations

datalog program to identify relevant rules to answer the query. The Inverse Rules algorithm relies on the datalog evaluation engine to either completely evaluate the datalog program and produce only relevant answers or perform smart evaluation by pruning the rules first. Kambhampati et al. [41] also describe a process to prune the datalog program generated by the Inverse Rules algorithm. QGM's process of identifying relevant rules is

```
MultiSpectralImageQuality(‘IS1’, ‘MultiSpectralImage’, date,
       originalresolution, multispectral, completeness):-
  IS1MultiSpectralImageQuality(date, originalresolution,
       multispectral, completeness)
BWImageQuality(‘IS2’, ‘BWImage’, date,
       originalresolution, multispectral, completeness):-
  IS2BWImageQuality(date, originalresolution,
       multispectral, completeness)
TopoMapsImageQuality(‘IS3’, ‘TopographicMaps’, date,
       originalresolution, multispectral, completeness):-
  IS3TopoMapsQuality(date, originalresolution,
       multispectral, completeness)
MultiSpectralImageQuality(‘IS4’, ‘MultiSpectralImage’, date,
       originalresolution, multispectral, completeness):-
  IS4MultiSpectralImageQuality(date, originalresolution,
       multispectral, completeness)
```

Figure 4.12: Inverted Descriptions for Raster Source Quality Relations

similar to the process described by Kambhampati et al [41]. The key extension in QGM is to utilize the conflicting spatial constraints to prune the generated datalog program.

QGM identifies the relevant rules using the procedure shown in Figure 4.13. QGM begins by expanding the query rule and adding the query rule to the list of relevant rules. For each domain relation in the query rule, QGM finds all rules with the relation in the head of the rule. QGM checks the constraints in the rule with the constraints in the query to find any conflicts. QGM checks the conflicting constraints based on both the order constraints and spatial constraints. QGM adds all selected rules with no conflicts to list of rules that need to be expanded. QGM repeats this process with all selected rules to expand until the list of rules to be expanded is empty. The list of relevant rules contains all datalog rules that are relevant to the user query. Once QGM identifies the relevant

Algorithm 4.2.1: IDENTIFYRELEVANTRULES(R, Q)

```
procedure IdentifyRelevantRules(R,Q)
 RulestoExpand ← Q
 RelevantRules ← φ
 while RulesToExpand ≠ φ
 {CurrentRule ← RulesToExpand.Pop()
 RelevantRules.Insert(CurrentRule)
 ListofRels ← CurrentRule.GetRelsinBody()
 for each Rel ∈ ListofRels
 {if typeOf(Rel) == DOMAIN
   then
 {RulesWithRelinHead ← FindRules(R, Rel)
 for each Rule ∈ RulesWithRelinHead
 {if NoConflicts(Rule, Q)
   then RulestoExpand.insert(Rule)

procedure NoConflicts(Rule,Q)
 RuleConstraints ← Rule.getConstraints()
 QueryConstraints ← Q.getConstraints()
 for each RConstraint ∈ RuleConstraints
 {for each QConstraint ∈ QueryConstraints
 {if Conflicts(RConstraint, QConstraint)
   then return (False)
 return (True)
```

Figure 4.13: QGM's Algorithm to Generate Relevant Rules

rules, it creates a datalog program with those rules and utilizes that as a new program to answer the user query.

In our running example, after expanding the query rule, QGM identifies the rules containing sources *VS1*, *VS2*, *VS3*, *VS4*, *IS1*, and *IS2* and corresponding quality rules as the relevant rules. The rules with the source *VS5* and *IS4* contain conflicting spatial constraints on the *bbox* attribute, while the rules with the source *VS6* and *IS3* are not

relevant as the relations, *Parks* and *TopographicMaps* are not required to answer the user query. Therefore, QGM removes the rules containing source relations *IS3*, *IS4*, *VS5*, and *VS6* from the generated datalog program. Moreover, it also identifies the domain hierarchy rules that define the *SatelliteImage* and *SatelliteImageQuality* domain relations as views over the *BWImage*, *MultiSpectralImage*, *BWImageQuality*, and *MultiSpectralImageQuality* relations as relevant rules. Therefore, the new datalog program generated to answer our example query contains the inverted descriptions of sources *VS1* through *VS4* and the inverted rules of the corresponding quality relations, the inverted descriptions of sources *IS1* and *IS2* and the inverted rules of the corresponding quality relations, the relevant domain hierarchy rules, and the rules describing the user query.

4.2.2 Converting Datalog Program to Plan Graph

In this section I show QGM's process of converting the generated datalog program to a plan graph. QGM utilizes the plan graph representation to ensure binding pattern satisfaction and optimization. The plan graph generation is different compared to the Inverse Rules algorithm which utilizes a datalog evaluation engine to execute the generated datalog program and does not need to convert the datalog program to a plan graph representation.

QGM's algorithm to convert the datalog program to a plan graph representation is shown in Figure 4.14. The generation of the graph representation begins by expanding the query rule. For each source relation in the query rule (if any), QGM inserts a retrieve operation in the graph. For each domain relation in the query rule QGM inserts a project operation in the graph and adds the domain relation to list of relations to be expanded.

Algorithm 4.2.2: GENERATEPLANGRAPHREPRESENTATION($RelevantRules, Q$)

```
procedure GeneratePlanGraphRepresentation(RelevantRules,Q)
 PredicatestoExpand.insert(Q.GetPredicatesinBody())
 ProjectNode ← Project(Q.head)
 Graph ← ProjectNode
 Parents.insert(ProjectNode)
 while PredicatesToExpand ≠ φ
 {CurrentPredicate ← PredicatesToExpand.Pop()
 CurrentParent ← Parents.Pop()
 if typeOf(CurrentPredicate) == SOURCEREL
   then
 Graph.add(Retrieve(CurrentPredicate), CurrentParent)
 if typeOf(CurrentPredicate) == DOMAINREL
   then
 {RulesWithRel ← Findrules(RelevantRules, CurrentPredicate)
 if RulesWithRel.count() > 1
   then {UnionNode ← GenerateUnion(RulesWithRel)
         Graph.add(UnionNode, CurrentParent)
   else Graph.add(ExpandRule(RulesWithRel(0))
 if typeOf(CurrentPredicate) == Constraint
   then
 {SelectNode ← SelectNode(CurrentPredicate)
 Graph.add(SelectNode, CurrentParent)
```

Figure 4.14: QGM's Algorithm to Generate Plan Graph

If two relations have one or more attributes with the same name, QGM inserts a join operation between the operations corresponding to the two relations. For each constraint, QGM inserts a select operation in the graph. QGM continues expanding the query rule until it exhausts all predicates in the query rule(s). The expansion of the predicates in the query is similar to how a datalog evaluation engine executes the datalog program using a top-down evaluation strategy.

Next, QGM extracts the first relation from the list of relations to expand. QGM identifies all rules that contain the selected relation in the head. QGM expands all the identified rules using the same process it used to expand the query rule. When QGM expands the rules, it unifies the attribute names with the attribute names in the domain relation present in the query. If QGM identifies more than one rule containing a domain relation in the head it inserts a union operation in the plan graph. Each child of a union operation represents one datalog rule with the domain relation as its head. QGM repeats this process until the list of relations to expand is empty.

In our running example, first QGM expands the rule for the query. The first predicate it encounters is the *Q1Quality* predicate. As the *Q1Quality* relation appears in the head of one rule, QGM inserts a project operation representing the predicate and adds the predicate to the list of predicates to expand. Next, it encounters the *Q1Data* predicate. QGM follows the same process of inserting a node with a project operation in the graph and adding the *Q1Data* predicate to the list of predicates to expand. In addition, QGM also adds a node representing a join operation between the *Q1Data* and the *Q1Quality* predicates as both predicates contain four attributes with same names (*vsource*, *vtype*, *isource*, *itype*). QGM connects both project nodes with the join node. The resulting graph is shown in Figure 4.15. The triangle shapes in the graph denote the subtrees that QGM will populate next.

Next, QGM expands the rules containing relations *Q1Quality* and *Q1Data* in the head, resulting in addition of more nodes in the graph and predicates to the list of predicates to expand. First it expands the rules for the *Q1Quality* predicate. There is only one rule

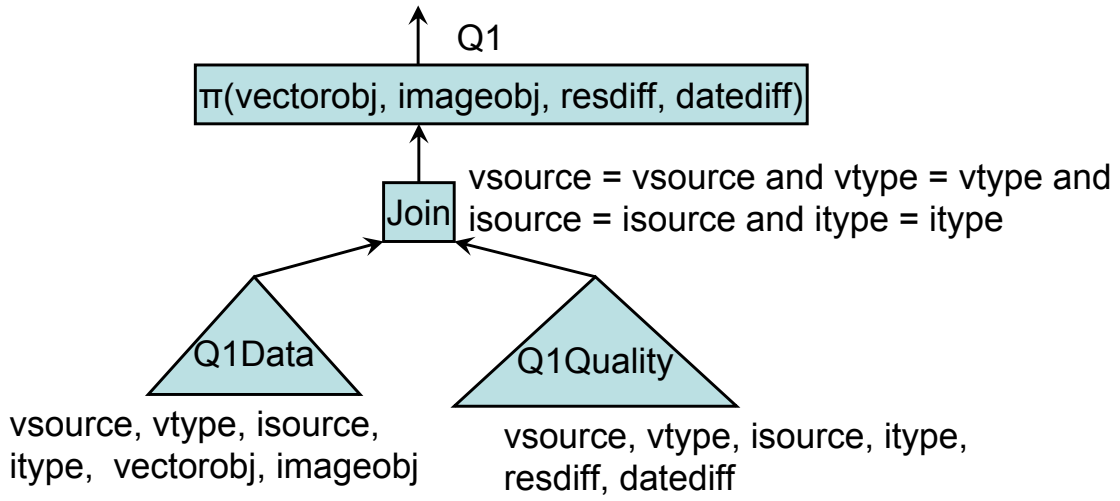

Figure 4.15: Initial Plan Graph

with the *Q1Quality* predicate in the head. This rule is part of the user query shown in Figure 4.8. I also show the rule below for reference.

```
Q1Quality(vsource, vtype, isource, itype, resdiff, datediff):-
  SatelliteImageQuality(isource, itype, idate, iresolution,
     multispectral, icompleteness)^
  RoadQuality(vsource, vtype, vresolution, vdate,
     horiz-accuracy, vert-accuracy, withinaccuracybounds,
     attr-completeness, vcompleteness)^
  subtract(iresolution,vresolution, resdiff)^
  subtract(idate,vdate, datediff)^
  pack(resdiff,datediff, aggregateresultrel)^
  SkylineMin(aggregateresultrel,skylineresultrel)^
  unpack(skylineresultrel, skylineresdiff, skylinedatediff)^
  skylinedatediff = datediff ^
```

```
skylineresdiff = resdiff
```

The rule with the *Q1Quality* relation in the head contains nine predicates. The first two predicates (*SatelliteImageQuality* and *RoadQuality*) are domain relations, so those are added to the list of relations to expand and the corresponding nodes with the project operations are added to the graph. The next five predicates (two *subtract* predicates, *pack*, *SkylineMin*, *unpack*) are relations that refer to different operations. Therefore, QGM adds a node containing retrieve operation corresponding to each call to the operations. QGM also adds necessary join conditions based on the common attributes between different relations. The last two predicates are constraints. QGM adds nodes containing select operations corresponding to each constraint.

Next, QGM expands the *SatelliteImageQuality* domain relation. There are two rules that contain the relation *SatelliteImageQuality* in the head (relevant rules for the domain hierarchy). Therefore, QGM inserts a node containing a union operation. The union operation has two children nodes corresponding to the two rules. Expansion of each rule results in addition of one source predicate to each child node. The graph of the *Q1Quality* branch is shown in Figure 4.16. Note that the plan graph for the *Q1Quality* contains a cross product between the union of the satellite images and union of the roads. The cross product is necessary to ensure that we compute quality of all possible combinations of satellite image and road vector data. However, the result of the cross product is bounded by the number of relevant sources for each type of data.

QGM also expands the subtree for the *Q1Data* predicate. The subtree for the *Q1Data* predicate is shown in Figure 4.17. Note that the subtree for the *Q1Data* contains requests

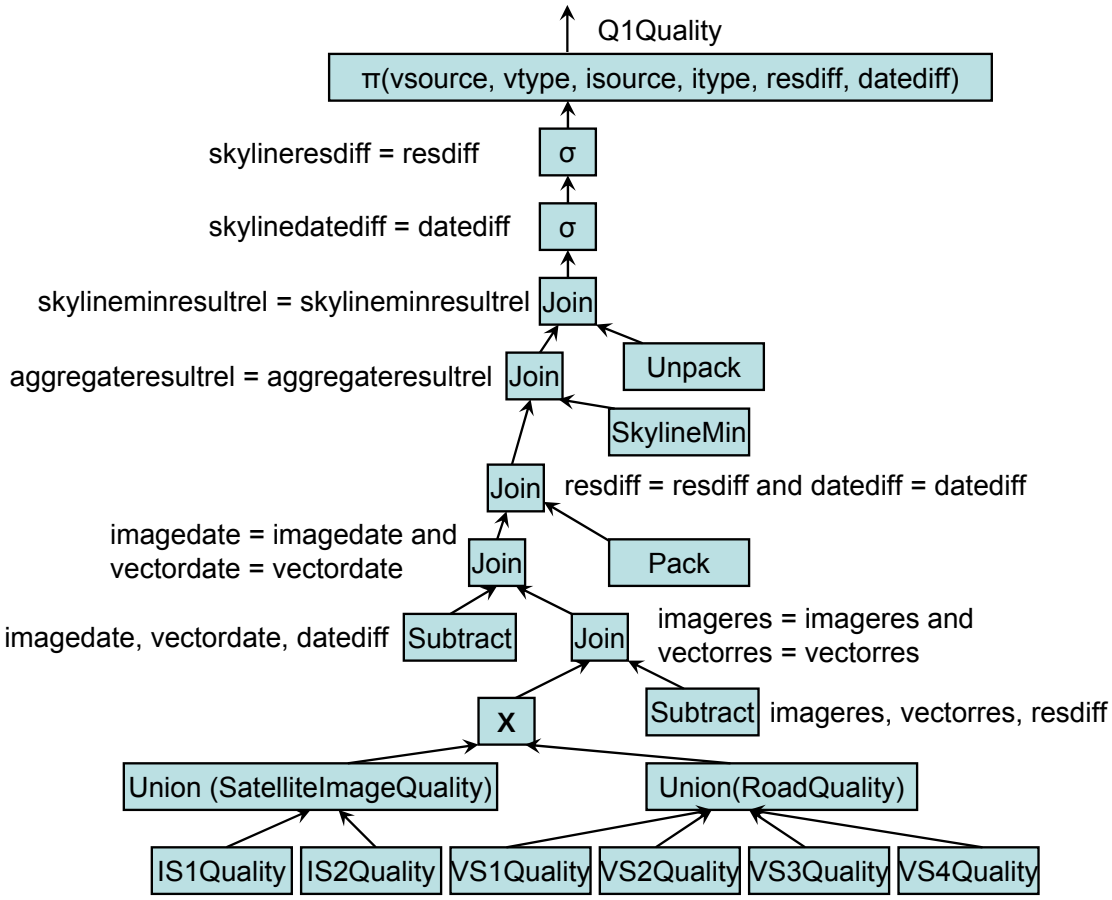

Figure 4.16: Plan Graph for Q1Quality

to relevant vector and raster data sources. However, all vector data sources have binding restrictions on the bounding box attribute. The binding restrictions are not satisfied in this plan graph as the value of the bounding box attribute is not specified. Similarly, the raster sources require the values of both the bounding box and the size of the image attributes.

