

USING GEOSPATIAL METADATA AND EXPERT SYSTEMS FOR THE PARTIAL AUTOMATION OF THE CARTOGRAPHIC DESIGN PROCESS

by

RICHARD ALLEN SMITH JR.

(Under the Direction of Xiaobai Yao)

ABSTRACT

Over the past thirty years, a revolution, spurred by technological innovation and driven by the importance placed on spatial analysis, has prompted a paradigm shift in the mapping world. Individuals once unable to participate in the map making process find themselves with the technology, if not the cartographic knowledge, to map whatever phenomena they desire. With this democratization of mapping comes a larger community of map makers and increased production and consumption of poorly designed maps and public perception of what a well-designed map is. To fight the trend of poorly designed maps, map makers must be educated in cartography and cartographic design. This research focuses on the combination of a map ontology, automatic theme identification through data mining, and a cartographic expert system to provide multiple starting points for taking the democratization of mapping to the next level: the democratization of cartography.

INDEX WORDS: cartography, maps, metadata, data mining, expert system, ontology, automation, artificial intelligence, democratization of cartography, democratization of mapping

USING GEOSPATIAL METADATA AND EXPERT SYSTEMS FOR THE PARTIAL
AUTOMATION OF THE CARTOGRAPHIC DESIGN PROCESS

by

RICHARD ALLEN SMITH JR.

B.S., Texas A&M University – Corpus Christi, 2003

M.S., Texas A&M University – Corpus Christi, 2006

A Dissertation Submitted to the Graduate Faculty of the University of Georgia in Partial
Fulfillment of the Requirements for the Degree

DOCTOR OF PHILOSOPHY

ATHENS, GEORGIA

2012

© 2012

Richard Allen Smith Jr.

All Rights Reserved

USING GEOSPATIAL METADATA AND EXPERT SYSTEMS FOR THE PARTIAL
AUTOMATION OF THE CARTOGRAPHIC DESIGN PROCESS

by

RICHARD ALLEN SMITH JR.

Major Professor: Xiaobai Yao

Committee: Thomas Hodler
Thomas Jordan
Lan Mu

Electronic Version Approved:

Maureen Grasso
Dean of the Graduate School
The University of Georgia
May 2012

DEDICATION

I dedicate this dissertation to my wonderful family and friends. Particularly to my supportive and encouraging wife, Seneca, who has kept me on task with her gentle nudges and her hours of reading and editing my dissertation. I look forward to working with her on our next big adventure: our new daughter, Claire.

ACKNOWLEDGEMENTS

I would like to thank all of the people who have help make this dissertation possible.

First I would like to thank my advisor, Xiaobai Yao for all of her guidance and helpful insights as well as Tom Hodler who was initially a mentor and professor, now friend and trusted colleague. I would also like to thank the rest of my committee for their support and for also letting me work on this dissertation from afar. Many thanks also to my colleagues at Texas A&M University – Corpus Christi who took a chance on hiring me while still working on this dissertation. I would also like to thank my fellow graduate students who provided invaluable support through their suggestions, encouragement, libations, and high Pac-Man scores. I couldn't have asked for a better cohort.

TABLE OF CONTENTS

	Page
ACKNOWLEDGEMENTS	v
LIST OF TABLES	ix
LIST OF FIGURES	x
CHAPTER	
1 INTRODUCTION	1
Purpose of the Research.....	1
Significance of the Research.....	7
Thesis Statement and Research Objectives.....	8
2 LITERATURE REVIEW	10
Introduction.....	10
Interactions between Geography and Artificial Intelligence	10
Decision Trees	21
Data Mining and Knowledge Discovery.....	23
Research and Uses of Ontologies in GIS.....	27
3 CONSTRUCTION OF THE MAP ONTOLOGY	31
Introduction.....	31
Overview of Ontologies	31
Methodology of Ontology Construction.....	34
Results: The Map Ontology	43

Chapter Summary	59
4 DATA MINING GEOSPATIAL METADATA	61
Introduction.....	61
Selecting the Inputs to Data Mine.....	61
Methods.....	65
Evaluating the Method with Experiments.....	101
Conclusions.....	124
Chapter Summary	125
5 CONSTRUCTION OF THE EXPERT SYSTEM.....	126
Introduction.....	126
Applicability of Expert Systems to Cartographic Symbol Design	126
Obtaining Expert Knowledge	130
Methodology	135
Results and Discussion	172
Conclusions.....	187
Chapter Summary	187
6 CONCLUSION.....	189
Conclusions.....	192
Future Work	194
REFERENCES	196
APPENDICES	
A Ontology Vocabulary.....	203
B Metadata Cleaning Program	206