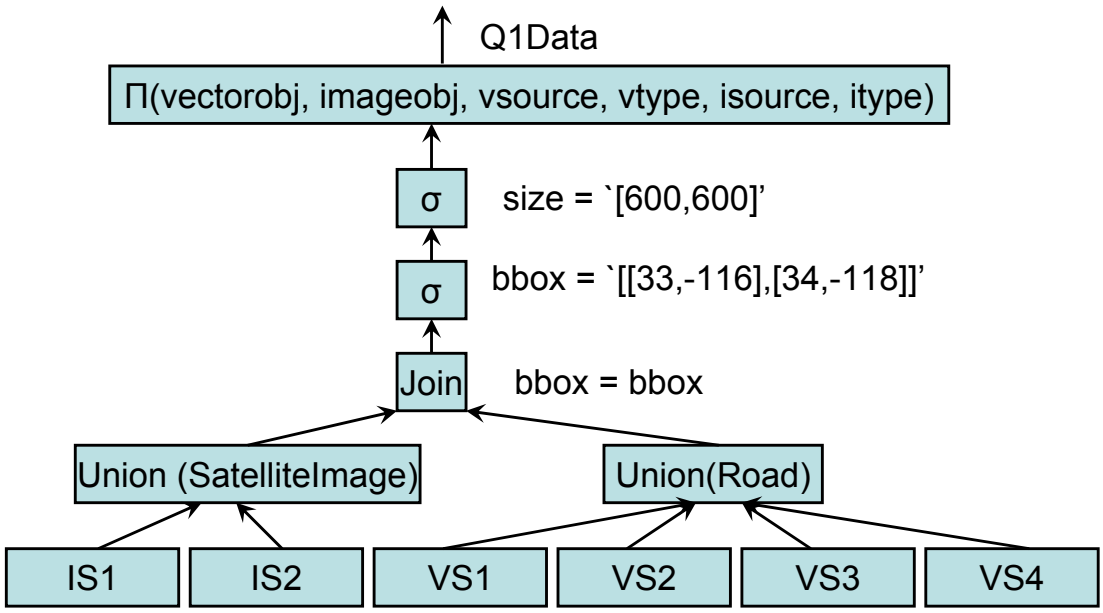

Figure 4.17: Graph Representation for Q1Data

4.2.3 Handling Binding Restrictions

In this section I describe QGM's approach to handling the binding restrictions. Geospatial sources often have required input values, such as a bounding box. In order to obtain data from the sources with binding restrictions, QGM must provide the values for the required input attributes.

The Inverse Rules algorithm [20] addresses the binding restrictions by generating one or more rules called *dom* rules for each attribute with binding restrictions. Intuitively, the idea behind the *dom* rules is to find all possible ways to generate input values for required inputs. Chen Li [48] describes an improvement over this idea by associating specific types to all attributes and generating *dom* rules for each type of required input attribute. However, both of these approaches do not scale in presence of a large number of sources. Therefore, QGM addresses the binding restrictions using a different approach.

In the geospatial domain, the values for the attributes with binding restrictions must come either from the sources relevant to the user query or order constraints in the query. QGM utilizes this fact to address the binding restrictions. For each retrieve operation in the graph, QGM checks to see if there are any binding restrictions on the source from which the retrieve operation obtains the data. If there are no binding restrictions, QGM continues to the next retrieve operation.

When QGM finds a retrieve operation from a source with binding restrictions, QGM first tries to find a select operation(s) containing the attribute(s) with the binding restriction by traversing the graph toward the root from the retrieval operation. If there exists a select operation with equality constraint with a constant value on the restricted attribute, QGM add a constant node in the graph containing the value(s) of the attributes with binding restriction and adds a dependency between the retrieval operation and the constant node.

For the running example, QGM is able to satisfy the binding restrictions of the image sources (*IS1* and *IS2*) and the vector data sources (*VS1*, *VS2*, *VS3*, and *VS4*) using the values provided by different select operations. For example, the source *IS1* has binding restrictions on attributes *bbox* and *size*. There exist select operations equating both attributes with constant values. Therefore, QGM adds a constant node containing values for both attributes and creates a dependency between the retrieval operation for the source *IS1* and the constant node. QGM adds similar constant nodes for all other sources as well.

If QGM does not find a selection on the restricted attribute and the parent of the retrieve operation is a join operation, QGM checks if the other node in the join provides

the restricted attribute. If it does and it is listed as a free attribute, QGM inserts a dependency between two nodes ensuring that the retrieval operation with the binding restrictions would be executed after the other operation and the binding restrictions are satisfied. If QGM cannot satisfy the binding restrictions using the above-mentioned rules, it checks if the attribute with the binding restriction is present in any other retrieve operation. If the attribute is not present, QGM returns the query as unsatisfiable. If the attribute is available, QGM attempts to reorder the join operations in the graph to see if any ordering satisfies the binding restrictions.

For different operations in the graph (*subtract*, *pack*, *SkylineMin*, and *unpack*), QGM adds a dependency between the nodes and their siblings to ensure that the binding restrictions are satisfied. The first part of the graph containing the root node (shown in Figure 4.15) does not change as it contains no retrieve operations. Figure 4.18 shows the modified graph for the *Q1Quality* subtree. The dependencies introduced due to the binding restrictions are shown using more prominent arrows.

Figures 4.19 shows the plan graph representation of the *Q1Data* predicate. The binding restrictions result in the addition of nodes representing constant values for different image and vector data sources.

4.3 Plan Optimization

In this section I discuss two plan optimization techniques implemented in QGM. The first technique is to identify and reuse common sub expressions. Common subexpression analysis [24] is a popular technique in database query optimization literature. The intuition

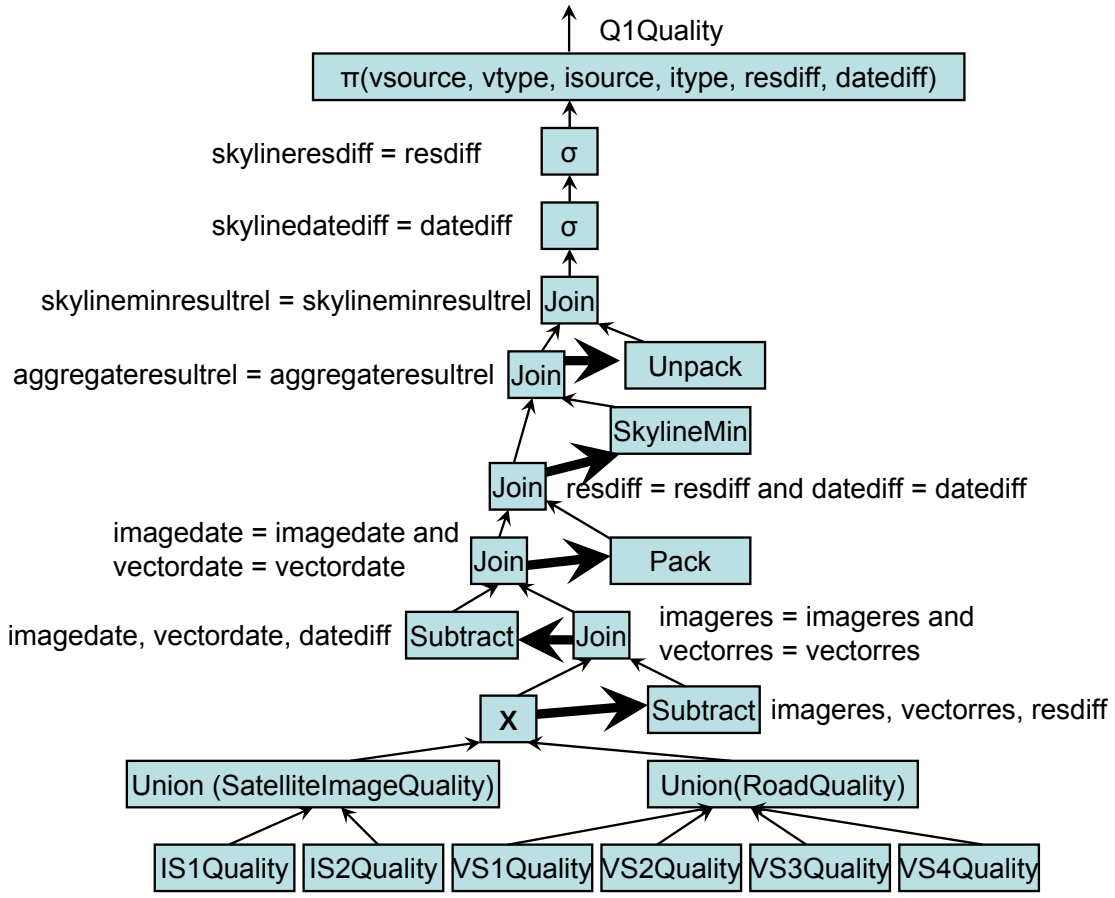

Figure 4.18: Subtree for Q1Quality after Binding Restriction Satisfaction

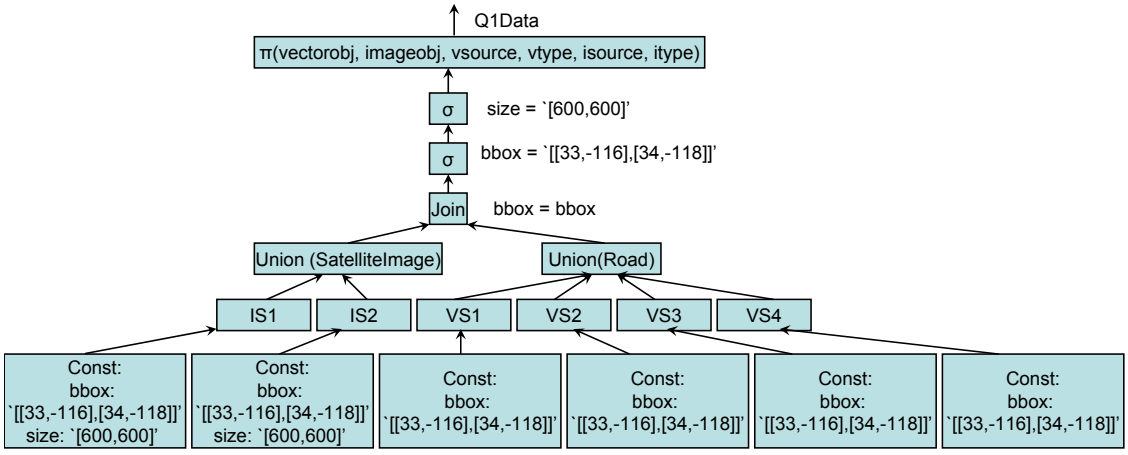

Figure 4.19: Subtree for Q1Data after Binding Restriction Satisfaction

behind the common subexpression analysis technique is to identify the common parts in one or more queries and reuse the results to reduce the response time. QGM implements a common subexpression analysis method to identify the common subtrees in the generated plan graph. The second optimization technique is to execute the quality criteria and and utilize the results to prune the plan graph. As evaluating the quality criteria does not involve retrieving geospatial data from sources, for most queries evaluating the quality criteria is much quicker than executing the entire plan graph. Moreover, for most queries, QGM can prune the plan graph by removing requests to sources that do not satisfy the quality criteria. Therefore, QGM evaluates the quality criteria and utilizes the results to prune the generated plan graph.

4.3.1 Identifying and Reusing Common Sub-expressions

In this section I describe QGM's process of identifying and eliminating common subexpressions. QGM's algorithm to identify common subexpressions is shown in Figure 4.20. As the most expensive operations in the plan graph are the retrieve operations, QGM identifies duplicate subtrees by analyzing all retrieve operations. First, it checks to see if any two retrieve operations contain request to the same source using the same input values. If QGM finds such retrieve operations, it traverses the graph toward the root from both retrieve operations (Call to the TreverseNodes procedure). At each step it checks to see if both subtrees are still the same. When the subtrees differ QGM stops and inserts a rename operation linking the parent of one of the common subtrees to another subtree.

In our running example, QGM does not find any common subexpressions. However, in general it may find some common subexpressions and simplify the plan graph.

Algorithm 4.3.1: IDENTIFYCOMMONSUBEXPRESSIONS(*PlanGraph*)

```
procedure IdentifyCommonSubexpressions(PlanGraph)
 RetrieveNodes ← GetRetrieveNodes(PlanGraph)
 i ← 0
 while i < RetrieveNodes.size()
 {Node ← RetrieveNodes.get(i)
 j ← i + 1
 while j < RetrieveNodes.size()
 {CompareNode ← RetrieveNodes.get(j)
 if CompareNode == Node
   then
 {TraverseNodes(Node, CompareNode, PlanGraph)
 return (PlanGraph)

procedure TraverseNodes(Node1, Node2, PlanGraph)
 Parent1 ← Node1.parent
 Parent2 ← Node2.parent
 if Parent1 == Parent2 and Address(Parent1)! = Address(Parent2)
   then {TraverseNodes(Parent1, Parent2, PlanGraph)
   else {NewNode ← RenameNode(Node1, Node2)
        Replace(Node2, NewNode)
```

Figure 4.20: QGM's Algorithm to Identify Common Subexpressions

4.3.2 Quality-driven Plan Optimization

In this section I discuss QGM's process of evaluating the quality criteria and pruning the graph based on the result. The quality-driven plan optimization process is divided into two steps. First, QGM executes the subtree corresponding to the quality criteria and obtains the results. Second, QGM utilizes the result to remove the requests to sources that do not satisfy the quality criteria from the plan graph.

4.3.2.1 Evaluating the Quality Criteria

In this section I discuss the evaluation of the quality criteria. Evaluation of the subtree corresponding to the quality criteria is performed in two steps. The first step is to retrieve information from the quality relations for each source and the second step is to execute the necessary operations to compute statistics and select qualifying tuples. I explain each step in turn.

The first step in evaluating the quality criteria is retrieving information about the quality of data provided by the sources from the source quality relations. The relevant source quality relations are typically represented as retrieve operations at the leaf level in the quality criteria subtree. In our example, the retrieval from the source quality relations refers to the following six retrieve nodes shown in Figure 4.18: (1) *IS1Quality*,(2) *IS2Quality*, (3) *VS1Quality*, (4) *VS2Quality*, (5) *VS3Quality*, and (6) *VS4Quality*.

The next step in evaluating the quality criteria is to execute operations to compute necessary statistics for the quality criteria. In our running example this step consists of finding all combinations of vector and raster data by performing a cross product between the quality tuples for the road vector data and the quality tuples for the image sources, computing differences in date and resolution for each combination, and using the *SkylineMin* operation to compute the non-dominated combinations of raster and vector data.

In our running example there are $4 * 2 = 8$ possible combinations of raster and vector data. Table 4.3 shows difference in the dates and resolutions for all possible combinations of vector and image data sources. The *SkylineMin* operation results in the selection of

Res. Diff. (meters)	Date Diff. (Days)	Vector Source	Vector Type	Raster Source	Raster Type
16	0	VS1	Road	IS2	SatelliteImage
16	0	VS2	Road	IS2	SatelliteImage
11.3	396	VS4	Road	IS1	SatelliteImage
2	1856	VS4	Road	IS2	SatelliteImage

Table 4.4: Rows Computed for Q1Quality Predicate

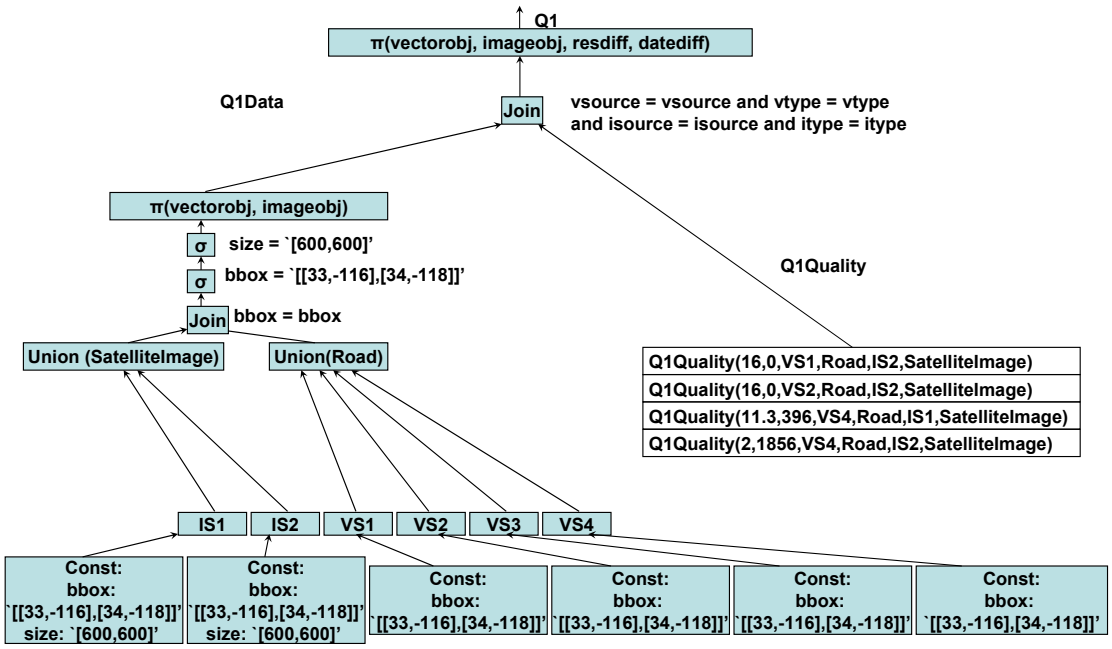

Figure 4.21: Graph After Replacing Q1Quality Subtree

the following combinations of raster and vector data: (1) *VS1* and *IS2*, (2) *VS2* and *IS2*, (3) *VS4* and *IS1*, (4) *VS4* and *IS2*. For each qualifying combination QGM obtains the values of the resolution difference, date difference, a vector data source, a vector data type, a raster data source, and a raster data type. Table 4.4 shows the tuples computed for the *Q1Quality* predicate.

QGM Generates one fact per row for the *Q1Quality* relation and replaces the subtree representing the *Q1Quality* relation with the node representing the facts. The new graph is shown in Figure 4.21.

4.3.2.2 Pruning Based on Quality Criteria Results

In this section I discuss QGM's process of removing unqualified source requests from the plan graph based on the results of the quality query. The intuition behind this optimization is to remove all source requests that do not satisfy the quality criteria by examining all join operations in which the quality criteria participates.

The algorithm to identify source requests that do not satisfy the quality criteria is similar to the algorithm to identify relevant rules (shown in Figure 4.13) in that in both processes QGM checks the constraints in the plan. However, as every node in a graph may participate in multiple branches, QGM must make sure any node it removes will not satisfy constraints in all branches it is connected to.

QGM's algorithm to prune nodes based on quality restriction is shown in Figure 4.22. As the results of the quality query are the only new constant values, QGM only needs to examine constraints that connect to the subtree corresponding to the quality query. Therefore, QGM identifies the branches to prune by examining all join conditions in which the attributes obtained from the quality query participate. In our running example, QGM only examines the join between the *Q1Data* relation and the *Q1Quality* relation.

For each join, QGM identifies the attribute from the quality query that participates in the join condition. QGM already knows the possible values for the attribute from the results of the quality criteria. So, QGM identifies the other child of the join operation and finds all possible values for the attribute involved in the join condition. The list of possible values for any attribute may contain one or more of the following: (1) a value coming from some source or (2) a constant value. If the list of values for the attribute

Algorithm 4.3.2: PRUNEBASEDONQUALITY($G, QualityNode$)

```
Parent ← QualityNode.parent
PrunedG ← G
while Parent ≠ NULL
{ if typeOf(Parent) == JOIN
    then
  { for each Joinattr ∈ Parent.Joinattrs
    { ValuesFromLeft ← FindValues(Joinattr, Parent.leftChild)
      ValuesFromRight ← FindValues(Joinattr, Parent.rightChild)
      for each Value ∈ ValuesFromLeft
      { if Value ∉ ValuesFromRight
          then
        { MarkBranchForDeletion(Value, Joinattr, Parent.rightChild) } } }
  Parent ← Parent.parent }
for each node ∈ PrunedG
{ if node.MarkedForDeletion
    then
  { Remove(node) }
return (PrunedG)
```

Figure 4.22: QGM's Algorithm Prune Nodes Based on Quality Results

participating in the join contains one or more constant values, QGM compares each value to the values coming from the quality query. If the values do not match QGM marks the branch that produces the value for deletion. For all values that match, QGM marks the corresponding branches as useful. After analyzing all join operations, QGM removes all branches marked for deletion.

In our running example, QGM finds that the join conditions for all identified join operations involve function symbols representing the raster and vector data objects. As the result of the quality criteria does not include any combination with the vector data from source *VS3* (see Table 4.4), QGM finds that the branches referring to the source

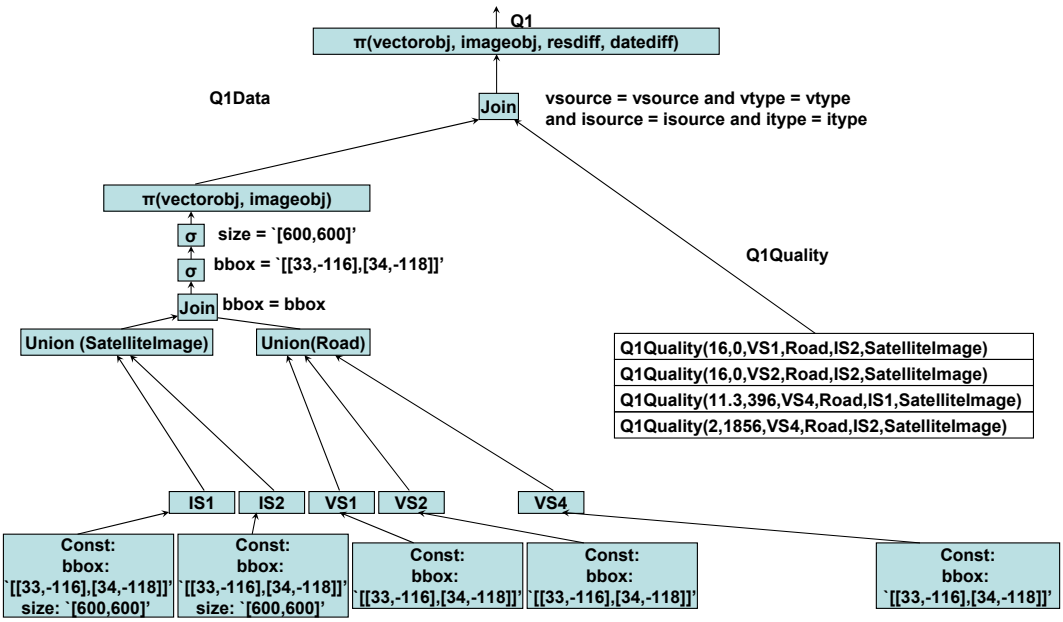

Figure 4.23: Graph After Pruning Based On Quality Results

VS3 results in vector data objects that do not satisfy the quality criteria. Therefore, QGM removes the branches related to the data source *VS3*. The plan graph after the removal of the branches is shown in Figure 4.23.

4.4 Plan Execution

In this section I describe QGM's process of executing the optimized plan graph using the Theseus execution engine [5]. Section 4.4.1 provides brief overview of Theseus' plan language. Section 4.4.2 describes the translation of the plan graph into a Theseus plan.

4.4.1 Brief Overview of Theseus Plan Language

Theseus execution engine is based on a streaming dataflow architecture [5, 16, 38, 40, 53]. The tuples of different relations in a Theseus plan are streamed between the operations

and Theseus executes the independent operations in the plan in parallel. Theseus has variety of operators ranging from data access operators to allow easy access to different types of data sources, such as, databases and web pages, to data management operators, such as select, project, and join. Theseus is also unique in its support for recursion and subplans among the streaming dataflow systems [5, 16, 38, 40, 53].

Most operators in Theseus accept one or more relations as inputs and produce one or more relations as outputs. Below is a list of different Theseus operators that are used in the plans generated by QGM.

Select: The select operator in Theseus performs a relational selection on a given relation of the data. The select operator accepts a relation and a select condition as input and generates a new relation with tuples that satisfy the select condition as output. An example of the select operation in Theseus plan language is shown below:

```
Select(inrelation, 'x = 5' : outrelation)
```

The term *inrelation* refers to the input relation, while '$x = 5$' is the select condition. If the input relation does not have an attribute named 'x', Theseus throws an exception. All rows in the output relation (*outrelation*) have value '5' for the attribute 'x'. The select condition in Theseus supports the following operators: $=, >, <, notlike, like$.

Project:The project operator in Theseus projects a set of attributes from a given relation. The project operator accepts a relation and a list of attributes to project as input and generates a new relation with the projected attributes. An example of the project operation is shown below:

```
Project(inrelation, 'x,y' : outrelation)
```

The term *inrelation* refers to the input relation, while 'x,y' is the list of attributes to project. All rows in the output relation (*outrelation*) have two attributes 'x' and 'y'.

Union: The union operator in Theseus performs a relational union of two relations. The union operation accepts two relations with the same schema and returns a new relation that contains union of the tuples from both input relations. An example of the union operation is shown below:

```
Union(inrelation1, inrelation2 : outrelation)
```

The terms *inrelation1* and *inrelation2* denote the input relations, while the term *outrelation* denotes the output relation.

Join: The join operator in Theseus performs a relation join between two relations. The join operator accepts two relations and a join condition as input and returns one output relation that contains the tuples that satisfy the given join condition. An example of the join operation is shown below:

```
Join(inrelation1, inrelation2, 'l.x=r.x' : outrelation)
```

The terms *inrelation1* and *inrelation2* denote the input relations, while the term 'l.x=r.x' represents the join condition. The prefix 'l.' denotes the first input relation (*inrelation1*), while the prefix 'r.' denotes the second input relation. If the attributes mentioned in the join condition do not exist in the input relations, Theseus throws an exception.

Dbquery: The *Dbquery* operator in Theseus allows querying information from an external database. The DbQuery operation accepts a query, a string containing connection information for the database, a user name, and a password as input and returns a

relation containing the result of the query. An example of the *Dbquery* operation is shown below:

```
Dbquery('select x, y from table1',
          'jdbc:oracle:thin:@mydb!scott!tiger' : outrelation)
```

Apply: The *Apply* operation in Theseus executes an external function for each tuple of the input relation. QGM utilizes the *Apply* operation to retrieve information from different types of geospatial sources using a call to an external function written in Java.[1] Below is an example of a call to an *Apply* operation:

```
apply(inrelation, "MyFunctions.GetData(bbox,'shape1.shp')",
          "shapeobj" : outrelation)
```

The example call makes a request to an external function called 'MyFunctions.GetData'. The external function in the example accepts two arguments. One of the argument is the *bbox* attribute from the *inrelation*, while the other argument is a constant value 'shape1.shp'. The function returns one argument called *shapeobj*. If the *inrelation* contains multiple tuples the *Apply* operation calls the external function multiple times. The output relation contains all attributes from the input relation and the result of the apply operation.

Aggregate: The *Aggregate* operation in Theseus executes an external function once passing values for all tuples of the input relation. QGM utilizes the *Aggregate* operation to implement several aggregation operations including *Sum* and *Average*. Below is an example of a call to an *Aggregate* operation:

[1] http://java.sun.com

```
aggregate(inrelation, "MyFunctions.Average(resolution)", "old_rel",
          "avgresolution" : outrelation)
```

The example call makes a request to an external function called 'MyFunctions.Average'. The external function in the example accepts one argument. In the example the argument is the *resolution* attribute of the input relation. The function averages all values of the attribute and returns the average value. The output relation contains two attributes. One attribute titled *avgresolution* contains the average value and the second attribute called *old_rel* contains the input relation.

Pack: The *Pack* operator in Theseus embeds a relation within another relation. QGM utilizes the *Pack* operator to implement the *pack* operation. Below is an example of the pack operator:

```
pack (inrelation, "packed_rel" : outrelation)
```

The output relation contains one attribute named *packed_rel* with one value that contains the input relation.