C	Data Mining Algorithm-Friendly Data Formatting Program.....	216
D	Dataset Theme Prediction Program	220
E	Cartographic Expert System Program	235

LIST OF TABLES

	Page
Table 2.1: Nine Steps of the KDD Process.....	26
Table 4.1: Non-Meaningful Terms to be Removed from Metadata Documents	78
Table 4.2: Descriptive Statistics of Extracted Keywords and Field Names by Theme ...	103
Table 4.3: Theme Identification Accuracy Assessment Results.....	122
Table 5.1: Data Sets used in Large Scale Map	173
Table 5.2: Data Sets used in Small Scale Map	180

LIST OF FIGURES

	Page
Figure 1.1: Positive feedback loop degrading map quality.....	2
Figure 2.1: Basic function of an Expert System	13
Figure 2.2: Decision Tree Heuristic for Map Type Determination	22
Figure 2.3: Decision Tree Anatomy.....	23
Figure 3.1: Overview of the Map Ontology.....	45
Figure 3.2: The Thing Class.....	47
Figure 3.3: Map Class and Subclasses	48
Figure 3.4: MapProjection Class and Subclasses	49
Figure 3.5: MapScale Class	49
Figure 3.6: ProductionMedium Class	50
Figure 3.7: MapLayoutElement Class and Subclasses	51
Figure 3.8: Map Composition using Concepts in Map Ontology	52
Figure 3.9: SpatialPhenomenon Class and Subclasses	55
Figure 3.10: Graphic Class and Subclasses	56
Figure 3.11: VisualVariable Class and Subclasses	58
Figure 3.12: Attribute Class and Subclasses	59
Figure 4.1: Pseudo-code of Script that Removes Shapefiles without Metadata Files	70
Figure 4.2: Hierarchy of Target Components of CSDGM Metadata Standard	71
Figure 4.3: Pseudo-code Checking XML Metadata file for Required Components.....	74

Figure 4.4: Pseudo-code Checking HTML Metadata file for Required Components	75
Figure 4.5: Pseudo-code Checking Text Metadata file for Required Components	75
Figure 4.6: Pseudo-code of Descriptive Statistics Creation.....	76
Figure 4.7: Removal of Non-Meaningful Keywords	79
Figure 4.8: Removal of Plural Keywords if both Singular and Plural Found.....	80
Figure 4.9: extractLog.txt for Roads Theme.....	82
Figure 4.10: fieldNames.txt for Trails Theme	83
Figure 4.11: keywords.txt for Airports Theme	84
Figure 4.12: Match Dataset Keywords and Field Names to Extracted Keywords and Field Names	86
Figure 4.13: Sample Output from Keyword and Field Name Comparisons for Airport Theme.....	87
Figure 4.14: Extracting Geometry Type from Shapefiles.....	88
Figure 4.15: Addition of Geometry to Output File	88
Figure 4.16: Determining Value of Training Variable	89
Figure 4.17: Addition of Training Variable to Output File	90
Figure 4.18: Example Output from Data Mining Algorithm.....	97
Figure 4.19: Prediction Program Main Algorithm.....	101
Figure 4.20: Decision Tree and Classification Rule for State and County Boundary Theme.	105
Figure 4.21: Decision Tree and Classification Rule for Zip Code Theme	107
Figure 4.22: Decision Tree and Classification Rule for Contour Theme	109
Figure 4.23: Decision Tree and Classification Rule for Water Theme.....	110

Figure 4.24: Decision Tree and Classification Rule for Airports Theme	112
Figure 4.25: Decision Tree and Classification Rule for Hospitals Theme	113
Figure 4.26: Decision Tree and Classification Rule for Parks Theme	115
Figure 4.27: Decision Tree and Classification Rule for Police/Fire Theme	117
Figure 4.28: Decision Tree and Classification Rule for Railroads Theme	118
Figure 4.29: Decision Tree and Classification Rule for Roads Theme	120
Figure 4.30: Decision Tree and Classification Rule for Trails Theme	121
Figure 5.1: Obtained Cartographic Information in Initial IF-THEN and List Format.....	133
Figure 5.2: Obtained Cartographic Information in Expert System Structure	134
Figure 5.3: Example Fact Stored in CLIPS	137
Figure 5.4: Basic Syntax of deftemplate	140
Figure 5.5: Example Portion of Fact-List in CLIPS	141
Figure 5.6: Example deffacts Construct.....	142
Figure 5.7: Example deffacts and deftemplate for Basic Family Tree	144
Figure 5.8: find-Joe Rule	145
Figure 5.9: has-son Rule	146
Figure 5.10: find-father Rule	147
Figure 5.11: find-grandson Rule	148
Figure 5.12: Portion of an Agenda in CLIPS.....	150
Figure 5.13: mapLayer deftemplate	153
Figure 5.14: theme-ordering deftemplate	154
Figure 5.15: theme-hsv-color-preferences deftemplate	155
Figure 5.16: hue-range deftemplate	155