Unpack: The *Unpack* operation unpacks a previously packed relation. QGM utilizes this operation to implement the *unpack* operation. Below is an example of the unpack operator:

```
unpack (inrelation, "packed_rel" : outrelation)
```

In the example, the output relation contains the relation that was packed in the *packed_rel* attribute.

4.4.2 Translating Plan Graph to Theseus Plan

In this section, I describe the translation of a graph into a Theseus Plan. I begin the discussion by first describing translation of different types of nodes in the plan graph to one or more Theseus operations. Second, I describe the details of the transformation of entire graph by discussing translation of the plan graph shown in Figure 4.23.

4.4.2.1 Translating Nodes to Theseus Operations

The plan graph generated by QGM contains six types of nodes: (1) retrieval, (2) project, (3) select, (4) join, (5) union, and (6) constant. QGM translates each type of node into a Theseus operation.

Retrieval: Each retrieval node corresponds to a request to obtain data from a source or a call to an operation, such as pack or unpack. QGM translates each retrieval node to a call to a Theseus subplan corresponding to the request to the data source. For example consider the call to the source *VS1Quality* in Figure 4.18. QGM translates the node to the following Theseus operation.

```
VS1Quality(:vs1qualityout)
```

The name of the subplan is same as the name of the source. The term *vs1qualityout* refers to the output relation of the plan. Note that the subplan for the *VS1Quality* does not accept any relations as the source does not have any binding restrictions. If a source has a binding restriction, then the subplan corresponding to the source accepts a relation as input containing values of the attributes with binding restrictions. An example of this is the node representing a call to the source *VS1* in Figure 4.23. Note that the source

VS1 requires a value for the *bbox* attribute. QGM translates the retrieve node to the following Theseus plan.

```
VS1(vs1in:vs1out)
```

The relation *vs1in* has a single attribute named *bbox*. The relation *vs1out* is the output of the subplan.

project: QGM translates each node corresponding a project operation into the project Theseus operation. As an example consider the project node at the root of the plan graph shown in Figure 4.23 that projects the following attributes: (1)*imageobj*, (2) *vectorobj*, (3) *resdiff*, and (4) *datediff*. QGM translates this node to the following Theseus operation.

```
project(Q1in, "imageobj, vectorobj, resdiff, datediff" : Q1)
```

The relation *Q1in* is the input to the project operation and must have values for the four attributes in the project condition. The output of the project operation is the relation *Q1*.

Select: Each select node in the plan graph represents a relational select operation. QGM translates every select node to a select operation. As an example consider the node representing a select operation on the *size* attribute in Figure 4.23. QGM translates the select node to the following Theseus operation.

```
select(selectin, "size = '[600],[600]'" : selectout)
```

Join: A join node in the plan graph represents a relational join between two relations. QGM translates every join node to a join operation. For example, consider the join

operation between the union of the satellite images and the union of the road vector datasets in Figure 4.23. QGM translates the node representing the join operation to the following Theseus operation.

```
join(satelliteimages, roads,  "l.bbox = r.bbox" : joinout)
```

Union:A union node in the plan graph represents union of two or more relations. QGM translates the nodes representing a union to one or more union operations in Theseus. For example consider the node representing union of the satellite image data in Figure 4.23. QGM translate the union node to the following Theseus operation.

```
union(satelliteimagedata1, satelliteimagedata2 : unionout)
```

If a node representing the union operation represents union of more than two relations, QGM translates it to a set of union operations in Theseus. For example, consider the node representing the union of road vector data from three sources. QGM translates the union to the following set of Theseus operations.

```
union(VS1out, VS2out : unionout1)
union(unionout1, VS3out: unionout)
```

Note that the translated Theseus operations first perform union between two relations and then union the result with the third relation.

Constant: The last type of nodes in the graph are constant nodes that are used to provide required inputs to the retrieval operations. QGM translates those nodes into input relations to the Theseus plan. For example, consider the constant node in Figure 4.23 that provides the value for the bounding box attribute to the retrieve operation for

source *VS1*. QGM translates the node to a relation containing one attribute (*bbox*) and one tuple with value '[[33,-116],[34,-118]]'.

4.4.2.2 Translating Relationships Between Nodes in Plan Graph

In addition to translating each node into one or more Theseus operation(s), QGM also needs to make sure that the relationships between the nodes in the plan graph are represented correctly in the generated Theseus plan. QGM ensures those relationships by specifying the input and output relations of different Theseus operations in the generated plan. For example, if one node labeled *nodeA* in the plan graph has incoming edge from another node called *nodeB*, QGM ensures that in the Theseus translation the input relation to the translation of *nodeA* is the same as the output relation of the Theseus translation of *nodeB*. In order to better understand this process, I walk through first few steps of translating the plan graph shown in Figure 4.23 to a Theseus plan.

The root node of the plan graph is a node with a project operation. QGM translates the root node into a project operation. The output of the root node is also the output of the Theseus plan. The input relation to the project operation is the output of the child node connecting to the root node. Below is the generated plan having traversed the root node.

```
PLAN qgmplan{
   INPUT:
   OUTPUT: stream qout
   BODY
   {
```

```
      project(rel0, "vectorobj,imageobj, resdiff,
                     datediff ": qout)
   }
}
```

Note that QGM added the output of the root node (*qout* relation) in the list of outputs. The child of the root node is a node with a join operation. QGM translates this node into a join operation in Theseus. Below is the Theseus plan having traversed first two nodes. The input relations in the join node are the output relations of the children nodes of the join node.

```
PLAN qgmplan{
   INPUT:
   OUTPUT: stream qout
   BODY
   {
      /*join between the Equals, Q1Quality, and Q1Data*/
      join(rel1,rel2,"l.imageobjdata = r.imageobjdata and
            l.vectorobjdata = r.vectorobjdata" : rel0)

      project(rel0, "imageobjdata,vectorobjdata, resdiff,
                     datediff" : outrel)
   }
}
```

The next node that QGM visits is the join node, which is translated into a join operation in the Theseus graph resulting in the Theseus plan shown below.

```
PLAN qgmplan{
   INPUT:
   OUTPUT: stream qualityout
   BODY
   {
      join(rel6, rel7, "l.bbox = r.bbox" : rel5)

      /*join between the Equals, Q1Quality, and Q1Data*/
      join(rel1,rel2,"l.imageobjdata = r.imageobjdata and
            l.vectorobjdata = r.vectorobjdata" : rel0)

      project(rel0, "imageobjdata,vectorobjdata, resdiff,
                     datediff" : outrel)
   }
}
```

QGM continues this process until it exhausts all nodes. The final plan is shown in Appendix B. QGM translates all retrieval operations to a call to subplans for individual operations. Appendix C shows subplans for different types of geospatial sources.

Once QGM has generated the Theseus plan it executes the plan and returns the results to the user. In our running example, QGM executes the plan and provides user with the

four combinations of image and vector data objects that satisfy the quality criteria. The four combinations are: (1) image from IS2 and vector data from VS1, (2) image from IS2 and vector data from VS2, (3) image from IS1 and vector data from VS4, and (4) image from IS2 and vector data from VS4. Those four combinations are the best quality results given the user query and quality criteria.

4.5 Experimental Evaluation

In this section, I describe the experimental results to support the techniques that I described in this chapter. In particular, I evaluate the quality of data provided by QGM in response to the user queries and measure the response time for answering the user queries. I compare performance of QGM to the performance of traditional data integration system with the extensions to support accessing geospatial data. The traditional data integration system that I used did not have the capability to represent the quality of data provided by the sources and did not take quality into account when answering user queries. My experiments tested two hypothesis: (1) as the number of available sources increases, the quality of data provided by QGM is significantly better compared to the traditional data integration system and (2) as QGM makes less source requests compared to the traditional data integration system, the response time of QGM is often lower compared to the traditional data integration system.

I begin this section by first describing the experimental setup. Then, I describe the sets of experiments that I performed and discuss the results.

4.5.1 Experimental Setup

I conducted all experiments on a machine running Windows 2003 server operating system with dual Xeon CPUs operating at 2.1GHz and 3GB memory. However, as each Xeon processor appears as two processors to the operating system and as my code did not take advantage of multiple CPUs, it only ran on 50% of one CPU. The machine was connected to Internet using a Gigabit Ethernet card. The average download speed from a random web page was over 1 mb/sec.

For each set of experiments, I compare QGM with the Prometheus [65] data integration system with extensions to support access to geospatial data. The Prometheus data integration system used in the experiments utilized the Inverse Rules algorithm to generate the datalog program to answer the user query, identified and eliminated common subexpressions, mapped the generated datalog program to a Theseus plan, and executed the generated Theseus plan to produce the results of the user query. The domain model for QGM and Prometheus contained the same sources. However, Prometheus' domain model did not have any quality relations. I utilize the Prometheus system for two reasons: (1) it is a good representative implementation of data integrations system and (2) I had access to the source code.

All my experiments involved real-world sources. The shapefile sources were downloaded and hosted on the same machine to avoid a large number of retrievals from the web during the experiments. The Web Map Server and ArcIMS Server sources were accessed at query time. The domain model for the experiments included 1268 data sources.

All sets of experiments included 10 queries and I report the average response time and the quality of data returned in response to the queries.

4.5.2 Experimental Results

The first set of experiments measure the response time for different queries. The response time depends on two aspects of the query: (1) number of geospatial data layers retrieved in the query and (2) number of sources that provide relevant data for the user query. Therefore, in this set of experiments I measured the response time of QGM and Prometheus for the queries that varied in both above-mentioned aspects.

Response time vs. number of geospatial data layers retrieved: In these experiments, I generated 10 bounding boxes covering about 0.05 degree * 0.05 degree area. I randomly generated bounding boxes until, I found 10 bounding boxes for which the available sources provided between 2-5 different types of geospatial data layers. I repeated the process to find 10 bounding boxes with 5-10 different available data layers and 10-20 data layers. I randomly selected 2 data layers from the available data layers for each bounding box from the first set of bounding boxes. For the second and third sets of bounding boxes I selected 5, and 10 data layers, respectively. For each bounding box in all three sets I asked QGM and Prometheus to find the selected data layers.

I also varied the quality query for QGM. In the first set of experiments the quality query only required QGM to find all geospatial data layers that had either vectors within accuracy bounds or horizontal accuracy better than some pre-determined value. For the vectors within accuracy bounds I used the constant value 50%, while for the horizontal accuracy I chose the constant value 7.0 meters.

Table 4.5 shows the response time of Prometheus and QGM for the first set of queries. I also report the number of results returned by Prometheus and QGM. All timings in the table are in seconds. Note that the number of results returned by each system is all possible combinations of the objects retrieved by the system, i.e. if a system was asked to find a satellite image and a road vector data covering some region and it found 2 satellite images from different sources and 3 road vector data layers, the total number of combinations will be 6. The plan generation time for both systems includes the time to run the Inverse Rules algorithm and generate the graph representation of the plan. The optimization time for Prometheus was the time taken in removing duplicate branches in the graph. For QGM the optimization time includes the time taken in removing duplicate branches, running the quality query, and pruning the generated graph based on the quality results.

While the optimization time for QGM is larger than the optimization time for Prometheus, QGM's execution time is lower compared to Prometheus. As the number of results increase (indicating a larger integration plan), the improvement due to QGM's quality-driven answering becomes more pronounced. However, in queries involving small number of layers, the quality query did not filter out many sources as most sources met the quality criteria. Therefore, while QGM's extra work on ensuring quality of the results did result in less source requests during the execution time, it was not enough to overcome the extra time spent during optimization. Therefore, when retrieving two geospatial data layers QGM was outperformed by Prometheus. Note that this only happens as the quality constraints do not filter out many requests when retrieving two layers. If the quality criteria filtered out more requests QGM would perform much better. Even with quality criteria

# of Layers	Prometheus					QGM				
	Time in Seconds				# results	Time in Seconds				# results
	Gen.	Opt.	Exec.	Total		Gen.	Opt.	Exec.	Total	
2	32	167	284	483	9	32	201	281	497	8
5	34	236	461	731	21	34	237	428	667	16
10	37	298	1190	1525	39	37	312	971	1164	28

Table 4.5: Comparison of Response Time on Simple Quality Query

# of Layers	Prometheus					QGM				
	Time in Seconds				# results	Time in Seconds				# results
	Gen.	Opt.	Exec.	Total		Gen.	Opt.	Exec.	Total	
2	32	167	284	483	9	32	199	131	362	2.7
5	34	236	461	731	21	34	231	244	509	6.3
10	37	298	1190	1525	39	37	316	314	667	12.9

Table 4.6: Comparison of Response Time on Quality Query with Aggregate

that does not filter out many requests, QGM outperforms Prometheus when retrieving five or ten data layers.

Next, I changed the quality query for QGM so that the quality query contained an aggregation operation. I randomly selected either *Min* or *Max* as the aggregate operation. Note that the quality of different geospatial objects retrieved in the queries were still independent of each other, i.e. I did not ask QGM to minimize the difference between quality of geospatial data layers, just asked it to minimize or maximize the value of the selected attribute for each object.

Table 4.6 shows the response time and the number of results returned by QGM for this set of queries. As Prometheus cannot process different quality queries, I show the response time from the first set of queries for comparison. Note that there is a sharp decrease in the response time for all queries executed by QGM. This is because the quality query with aggregate operation prunes most source requests. Due to the reduced number of requests, the execution time of the queries decreases significantly.

# of Layers	Prometheus					QGM				
	Time in Seconds				# results	Time in Seconds				# results
	Gen.	Opt.	Exec.	Total		Gen.	Opt.	Exec.	Total	
2	35	178	394	607	12	35	219	349	603	8.2

Table 4.7: Comparison of Response Time on Quality Query with SkylineMin

In the third set of queries, I changed both the data queries and the quality queries. I asked QGM and Prometheus to retrieve two data layers. I changed the quality query so that QGM had to use the *SkylineMin* operation to minimize the difference between the randomly selected quality attribute (vectors within accuracy bounds or completeness) of the first and the second geospatial data layer. For example, if a query asked to retrieve the road vector data and railroad vector data, QGM had to minimize the difference between the values of the completeness attribute or the vectors within accuracy bounds attribute for the road and the railroad vector data layers.

Table 4.7 shows the results of the queries. Compared to the quality criteria with aggregate operations and order constraints, the quality criteria with Skyline operations take longer to execute. In addition, *SkylineMin* operation resulted in more possible combinations compared to the *Min* or the *Max* operations. Due to this QGM's response time was about the same as Prometheus' response time. However, the number of results returned by QGM were lower than the results returned by Prometheus. Therefore, if a user had to analyze the results and select the best combinations, the extra work done by QGM in pruning results with low-quality would save a lot of user's time.

Response time vs. number of relevant sources: In these experiments, I generated 10 bounding boxes covering about 0.05 degree * 0.05 degree area. I randomly generated bounding boxes until, I found bounding boxes for which there were between

Query	# of Sources	Prometheus					QGM				
		Time in Seconds				# results	Time in Seconds				# results
		Gen.	Opt.	Exec.	Total		Gen.	Opt.	Exec.	Total	
Constraint	0-5	32	98	126	256	3.7	32	109	113	254	3.2
Constraint	5-10	33	119	279	431	7.9	33	116	196	345	5.8
Constraint	10-20	32	131	872	1035	16.1	32	138	524	694	11.2
Constraint	20-30	34	168	1985	2187	24.3	34	159	871	1064	17.6
Aggregate	0-5	32	98	126	256	3.7	32	113	102	247	1.3
Aggregate	5-10	33	119	279	431	7.9	33	116	115	264	2.1
Aggregate	10-20	32	131	872	1035	16.1	32	137	167	336	3.7
Aggregate	20-30	34	168	1985	2187	24.3	34	162	190	386	4.1
Skyline	0-5	32	98	126	256	3.7	32	140	134	306	2.9
Skyline	5-10	33	119	279	431	7.9	33	192	184	409	4.6
Skyline	10-20	32	131	872	1035	16.1	32	297	372	701	7.2
Skyline	20-30	34	168	1985	2187	24.3	34	421	579	1034	9.8

Table 4.8: Comparison of Response Time as Number of Relevant Sources Increase

0-5, 5-10, 10-20, and 20-30 data sources that provided the same type of geospatial data. For example, I selected a bounding box for which the there were between 0-5 sources of road vector data.

I used three different types of quality queries. The first set of quality queries contained a constraint on one quality attribute without any aggregate operations similar to the first set of queries for the previous set of experiments. The second set of quality queries contained an aggregate constraint on one of the quality attributes. The third set of quality queries contained a *SkyLineMin* operation to minimize two quality attributes.

Table 4.8 shows the results of the queries. It is clear that as the number of relevant sources to the user query grows, QGM significantly outperforms Prometheus for all types of queries. This is mainly due to the fact that QGM is able to prune a lot of requests, resulting in much smaller execution time. Even though QGM spends more time optimizing the generated plans, the lower execution time of the optimized plans makes up the time spent in optimization.

Quality of results:

Type	QGM		Average		Std. Deviation	
	% Comp.	% Acc.	% Comp.	% Acc.	% Comp.	% Acc.
Constraint	59.81	87.61	47.71	83.12	17.36	9.31
Aggregate	68.19	89.97	47.71	83.12	17.36	9.31
Skyline	64.03	87.90	47.71	83.12	17.36	9.31

Table 4.9: Quality of Data

In order to show that QGM significantly improves the quality of answers for the user queries, I randomly generated 20 bounding boxes where there was at least one type of geospatial data available. I asked QGM to retrieve one type of geospatial data available in each bounding box using three different types of quality restrictions. The first quality restriction was on either completeness or features within accuracy bounds attribute. It contained a constraint that the selected attribute must be greater than 50. The second quality query asked QGM to find data with the highest value for completeness or features within accuracy bounds. Finally, the third quality query asked QGM a skyline query with the completeness and features within accuracy bounds. I compare the accuracy and completeness of results of QGM with average accuracy and completeness of all objects returned by Prometheus and the standard deviation among the data objects returned by Prometheus.

QGM always returned sources with the best quality given the quality criteria. As shown in Table 4.9, on an average compared to average of all objects returned by Prometheus, QGM provided 12% more complete results for constraint queries, 21% more complete results for aggregate queries, and 16% more complete datasets for skyline queries. The data returned by QGM on average had 5% more features within the accuracy bounds compared to the average of all relevant sources. For all queries QGM's

completeness was one standard deviation better than the average, while for the accuracy QGM's response was almost a half standard deviation better than the average values.

Overall, the experimental evaluation shows that QGM scales much better compared to Prometheus in terms of the response time of the user queries despite spending more time in processing the quality criteria and optimizing the plans by utilizing the quality criteria specified by the user. Moreover, the quality of results returned by QGM is much better compared to the quality of results returned by Prometheus.

Chapter 5

Related Work

In this section, I describe the closely related research to my work. I divide the related research into several categories. The first category of research are papers that focus on geospatial data integration using either data integration or semantic web techniques. The second category of related research is the GIS standards and other web-based mapping applications. The third category of work is on data integration systems and more specifically data integration systems that focus on data quality. The fourth area is the source modeling research that is related to QGM's automatic source description generation process. The fifth area of related research is the research on quality of geospatial data in the GIS community that relates to the representation of quality in QGM. The sixth related research area is the work on query optimization in the database community that relates to some of the optimizations utilized in QGM.

5.1 Geospatial Data Integration

The first area of related research is on integrating geospatial data using data integration or semantic web-based techniques. Hermes [1], MIX [35], VirGIS [10, 11, 12, 13, 22], and

Geongrid [36, 50, 70, 71] are examples of data integration systems that have been used to integrate geospatial data, while the Ontology-driven Geospatial Information System (ODGIS) [26, 27, 28], Geospatial Semantics and Analytics (GSA)[14], and SWING [44, 49, 59] are examples of frameworks that utilize semantic web-based techniques to integrate geospatial data.

The Hermes mediator system [1] integrates multimedia data and spatial data is one type of data it integrates. Therefore, the focus of the work is only on showing that Hermes can integrate spatial data with other datasets. The MIX [35] and VirGIS [10, 11, 12, 13, 22] are XML-based mediator systems that support integration of geospatial data and some grouping and aggregation operations. In MIX, a domain expert can provide list of XML tags related to the spatial data and operations and specify the rules for processing each type of tags. The integration plans in MIX are determined based on Global-As-View rules provided by a domain expert. VirGIS is a mediation system based on OpenGIS standards namely Web Feature Server (WFS) and Geographic Markup Language (GML). The querying and wrapper mechanism in VirGIS are similar to MIX, but it utilizes GML instead of user-specified tags. VirGIS utilizes one to one mapping between source relations and domain relations and can integrate different types of vector and imagery data. QGM's research focus is on quality-driven, large-scale geospatial data integration. Therefore, in addition to the ability to integrate geospatial sources similar to Hermes, MIX, and VirGIS, QGM also automatically generates representation of the content and the quality of geospatial data sources and exploits the quality information to provide high quality geospatial data in response to user queries.

The Geongrid [36, 50, 70, 71][1] describes the grid-enabled mediation services (GEMS) architecture for integrating geospatial data. As a part of the GEMS architecture the authors describe five services for integrating geospatial sources: (1) a registration service, (2) an ontology-based query rewriting service, (3) a spatial results assembly service, (4) a data quality-based rewriting and evaluation service, and (5) map composition service. The registration service is used to automatically collect the metadata for the Oracle spatial sources and ArcIMS services. While in spirit this is similar to QGM's automatic source description generation module, the key difference is that GEMS architecture relies on the sources to utilize standardized names of layers that match with the concepts in the ontology, while QGM utilizes a matching algorithm based on string similarity. GEMS also allows a domain expert to manually add the matching concept in the ontology and quality information for each data layer provided by sources.

The ontology-based query rewriting service in GEMS reformulates the user query specified using the terms in the ontology to a query on the available sources. The spatial results assembly service is responsible to visualizing the results of user queries using a map display. The data quality-based rewriting and evaluation service rewrites the user queries that specify quality criteria. A user can ask GEMS to retrieve all features that are *Definitely* or *Possibly* within the area of interest or within some distance of another feature. The data quality-based rewriting and evaluation service utilizes the accuracy bounds of various data sources to identify qualifying features. The map assembly service in GEMS takes different fragments of raster images provided by different source,

[1] http://www.geongrid.org

the metadata about the coordinates and the resolution of the images, and generates a combined map.

GEMS is similar to QGM in that both are mediator-based architectures. I can improve QGM's query answering by incorporating the research ideas in map assembly service and studying the impact of such assembly on the quality of the resulting image. QGM's representation of quality is more general as it can represent different quality attributes and allows users to specify quality restrictions using datalog rules that are more expressive than utilizing the terms *definitely* or *possibly*. In addition, QGM can automatically match the layers provided by the sources with the concepts in the ontology and automatically estimate the quality of data provided by the sources.

Fonseca et al. [26, 27, 28] describe a framework called ontology-driven geographic information system (ODGIS). The focus of the work is on developing a user interfaces for browsing various classes in the ontology, reasoning based on ontology classes, and utilizing terms from Wordnet to resolve linkage issues between sources. This work assumes that the framework has access to a semantic mediator that generates answers to the user queries by retrieving and integrating geospatial information from the relevant sources. The framework described in this thesis would be an ideal choice to work as a semantic mediator to support the ODGIS framework.

Arpinar et al. [14] describe the process of manually developing a geospatial ontology, automatically and semi-automatically extracting information about entities from sources, and representing entities as instances in such ontology. In addition, they also represent the relationships between the entities as relationships between the instances of the entities. They describe spatial reasoners that can be used to reason about the spatial and temporal

relationships between the instances in the ontology, such as proximity of two instances. The research is implemented in a framework titled Geospatial Semantics and Analytics (GSA). While QGM and GSA are similar in the sense that both provide a uniform interface to geospatial data, the approach taken by both systems is different. QGM relies on automated techniques to generate representation of sources and utilizes a novel quality-driven query answering algorithm to ensure quality of data, while GSA relies on human expertize to carefully collect high quality data. Moreover, the GSA approach is similar to a data warehousing approach in that it relies on populating instances an an ontology by extracting information from sources. Creating a massive data warehouse for all geospatial data available may require a lot of resources. Instead QGM focuses on a mediator-based approach that does not require creating a warehouse with all available data.

Lutz et al. [44, 49, 59] describe their vision of SWING, a semantic framework for geospatial services. The SWING framework shows the feasibility of utilizing an ontology-based reasoner to overcome the semantic heterogeneity in names of layers and attributes in different geospatial services. Their approach is to utilize string similarity to match layers in the user query with the layers in the ontology. The layers in the ontology can be linked to the source using string similarity (automatic) or manually written mappings. The string matching approach used in QGM's automated source generator module can benefit from their extensive work on identifying synonyms in geospatial domain. QGM's automatic description generation and quality estimation modules remove the need to manually write mappings to link sources with the layers in the ontology. The integration in SWING will rely on a geospatial data integration component, which the authors plan to

develop in future. QGM would be an ideal framework to use as the integration component as it can not only integrate geospatial data, but also ensure the quality of data returned to the user.

5.2 GIS Standards and Commercial Mapping Applications

The second area of related work is the various standards for geospatial data and existing mapping systems. The Web Map Server (WMS) and Web Feature Server (WFS) are standards for hosting raster and vector data proposed by OpenGIS. These standards allow programs to dynamically determine which layers are provided by a server and how to access the layers. These standards address the syntactical issues involved in accessing data from various sources. QGM can automatically generate source descriptions for the sources that support these standards. However, the standards do not require the servers to utilize meaningful names for their layers. QGM addresses this problem by using string matching techniques to automatically link name and description of the layers with the mediated schema.

The Federal Geographic Data Committee (FGDC) has provided several documents to define the quality metadata for different types of geospatial data. The FGDC efforts are to ensure that correct metadata is associated with different types of geospatial data available on the Internet. QGM utilizes the FGDC standards for the quality metadata and provides the metadata for the data it produces in response to user queries.

There exist two types of mapping systems. First, the web-based mapping tools, such as Google Maps and MSN Virtual Earth and second desktop GIS applications,

such as ESRI's ArcView. While the web-based mapping systems address the problem of visualizing geospatial data, they do not address the issues relating to querying the data and the accuracy of the integrated data. Nevertheless, QGM can use these tools to visualize the result of different queries.

ESRI[2] software's ArcInfo and ERDAS Imagine[3] are very commonly used GIS systems. The commercial GIS systems are great for visualizing different data layers and often allow developers to encode various operations, such as, conflation as plug-ins. However, they provide limited dynamic integration capabilities. The focus of our work is on dynamically selecting relevant sources, retrieving data from those sources, and integrating the retrieved data using relevant operations. The results of the queries to QGM can be visualized using any GIS system that supports OpenGIS standards.

5.3 Data Integration Systems

The third category of related research is work on data integration systems, such as, the Information Manifold [47], InfoMaster [31], InfoSleuth [6], and Ariadne [45]. While some of these systems are used to integrate images or semi-structured data related to geospatial entities, none of these systems supported different geospatial data types, formats, or operations. One can extend any of the above-mentioned systems to develop a framework to integrate geospatial data. However, the required extensions would be similar to various features of QGM. The key advantages of QGM are support for automatic source description generation, automatic estimation of quality of data provided by sources, support for

[2]http://www.esri.com
[3]http://gis.leica-geosystems.com/

geospatial operations, such as coordinate and format conversion operations, and support for combination of GAV and LAV rules.

The data integration community also contains research on quality-driven data integration [7, 8, 21, 51, 54, 55, 52, 61]. Berti-Equille [7], Bleiholder et al. [8] , Eckman et al. [21], and Mihaila et al. [51] describe approaches to ensure maximally complete answers for queries on life science data sources by analyzing all possible plans to compute data. The goal of their work is to ensure highest quality data by providing maximally complete data by evaluating all possible plans to integrate data. QGM's representation of quality is more expressive as it can represent multiple attributes of quality including completeness. Moreover, QGM also allows a user-specified quality criteria.

Naumann et al.[54, 55] describe a quality-driven integration system that integrates biological data. The authors describe a mediator system that utilizes Local-As-View rules and associates quality with each source similar to QGM. Naumann et al.'s mediator system divides the quality attributes into three categories: (1) source specific criteria, (2) content specific criteria, and (3) attribute specific criteria. Given a user query, their system first utilizes the Data Envelopement Analysis (DEA) Method to select the sources that provide high quality data based on the source specific quality criteria. The DEA method is similar to the Skyline queries in that given points in two or more dimensions, it selects all points on the frontier (i.e. all points not dominated in at least one dimension). Next, Naumann et al.'s framework utilizes the query correspondence assertions (QCA) to generate all possible plans that answer the user query using only the sources selected in the first step. A QCA is similar to a Local-As-View source description in that it defines the relationship between the sources and mediated schema. Finally, it computes

a quality score for each plan by aggregating the quality of all sources and operations used in the plan and ranks the plans based on the user supplied weights for different quality attributes.

QGM differs from the framework described in Naumann et al. in several ways. First, QGM also contains modules to automatically generate source descriptions for geospatial sources and estimate the quality of their data. Second, QGM's quality criteria specification is more expressive as it allows users to use different operations, such as min, max, or skyline, and constraints. Some of the quality criteria, such as minimizing both the difference between the resolutions at which two datasets were collected and the dates on which two datasets were collected cannot be specified using Naumann et al.'s framework. Moreover, even simpler quality criteria such as finding all vector datasets with completeness above 50% are hard to specify using different weights for the quality attributes specified in Naumann et al.'s framework. Third, QGM's approach to query answering explores the plan space by first generating a plan containing all possibly relevant sources based on the content and then removing sources based on the quality criteria. As geospatial datasets often have limited coverage, Naumann et al.'s approach of pruning first based on only the source specific quality attributes may result in pruning all sources that provide data for the area. Therefore, QGM's query answering algorithm first selects only the relevant sources based on the content and then explores the search space based on the quality requirements. As the number of sources that provide data for the layers and areas in the query tend to be small in general, the search space is relatively small. Fourth, in addition to ranking the plans, QGM executes the generated plans efficiently using a streaming, dataflow-style execution engine to reduce the response time of user queries.

Scannapieco et. al [52, 61] describe a Global-As-view information integration system called DaQuinCIS that exploits the quality information about the data provided by the sources, availability of sources, and their response times to provide high quality answers to the user query in a peer-to-peer environment. The DaQuinCIS architecture represents the content and the quality of each source using Global-As-View rules. The data quality broker module in DaQuinCIS computes the plan to answer the user query by unfolding the user query and the Global-As-View rules, executes the generated plans, and evaluates the quality of data produced by each plan. The data quality broker only returns the high quality data to the users. In addition to quality-driven query answering, QGM also supports automatic source description generation and automatic quality estimation. As QGM utilizes the Local-As-View rules, addition of new sources is much easier compared to DaQuinCIS. Moreover, in the geospatial domain executing all generated plans is not feasible as it may take a long time to execute all generated plans and execution may result in a huge amount of data.

5.4 Source Modeling

The problem of source modeling aims at understanding what a new source does in terms of existing sources [15, 58]. The source modeling techniques are also directly relevant to the automatic source description generation module of QGM. However, QGM's problem is simpler as it only generates the descriptions for sources that adhere to well-known formats or standards. Therefore, QGM already has some information about the content and coverage of the sources. One problem with the existing Web Map Servers is that they

often overstate their coverage. The reason behind this is that the OpenGIS standard only allows sources to specify a bounding box for each layer. So, if a source provides data for all major cities in United States, it ends up specifying entire United States as its coverage. However, the true coverage of the source is union of all metro areas. As a result answers to user queries that require images in rural areas often come back with empty images. While QGM's coverage estimation eliminates most empty images, it also results in some incorrect coverage information. We can improve QGM's coverage estimation techniques by utilizing the techniques described by Mark Carman [15] to learn more accurate coverage restrictions for geospatial sources. The key idea in Mark Carman's work is to learn more accurate class boundaries by sampling data at the class boundaries. We can use this idea in QGM to sample more data at the class boundaries and improve the coverage information.

5.5 Visualizing and Improving Quality in the Geospatial Domain

GIS researchers have worked on different approaches and ontologies to model and visualize accuracy in geospatial data [32, 33, 68, 69]. QGM allows users to model various concepts and characteristics, such as completeness or alignment, described in [68, 69] as quality attributes and pose queries to retrieve geospatial data that satisfies the accuracy requirements based on the given characteristics. As the focus of my research is not on visualizing geospatial datasets, my work does not address the issue of visualizing quality

of geospatial data returned by QGM. However, as QGM returns the quality information, we can visualize the quality of results using the approaches described in [32, 33].

In addition, there has also been work on improving the quality of raster or vector data by aligning the data with more accurate datasets. For example, the quality of road network data can be improved by aligning the data with a ortho-rectified satellite image. The process of aligning one geographic dataset with another dataset is call conflation [60]. The process of conflation changes the quality of geographic data as it may change the actual location of geographic features within the dataset. Commercial GIS systems, such as ESRI's ArcInfo, ERDAS Imagine, allow users to manually conflate two datasets by providing a set of control points between two datasets. Ching-Chien Chen [17] describes a method to automatically generate control points by extracting features from geospatial datasets thereby automating the conflation process. Conflation operations and their effects on the quality of geospatial data can be represented in QGM. The addition of conflation operation would allow QGM to provide more accurate answers to the user queries.

5.6 Database Query Optimization

The sixth category of research is on reducing the response time of integration plans by optimizing the plans [3, 19, 39, 41, 43, 66]. Kifer and Lozinskii [43] describe an approach to 'push' selections and projections close to sources in deductive databases. This work is relevant to our database-style optimizations. It may also be possible to add other similar database-style optimizations to QGM. However, focus of my research is to show that it is

feasible to accurately and efficiently integrate a large number of geospatial sources using a quality-driven data integration system. Therefore, I focus more on improving the quality of data produced by the mediator framework.

Kambhampati et al. [41] describe strategies to optimize the recursive and non-recursive datalog programs generated by the Inverse Rules algorithm. The research focus of their work is to remove redundant data accesses and to order access to different sources to reduce the query execution time in the presence of overlapping data sources. In our framework, we have implemented several optimizations similar to those described by Kambhampati et al.

Ashish et al. [3] described the idea of using sensing operations to optimize data integration plans. We have also worked on tuple-level filtering algorithm [63, 64, 65] technique that generalizes the idea of introducing sensing operations to reduce the number of source requests. In future, we can add this type of functionality in QGM to further improve the response time of user queries.

There are many techniques in the literature to optimize datalog programs [66], such as magic sets [4]. One can easily imagine an extension to our system that optimizes the generated datalog program using these techniques.

In the database community, there has been much research on parametric query optimization [19, 39]. The key focus of those algorithms is to determine order of the execution of different operations in the query plan. The focus of our paper is on representing the content and quality of geospatial sources and providing the best-quality results for the user queries by exploiting the quality information. In future, we may be able to utilize

some of the database query optimization techniques to reduce the optimization time in QGM.

Chapter 6

Discussion and Future Work

In this chapter I first summarize the contributions of this thesis. Next, I discuss several application areas that would benefit from utilizing my research. Finally, I list possible improvements for the techniques discussed in this thesis.

6.1 Contributions

The key contribution of this thesis is a Quality-driven Geospatial Mediator (QGM) framework to accurately and efficiently integrate geospatial sources and operations. I believe QGM can fulfill the need for a large-scale geospatial data integration framework and support a variety of application areas. Apart from the applications, I believe my research provides a representation methodology to declaratively specify and utilize the quality of data provided by sources in a data integration framework. The work presented in thesis can also be utilized to automatically obtain information about the content and the quality of a large number of geospatial data sources available on the web. The discovered information can be utilized to create a virtual or materialized warehouse of geospatial data to support disaster rescue or urban planning applications. As QGM's quality-driven

query answering algorithm ensures that all data returned to the user meets the quality criteria, domain experts can add new sources to the warehouse without worrying about impacting the quality of results returned to the user.

6.2 Application Areas

I believe that my research can provide significant improvement for geospatial data integration applications in several areas, such as urban planning and disaster response. In particular, QGM can be used to provide high quality geospatial data for urban planning or disaster response applications. The administrators of the applications could provide QGM with the locations of various relevant repositories of geospatial sources and some seed data. QGM can then automatically determine the content and the quality of data that the sources in the repository provide and utilize that information to answer queries that require retrieving different types of geospatial data from the sources.

6.2.1 Urban Planning Applications

An example of an urban planning application is the Green Visions Plan[1]. The goal of this project is to allow access to information about parks, rivers, roads, and parcel information in the Los Angeles area to create practical planning tools to promote habitat conservation, watershed health, and recreational open space.

A part of the Green Visions project is to provide access to information about road networks, rivers, parks, and parcels on the web and allow administrators to quickly see different statistics, such as the number of parks within 1 mile of a given parcel. QGM can

[1] http://greenvisionsplan.net/

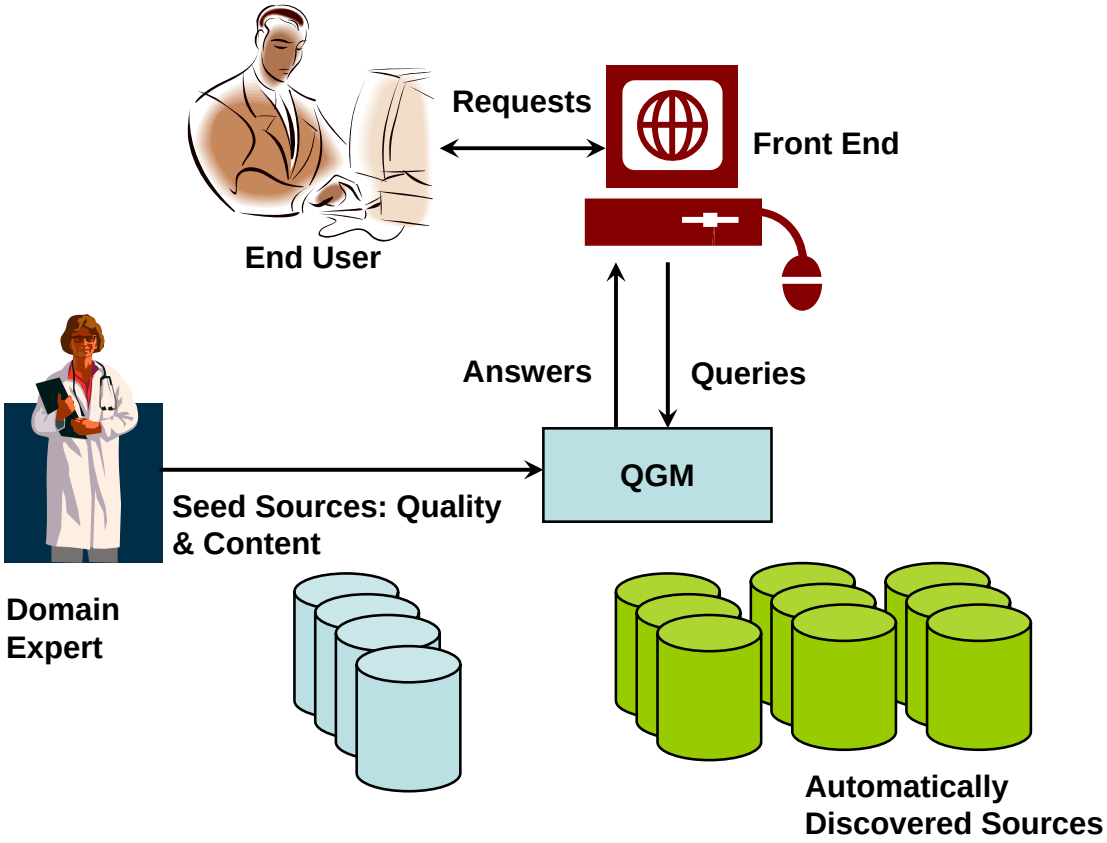

Figure 6.1: Example Architecture for Urban Response Systems

easily be used as a back-end for such a system. As the accuracy of the statistics depends on the quality of input data, QGM's quality-driven query answering would ensure that the best-quality geospatial data is used to compute the statistics shown to the user.

The overall architecture for a website that provides information for such applications is shown in Figure 6.1. The domain experts identify several seed sources and provide QGM information about the quality of data from those sources. Given this information and a list of available geospatial source repositories, QGM can automatically determine content and quality of data provided by sources. When the users ask queries through the web, user interface module utilizes the user inputs to formulate queries to QGM. Given

the query, QGM utilizes the information about the content and the quality of available sources to answer the user query. The user interface module shows the results to the user.

6.2.2 Disaster Response

Disaster response applications have a unique requirement in that the location of a disaster is not pre-determined. This implies that a disaster response application must have access to geospatial data that covers a large area. However, disaster response applications also require accurate geospatial data. As collecting different types of geospatial data for large areas is a very tedious, time-consuming, and expensive process, disaster response applications often rely on data in public domain or data collected by other agencies. However, determining which sources should be used for different types of geospatial data is a non-trivial task.

QGM can act as a back-end for disaster response applications and provide high quality geospatial data in response to user queries. QGM is well-suited for such applications due to three reasons: (1) its ability to automatically determine the content of the sources, (2) its ability to estimate the quality of data provided by different sources, and (3) its flexible and expressive approach to describe the quality restrictions. Disaster response applications can use different quality metrics for different queries. For example, in case of a query to identify all roads over a certain elevation to respond during flooding, the positional accuracy of the elevation data may be very important to the responders. While in case of a query to find all buildings effected by a fire the completeness attribute may be more important to ensure that all possible buildings are accounted for.

6.3 Future Work

There are a number of future directions for this work that will allow the techniques I developed to be applied more broadly. These directions can be categorized as improving one of the following five areas: (1) source discovery, (2) automatic source description generation, (3) quality estimation, (4) query answering, and (5) utilizing QGM in other domains. I discuss some possibilities in this sections.

6.3.1 Source Discovery

QGM does not have a source discovery module. Instead it relies on identifying sources from the list of source repositories given by a user. It would be fairly easy to design a crawler that searches for geospatial data and provides the list of discovered sources to QGM. QGM can then use its automatic source description generator module to automatic determine the content and quality of the data provided by the discovered sources.

Another approach to source discovery would be to analyze QGM's list of available sources, identify where QGM either does not have a good coverage of certain type of data or does not have access to high quality data and use that information to guide the discovery process. By utilizing the information about the available sources, we can find sources that have higher probability of increasing QGM's coverage of a type of data or improve the quality of data QGM can provide in some areas.

6.3.2 Automatic Source Description Generation

Another area of possible improvements to QGM is in the automatic source descriptions generation module. I believe there are three possible ideas for improvement. The first

idea is to create a domain concepts hierarchy that covers more types of geospatial and temporal entities. For example, if we add more weather related domain concepts, QGM can differentiate between layers named 'rainfall in past 15 days' and 'rainfall in past 30 days'. Depending on the type of queries, having a more detailed domain model may be very useful.

The second idea is to adopt an existing information retrieval techniques to improve the accuracy of automatic matching of domain concepts and layer names. The current automatic matching technique does not use any very commonly occurring words as stop words or utilize frequency of terms. The performance of the automatic matching technique can be improved by utilizing an existing information retrieval techniques that take into account not only the stop words and frequency of terms.

The third approach to improving the automatic source description generation is to improve the coverage information in the generated rules. QGM can generate more accurate coverage information for different raster sources by utilizing techniques described by Mark Carman in his thesis [15]. Intuitively after performing initial sampling of raster data and generating the Voronoi diagram, QGM can sample data at the class boundaries to further improve the coverage information.

6.3.3 Quality Estimation

QGM's quality estimation process can be improved in two ways. First, we can add some image processing capability in QGM to analyze the images returns by the *Raster* data sources, so that we can also analyze the quality of image sources. This would really be useful for images that provide information about real-world geospatial entities, such as

transportation maps, as we can check to see how many features appear in the image. We can also utilize the text-metadata provided by the image sources to identify the date image was collected and the resolution of the image for aerial photos or satellite images.

Second, we can improve QGM's sampling process by utilizing the features from the reference set. In particular, if we have access to several reference sources for a given area or a high-quality reference set, we can use the density of features in the reference set to determine the location where we should sample the data. Sampling in a dense area would result in more accurate values for the completeness attribute as QGM will be able to analyze a larger number of features.

6.3.4 Query Answering

The final area of future work is on improving the response time of query answering. There are several ways to improve the query answering. We can reduce the plan generation time by utilizing a spatial database to represent the coverage of the sources and prune the list of sources based on the area of interest specified in the query as a pre-processing step using the database. We can run the rest of the query answering step using the pruned list of sources.

In the experimental results we saw that when answering queries involving a large number of sources and complicated quality restriction, QGM may take as long as 10 minutes. We can address this problem by using several instances of QGM running on separate processors or on separate machines and divide the available sources between all instances. We could use one instance of QGM that represents all other instances as sources to integrate the results from different instances as shown in Figure 6.2. The key

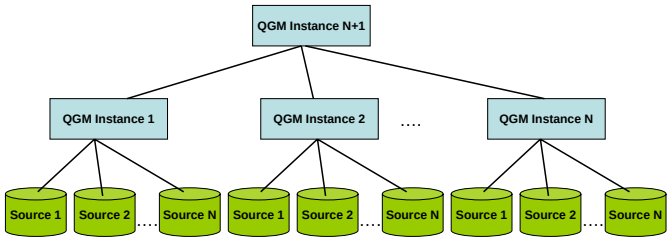

Figure 6.2: Hierarchy of QGM Instances

advantage of this is that it can be done without any change to QGM's existing code and can significantly improve the response time of the user queries.

We can also add new operators in QGM to support a wider variety of applications. In particular, it would be very useful to add operators that perform various statistical operations implemented by a GIS system as users may want to utilize those operations in determining the best-quality datasets. As QGM already allows developers to connect to ESRI's ArcGIS Engine Developer kit, it would be easy to add the capability to perform more statistical operations implemented in the developer kit.

6.3.5 Using QGM in Other Domains

The implementation of QGM is not specific to the geospatial domain. The only specific parts to geospatial data are the spatial selection operations and automatic generation of plans to access geospatial sources. In order to apply the automatic source generation and quality estimation techniques to other domains, we will need to use different measures, but the basic idea of sampling data and using it to estimate quality will apply in all domains.

However, QGM's quality-driven query answering method will apply in most domains. In order to utilize QGM in a new domain, first we need to define a set of domain concepts and their relationships that apply to the new domain. Next, we would need to define the quality attributes for different domain concepts. Next, we can write Local-As-View rules to define available sources as views over the selected domain concepts. In addition, we also provide rules that relate the quality of data provided by the sources to the quality relations defined at the domain level. Finally, we also provide facts related to the quality of data provided by different sources to QGM. If necessary we may need to implement some operations applicable to the new domain. Given all this information, QGM would be able to provide high quality data in response to user queries.

Reference List

[1] S. Adali and R. Emery. A uniform framework for integrating knowledge in heterogeneous knowledge systems. In *Proceedings of the Eleventh IEEE International Conference of Data Engineering*, 1995.

[2] Y. Arens, C. A. Knoblock, and W.-M. Shen. Query reformulation for dynamic information integration. *Journal of Intelligent Information Systems - Special Issue on Intelligent Information Integration*, 6(2/3):99–130, 1996.

[3] N. Ashish, C. A. Knoblock, and A. Levy. Information gathering plans with sensing actions. In *European Conference on Planning, ECP-97*, Toulouse, France, 1997.

[4] F. Bancilhon, D. Maier, Y. Sagiv, and J. D. Ullman. Magic sets and other strange ways to implement logic programs. In *Proceedings of Principles of Database Systems (PODS)*, pages 1–15, 1986.

[5] G. Barish and C. A. Knoblock. An expressive language and efficient execution system for software agents. *Journal of Artificial Intelligence Research*, 23:625–666, 2005.

[6] R. J. Bayardo Jr., W. Bohrer, R. S. Brice, A. Cichocki, J. Flower, A. Helal, V. Kashyap, T. Ksiezyk, G. Martin, M. Nodine, M. Rashid, M. Rusinkiewicz, R. Shea, C. Unnikrishnan, A. Unruh, and D. Woelk. Infosleuth: Agent-based semantic integration of information in open and dynamic environments. In *Proceedings of 1997 ACM international conference on Management of data (SIGMOD)*, 1997.

[7] L. Berti-Equille. Integration of biological data and quality-driven source negotiation. In *ER*, pages 256–269, 2001.

[8] J. Bleiholder, S. Khuller, F. Naumann, L. Raschid, and Y. Wu. Query planning in the presence of overlapping sources. In *Proceedings of Extending Database Technology (EDBT)*, pages 811–828, 2006.

[9] S. Borzsonyi, D. Kossmann, and K. Stocker. The skyline operator. In *Proceedings of the 17th International Conference on Data Engineering*, pages 421–430, Washington, DC, USA, 2001. IEEE Computer Society.

[10] O. Boucelma and F.-M. Colonna. Mediation for online geoservices. In *Proceedings of Web and Wireless Geographical Information Systems*, pages 81–93, 2004.

[11] O. Boucelma, F.-M. Colonna, and M. Essid. The VirGIS geographic integration system. In *BDA*, pages 357–361, 2004.

[12] O. Boucelma, M. Essid, Z. Lacroix, J. Vinel, J.-Y. Garinet, and A. Bétari. VirGIS: Mediation for geographical information systems. In *Proceedings of the International Conference on Data Engineering (ICDE)*, pages 855–856, 2004.

[13] O. Boucelma, J.-Y. Garinet, and Z. Lacroix. The VirGIS wfs-based spatial mediation system. In *Proceedings of the Conference on Information and Knowledge Management (CIKM)*, pages 370–374, 2003.

[14] I. Budak Arpinar, A. Sheth, C. Ramakrishnan, E. Lynn Usery, M. Azami, and M.-P. Kwan. Geospatial ontology development and semantic analytics. *Transactions in GIS*, 10(4):551–575, 2006.

[15] M. Carman. *Learning Semantic Definitions of Information Sources on the Internet*. PhD thesis, School of Information and Communication Technologies, University of Trento, 2006.

[16] S. Chandrasekaran, O. Cooper, A. Deshpande, M. J. Franklin, J. M. Hellerstein, W. Hong, S. Krishnamurthy, S. Madden, V. Raman, F. Reiss, and M. Shah. Telegraphcq: Continuous dataflow processing for an uncertain world. In *Proceedings of the Conference on Innovative Data Systems Research (CIDR)*, 2003.

[17] C.-C. Chen. *Automatically and Accurately Conflating Road Vector Data, Street Maps and Orthoimagery*. PhD thesis, University of Southern California, 2005.

[18] C.-C. Chen, S. Thakkar, C. A. Knoblock, and C. Shahabi. Automatically annotating and integrating spatial datasets. In *Proceedings of International Symposium on Spatial and Temporal Databases*, Santorini Island, Greece, 2003.

[19] R. L. Cole and G. Graefe. Optimization of dynamic query evaluation plans. In *Proceedings of the 1994 ACM SIGMOD international conference on Management of data*, pages 150–160, New York, NY, USA, 1994.

[20] O. M. Duschka. *Query Planning and Optimization in Information Integration*. PhD thesis, Stanford University, 1997.

[21] B. A. Eckman, T. Gaasterland, Z. Lacroix, L. Raschid, B. Snyder, and M.-E. Vidal. Implementing a bioinformatics pipeline (bip) on a mediator platform: Comparing cost and quality of alternate choices. In *Proceedings of the International Conference on Data Engineering (ICDE) Workshops*, page 67, 2006.

[22] M. Essid, O. Boucelma, F.-M. Colonna, and Y. Lassoued. Query processing in a geographic mediation system. In *12th ACM International Workshop on Geographic Information Systems(ACM-GIS 2004)*, pages 101–108, 2004.

[23] O. Etzioni and D. Weld. A softbot-based interface to the internet. *Communications of the ACM*, 37(7):72–76, 1994.

[24] S. Finkelstein. Common expression analysis in database applications. In *Proceedings of the 1982 ACM SIGMOD international conference on Management of data*, pages 235–245, New York, NY, USA, 1982. ACM Press.

[25] D. Florescu, L. Raschid, and P. Valduriez. A methodology for query reformulation in CIS using semantic knowledge. *International Journal of Cooperative Information Systems*, 5(4):431–468, 1996.

[26] F. Fonseca, M. Egenhofer, P. Agouris, and G. Camara. Using ontologies for integrated geographic information systems. *Transactions in GIS*, 6(3):231–257, 2002.

[27] F. Fonseca, M. Egenhofer, C. Davis, and K. Borges. Ontologies and knowledge sharing in urban GIS. *Computers, Environment and Urban Systems*, 24(3):251–272, 2000.

[28] F. Fonseca, M. Egenhofer, C. Davis, and G. Camara. Semantic granularity in ontology-driven geographic information systems. *Annals of Mathematics and Artificial Intelligence*, 36(1-2):121–151, 2002.

[29] S. J. Fortune. A sweepline algorithm for voronoi diagrams. *Algorithmica*, pages 153–174, 1987.

[30] H. Garcia-Molina, J. Hammer, K. Ireland, Y. Papakonstantinou, J. Ullman, and J. Widom. Integrating and accessing heterogeneous information sources in TSIMMIS. In *Proceedings of the AAAI Symposium on Information Gathering*, Stanford, CA, 1995.

[31] M. R. Genesereth, A. M. Keller, and O. M. Duschka. Infomaster: An information integration system. In *Proceedings of ACM Conference on Management of Data (SIGMOD)*, 1997.

[32] M. GoodChild. Theoretical models for uncertain GIS. In W. Shi, M. F. Goodchild, and P. F. Fisher, editors, *Spatial Data Quality*, pages 1–4, 2002.

[33] M. GoodChild. Models for uncertainty in area-class maps. In W. Shi, M. F. Goodchild, and P. F. Fisher, editors, *Proceedings of the Second International Symposium on Spatial Data Quality*, pages 1–9, 2003.

[34] M. F. Goodchild. Geographical information science. *International Journal of Geographical Information Systems*, 1992.

[35] A. Gupta, R. Marciano, I. Zaslavsky, and C. Baru. Integrating gis and imagery through xml-based information mediation. In *Proceedings of NSF International Workshop on Integrated Spatial Databases: Digital Images and GIS*, 1999.

[36] A. Gupta, A. Memon, J. Tran, R. P. Bharadwaja, and I. Zaslavsky. Information mediation across heterogeneous government spatial data sources. In *Proceedings of the 2002 annual national conference on Digital government research*, pages 1–6, 2002.

[37] R. H. Guting. An introduction to spatial database systems. *The VLDB Journal*, 3(4):357–399, 1994.

[38] J. M. Hellerstein, M. J. Franklin, S. Chandrasekaran, A. Deshpande, K. Hildrum, S. Madden, V. Raman, and M. A. Shah. Adaptive query processing: Technology in evolution. *IEEE Data Engineering Bulletin*, 2000.

[39] Y. E. Ioannidis, R. T. Ng, K. Shim, and T. K. Sellis. Parametric query optimization. *VLDB Journal: Very Large Data Bases*, 6(2):132–151, 1997.

[40] Z. G. Ives, D. Florescu, M. Friedman, A. Levy, and D. S. Weld. An adaptive query execution system for data integration. In *Proceedings of the ACM Conference on Management of Data (SIGMOD)*, 1999.

[41] S. Kambhampati, E. Lambrecht, U. Nambiar, Z. Nie, and S. Gnanaprakasam. Optimizing recursive information gathering plans in EMERAC. *Journal of Intelligent Information Systems*, 2003.

[42] A. Kiernan. Homeland security and geographic information systems how gis and mapping technology can save lives and protect property in post-september 11th america. *Public Health GIS News and Information*, 52:20–23, 2003.

[43] M. Kifer and E. L. Lozinskii. On compile-time query optimization in deductive databases by means of static filtering. *ACM Transactions in Database Systems*, 15(3):385–426, 1990.

[44] E. Klien, M. Lutz, and W. Kuhn. Ontology-based discovery of geographic information services - an application in disaster management. *Computers, Environment and Urban Systems*, 30(1):102–123, 2006.

[45] C. A. Knoblock, S. Minton, J.-L. Ambite, N. Ashish, I. Muslea, A. Philpot, and S. Tejada. The ariadne approach to web-based information integration. *International Journal on Intelligent Cooperative Information Systems (IJCIS)*, 10(1-2):145–169, 2001.

[46] M. Lenzerini. Data integration: A theoretical perspective. In *Proceedings of ACM Symposium on Principles of Database Systems*, Madison, Winsconsin, USA, 2002.

[47] A. Y. Levy, A. Rajaraman, and J. J. Ordille. Query-answering algorithms for information agents. In *Proceedings of AAAI-96*, 1996.

[48] C. Li. Computing complete answers to queries in the presence of limited access patterns. *The VLDB Journal*, 12:211–227, 2003.

[49] M. Lutz and E. Klien. Ontology-based retrieval of geographic information. *International Journal of Geographical Information Science*, 20(3):233–260, 2006.

[50] V. Manpuria, I. Zaslavsky, and C. Baru. Web services for accuracy-based spatial query rewriting in a wrapper-mediator system. In *Proceedings of the Fourth International W2GIS Conference*, pages 63–71, 2003.

[51] G. A. Mihaila, F. Naumann, L. Raschid, and M.-E. Vidal. A data model and query language to explore enhanced links and paths in life science sources. In *WebDB*, pages 133–138, 2005.

[52] D. Milano, M. Scannapieco, and T. Catarci. Peer-to-peer data quality improvement in the daquincis system. *Journal of Digital Information Management*, 3(3), 2005.

[53] J. F. Naughton, D. J. DeWitt, D. Maier, A. Aboulnaga, J. Chen, L. Galanis, J. Kang, R. Krishnamurthy, Q. Luo, N. Prakash, R. Ramamurthy, J. Shanmugasundaram, F. Tian, K. Tufte, S. Viglas, Y. Wang, C. Zhang, B. Jackson, A. Gupta, and R. Chen. The niagara internet query system. *IEEE Data Engineering Bulletin*, 24(2):27–33, 2001.

[54] F. Naumann. From databases to information systems - information quality makes the difference. In *Proceedings of the International Conference on Information Quality (IQ)*, Cambridge, MA, 2001.

[55] F. Naumann, U. Leser, and J. C. Freytag. Quality-driven integration of heterogenous information systems. In *Proceedings of 25th International Conference on Very Large Data Bases*, pages 447–458, 1999.

[56] H. Onsrud, B. Poore, R. Rugg, R. Taupier, and L. Wiggins. The future of the spatial information infrastructure. In R. B. McMaster and L. E. Usery, editors, *A Research Agenda for Geographic Information Science*, pages 225–255. Boca Raton: CRC Press, 2004.

[57] D. Papadias, Y. Tao, G. Fu, and B. Seeger. An optimal and progressive algorithm for skyline queries. In *Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, pages 467–478, New York, NY, USA, 2003.

[58] M. Perkowitz and O. Etzioni. Category translation: Learning to understand information on the internet. In *International Joint Conference on Artificial Intelligence, IJCAI-95*, pages 930–938, Montreal, Canada, 1995.

[59] D. Roman and E. Klien. Swing - a semantic framework for geospatial services. In K. T. Arno Scharl, editor, *The Geospatial Web. In the Advanced Information and Knowledge Processing Series*, pages 227–237. Springer, 2007.

[60] A. Saalfeld. *Conflation: Automated Map Compilation*. PhD thesis, University of Maryland, 1993.

[61] M. Scannapieco, A. Virgillito, C. Marchetti, M. Mecella, and R. Baldoni. The architecture: a platform for exchanging and improving data quality in cooperative information systems. *Information Systems*, 29(7):551–582, 2004.

[62] C. Sister, J. Wolch, J. Wilson, A. Linder, M. Seymour, J. Byrne, and J. Swift. Green visions plan for 21st century southern california. 14. park and open space resources in the green visions plan area. Technical report, University of Southern California GIS Research Laboratory and Center for Sustainable Cities, Los Angeles, California, 2007.

[63] S. Thakkar, J. L. Ambite, and C. A. Knoblock. A view integration approach to dynamic composition of web services. In *Proceedings of 2003 ICAPS Workshop on Planning for Web Services*, Trento, Italy, 2003.

[64] S. Thakkar, J. L. Ambite, and C. A. Knoblock. A data integration approach to automatically composing and optimizing web services. In *Proceedings of 2004 ICAPS Workshop on Planning and Scheduling for Web and Grid Services*, 2004.

[65] S. Thakkar, J. L. Ambite, and C. A. Knoblock. Composing, optimizing, and executing plans for bioinformatics web services. *The VLDB Journal, Special Issue on Data Management, Analysis, and Mining for the Life Sciences*, 14(3):330–353, 2005.

[66] J. Ullman. *Principles of Data and Knowledge-Base Systems*. Computer Science Press, New York, 1988.

[67] C. J. van Rijsbergen. *Information Retrieval*. Butterworth Publications, 1979.

[68] M. F. Worboys. Computation with imprecise geospatial data. *Computers, Environment and Urban Systems*, 22(2):85–106, 1998.

[69] M. F. Worboys and E. Clementini. Integration of imperfect spatial information. *Journal of Visual Languages and Computing*, 12:61–80, 2001.

[70] I. Zaslavsky and C. Baru. Grid-enabled mediation services for geospatial information. In *Proceeding of workshop on Next Generation Geospatial Information*, 2003.

[71] I. Zaslavsky and A. Memon. Geon: Assembling maps on demand from heterogeneous grid sources. In *Twenty-fourth Annual ESRI International User Conference*, 2004.

Appendix A

List of Data Layers

List of vector data layers in the domain hierarchy :

- Vector,
  - Boundaries
    - Administrative
      - Area Codes
      - Census Blocks
      - Census Block Groups
      - Census Tracts
      - Land Use
      - Zipcodes
      - Zoning
    - Political
      - Administrative Areas
      - Cities
      - Coast Lines
      - Counties
      - Countries
      - International Borders
      - Precincts
      - States
    - Others
      - Barrier Lines
      - Markers
  - Buildings
    - Schools
      - High Schools
      - Middle Schools
      - Elementary Schools

        - Other
    - Churches
        - Churches
    - Hospitals
        - Hospitals
    - Other
        - Libraries
        - Post Offices
        - Other
- Cartography
    - Property
        - Lots
        - Parcels
        - Other
- Elevation
    - Contours
        - Elevation Contours
        - Depth Contours
    - Other
        - Elevation Void Collection Areas
        - Spot Elevations
        - Terrain
- Hydrography
    - Coast
        - Coastal Areas and Islands
        - Inland Shorelines
        - Oceans
    - Rivers/lakes
        - Dams
        - Lakes
        - Rapids and Waterfalls
        - Rivers
        - Water Courses and Bodies
        - Wells and Springs
    - Other
        - Aqueduct
        - Arrow Points
        - Cistern Point Features
        - Danger

            - Hydrography Void Collection Area
            - Inundation Areas
            - Locks
            - Sea Structures
- Industry,
    * Agriculture
        - Agriculture Storage Sites
        - Farms
    * Commercial
        - Material Treatment Plants
        - Mines/Quarries
        - Non-Communication Towers
    * Facilities
        - Disposal
        - Extraction
        - Power
        - Processing/treatment Plants
        - Rigs and Wells
        - Storage
        - Tank and Water Towers
    * Other
        - Cistern Points
        - Complex Areas
        - Obstructions
- Physiography
    * Physiography
        - Asphalt Lakes
        - Bluffs/Cliffs/Escarpments
        - Cave and Mountain Passes
        - Cuts and Embankments/Fills
        - Geothermal Features
        - Glaciers and Snow/Ice Fields
        - Ground Surface Areas
        - Ice Shelf/Polar Ice/Pack Ice Areas
        - Islands and Ground Surface Areas
        - Landform Areas
        - Moraines
        - Physiography Void Collection Areas
        - Salt Pans/Sebkhas/Sand Dunes/Hills

- Population
  - Population
    - Built-Up Area
    - Fortification Areas Sites
    - Huts
    - Landmarks
    - Military Areas
    - Miscellaneous Population
    - Mobile Home Areas
    - Parks
    - Plaza
    - Race Tracks
    - Ruins
    - Settlement
    - Sport Field
- Transportation
  - Air
    - Aerial Cableways
    - Aircraft Facility Area
    - Airport/Airfield
    - Air routes
    - Runways
  - Ground
    - Bridge/Overpasses
    - Driveway Entrances
    - Hubs
    - Interchanges
    - Interstates
    - Major Roads
    - Minor Roads
    - Ramp Lines
    - Rest Areas
    - Roads
    - Traffic Counts
    - Tram Lines
    - Tunnels
    - Vehicle Stopping Area
    - Vehicle Storage/Park
  - Other

            - Anchorage
            - Culvert
            - Fords
            - Lighthouse Points
            - Snow/Ice Shed
        * Water
            - Ferry Crossings
            - Harbor Areas
            - Pier/Wharf/Quay
            - Waterways
- Utilities
    * Communication
        - Communication Buildings
        - Communication Towers/Disks
    * Pipelines
        - Pipelines
        - Pumping Stations
    * Powerlines
        - Power Transmission Lines
        - Power Plant Sites
        - Substation
    * Telephone lines
        - Telephone Lines
- Vegetation
    * Agriculture
        - Cropland
        - Orchad/Vineyard
        - Oasis
    * Other
        - Grassland
        - Hedgerow
        - Marsh Swamp
        - Tundra
        - Trees
        - Other
- Weather
    * Rainfall
        - Ice
        - Total Rainfall

        * Stations
            · Nexrad Stations
            · Weather Stations
        * Temperature
            · Barometer
            · Average Temperature
            · Low Temperature
            · High Temperature
            · Relative Humidity
        * Warnings
            · Conditions
            · Watches and Warnings
        * Wind
            · Direction
            · Speed

List of raster data layers in the domain hierarchy :

- Raster
    - Image
        * Aerial Photo
            · Ortho Photo
            · Other
        * Satellite Image
            · Ortho Images
            · Multi-spectral Image
            · B/W Image
    - Map
        * Transportation
            · Road Maps
            · Other
        * Topographic
            · Topo Maps
    - Elevation
        * Elevation Grids
            · Elevation Maps
    - Weather
        * Weather Maps
            · Cloud Cover
            · Weather Map

Appendix B

Final Theseus Plan for Running Example

```
PLAN finalplan{
   INPUT:stream constrel1, stream constrel2, stream constrel3,
         stream consrel4, stream constrel5, stream qualityrel
   OUTPUT: stream outrel
   BODY
   {
      /*requests to quality relations*/
      IS1Quality(:rel33)
      IS2Quality(:rel34)
      VS1Quality(:rel35)
      VS2Quality(:rel36)
      VS4Quality(:rel37)

      /*source requests*/
      IS1(consrel1:rel8)
      IS2(consrel2:rel9)
      VS1(constrel3:rel12)
      VS2(constrel4:rel13)
      VS4(constrel5:rel11)

      /*union for image sources*/
      union(rel8, rel9: rel6)
      /*union for vector data sources*/
      union(rel10,rel11:rel7)
      union(rel12,rel13:10)

      /*joins for image data*/
      join(rel8,rel33, "1=1" :rel28)
      join(rel9,rel34, "1=1" :rel29)
      /*joins for vector data*/
      join(rel12,rel35, "1=1" :rel30)
      join(rel13,rel36, "1=1" :rel31)
      join(rel11,rel37, "1=1" :rel32)
```

```
   /*selects for images*/
   select(rel26, "size = '[600,600]'":rel19)
   select(rel27, "size = '[600,600]'":rel21)
   select(rel28, "bbox = '[[33,-116],[34,118]]'" :rel26)
   select(rel29, "bbox = '[[33,-116],[34,118]]'" :rel27)
   /*selects for vector data*/
   select(rel30, "bbox = '[[33,-116],[34,118]]'" :rel23)
   select(rel31, "bbox = '[[33,-116],[34,118]]'" :rel24)
   select(rel32, "bbox = '[[33,-116],[34,118]]'" :rel25)

   /*Equals subtree for vector data*/
   project(rel17, "vectorobjquality,imageobjquality" : rel16)
   union(rel18,rel19:rel17)
   union(rel20,rel21:rel18)
   union(rel22,rel23:rel20)
   union(rel24,rel25:rel22)

   /*Equals subtree for image, which is just a rename from
     Equals subtree for vector data*/
   rename(rel16, "vectorobjquality imageobjquality,
           vectorobjdata imageobjdata" : rel15)

   /*join between Equals for vector data and Q1Quality*/
   join(rel16,qualityrel, "l.vectorobjquality =
         r.vectorobjquality" : rel14)

   /*join between Equals for image and join of Equals for
     vector and Q1Quality*/
   join(rel14,rel15, "l.imageobjquality = r.imageobjquality"
         : rel2)

   /*the Q1data subtree*/
   project(rel3, "imageobjdata,vectorobjdata" : rel1)
   select(rel4, "size = '[600,600]'"' :rel3)
   select(rel5, "bbox = '[[33,-116],[34,118]]'" :rel4)
   join(rel6, rel7, "l.bbox = r.bbox" : rel5)

   /*join between the Equals, Q1Quality, and Q1Data*/
   join(rel1,rel2,"l.imageobjdata = r.imageobjdata and
         l.vectorobjdata = r.vectorobjdata" : rel0)

   project(rel0, "imageobjdata,vectorobjdata, resdiff,
                  datediff" : outrel)
}
```

```
}
```

Appendix C

Generating Subplans to Access Geospatial Sources

QGM has the ability to automatically generate Theseus plans to retrieve data from databases, shapefiles, Web Map Servers (WMS), ArcIMS Servers, and Web Services. In this section, I describe the plans generated for popular geospatial source types: shapefiles, Web Map Servers, and ArcIMS Servers.

Shapefiles: QGM utilizes the apply operator to call an external function that accesses information from the shapefiles. I first describe the Java function that the apply operator calls followed by the Theseus plan with an apply operator to call the function.

The Java function that I wrote acts as a wrapper around the shapefile. It accepts two arguments: (1) complete path to the shapefile and (2) the bounding box of the area for which we need to retrieve the data. If the bounding box is not provided (or it is set to empty) the external function queries all data in the shapefile. Given the path and the bounding box, the function uses the Geotools open source toolkit[1] to access the shapefile and query all features located within the bounding box. The function writes the qualifying data to another shapefile and returns the complete path to the newly written shapefile.

The Theseus plan generated to access a shapefile data source contains one call to apply operation to access the shapefile. Below is an example plan:

```
PLAN roads1shapeplan{
   INPUT:stream inrel
   OUTPUT: stream outrel
   BODY
   {
     apply(inrel,"QGM.geo.getShapeData('c:\shapedata\roads1.shp',bbox)",
          "outdata" : outrel)
   }
}
```

The first argument to the external function contains the path to the shapefile, while the second argument contains the bounding box from the user query. The output relation contains the bounding box and complete path to the shapefile returned by the external function.

[1] http://www.geotools.org

Web Map Servers: Similar to shapefiles, I wrote another Java function using the Geotools Open Source toolkit that acts as a wrapper around any Web Map Server. The function to access Web Map Servers accepts the following arguments: (1) URL to the Web Map Server, (2) a bounding box, (3) name of the data layer, (4) size of the image requested, and (5) format for the image. The function returns a URL to the image returned by the Web Map Server.

The Theseus plan generated to access a Web Map Server source contains one call to apply operation. Below is an example plan:

```
PLAN wms1Topolayerplan{
   INPUT:stream inrel
   OUTPUT: stream outrel
   BODY
   {
     apply(inrel,"QGM.geo.getWMSData('http://www.wms.org',bbox,
          'Topolayer1', size, format)", "outdata" : outrel)
   }
}
```

The output relation contains the bounding box, size of the image, format of the image, and a URL to the image returned by the external function.

ArcIMS Servers: I wrote a Java function using the ArcGIS Engine Developer Kit[2] provided by ESRI.[3]. The Java function acts as a wrapper around any ArcIMS Server. Similar to the function to access Web Map Servers, the function to access ArcIMS Servers accepts the following arguments: (1) URL to the ArcIMS Server, (2) a bounding box, (3) name of the data layer, (4) size of the image requested, and (5) format for the image. The function returns a URL to the image returned by the ArcIMS Server.

The Theseus plan generated to access a ArcIMS Server source contains one call to apply operation. Below is an example plan:

```
PLAN ArcIMS1roadlayerplan{
   INPUT:stream inrel
   OUTPUT: stream outrel
   BODY
   {
     apply(inrel,"QGM.geo.getArcIMSData('http://www.arcims.org',bbox,
          'roadlayer', size, format)", "outdata" : outrel)
   }
}
```

The output relation contains the bounding box, size of the image, format of the image, and a URL to the image returned by the external function.

[2]http://www.esri.com/software/arcgis/arcgisengine/about/devkit.html
[3]http://www.esri.com