Figure 5.17: saturation-range deftemplate	156
Figure 5.18: value-range deftemplate	157
Figure 5.19: theme-symbol-preferences deftemplate	158
Figure 5.20: relative-outline-color deftemplate	158
Figure 5.21: scale-level deftemplate	159
Figure 5.22: current-scale deftemplate	159
Figure 5.23: input-scale deftemplate.....	160
Figure 5.24: Sample of initial-theme-ordering deffacts.....	161
Figure 5.25: Sample of initial-color-preferences deffacts	162
Figure 5.26: Sample of hsv-information deffacts	163
Figure 5.27: Sample of scales deffacts	163
Figure 5.28: initial-relative-outline-colors deffacts	164
Figure 5.29: Sample of initial-theme-symbol-preferences deffacts.....	164
Figure 5.30: choose-random-color defrule	166
Figure 5.31: fix-duplicate-drawOrders defrule	167
Figure 5.32: order-mapLayers defrule	167
Figure 5.33: set-scale defrule.....	169
Figure 5.34: assign-color-preferences defrule	170
Figure 5.35: match-theme-colors defrule.....	171
Figure 5.36: assign-symbology defrule.....	172
Figure 5.37: Large Scale Map with Default Colors	174
Figure 5.38: First Run of Expert System for Large Scale Data Sets	176
Figure 5.39: Second Run of Expert System for Large Scale Data Sets	177

Figure 5.40: Third Run of Expert System for Large Scale Data Sets.....	178
Figure 5.41: Summary of Default and Expert System Designed Maps.....	179
Figure 5.42: Small Scale Map with Default Colors.....	181
Figure 5.43: First Run of Expert System for Small Scale Data Sets	183
Figure 5.44: Second Run of Expert System for Small Scale Data Sets.....	184
Figure 5.45: Third Run of Expert System for Small Scale Data Sets.....	185
Figure 5.46: Summary of Default and Expert System Designed Maps.....	186

CHAPTER 1

INTRODUCTION

Purpose of the Research

The most common output from a geographic information system (GIS) are maps, which are the primary means of representing data stored and processed by a GIS. As maps are a primary means of representing geospatial data, it is important to clearly convey the graphical information within them to maximize usefulness and comprehension by the map reader. Effective map communication is achieved by creating well-designed maps, utilizing proper cartographic design principles throughout the cartographic design process. The cartographic design process is a complex and time consuming practice composed of choosing colors, line weights, symbols, patterns, arranging data layers and placing map elements. With an array of choices to be made with respect to cartographic design, designing an effective map may prove challenging and time consuming to both novice and expert map makers. In the past, map making was under the domain of cartographers who had formal cartographic training and experience. However, recent advances in computer and software technology has allowed for those without cartographic training to enter the map making process. This rapid expansion in the population of map makers poses an interesting problem; people are making maps without formal cartographic training and experience.

This democratization of mapping, spurred by the wide spread use of personal computers and prevalence of free and commercial GIS software packages, enables

anyone with a computer to create a map with a few clicks of a button (Kumar 2000). However, many maps produced during this recent mapping revolution do not effectively convey the intended information, which can frequently be attributed to a map makers' lack of cartographic design training. While the democratization of mapping is generally seen as a positive trend for society, (Rød and Ormeling 2001; Mattmiller 2006), it has been noted that “cartographic monstrosities” (Monmonier 1984) created by users with subpar (or no) cartographic training abound (Jenks 1976; Muller 1983; Robinson and Jackson 1985). The recent acceleration of the democratization of mapping continues to amplify Monmonier’s observation.

An enlarged community map makers coupled with an increase in the consumption of maps amplifies the possibility for the consumption of poorly designed maps and the public perception of what a well-designed map is, thus, creating a positive feedback loop (Figure 1.1) therefore, reducing the quality of maps that will be created in the future.

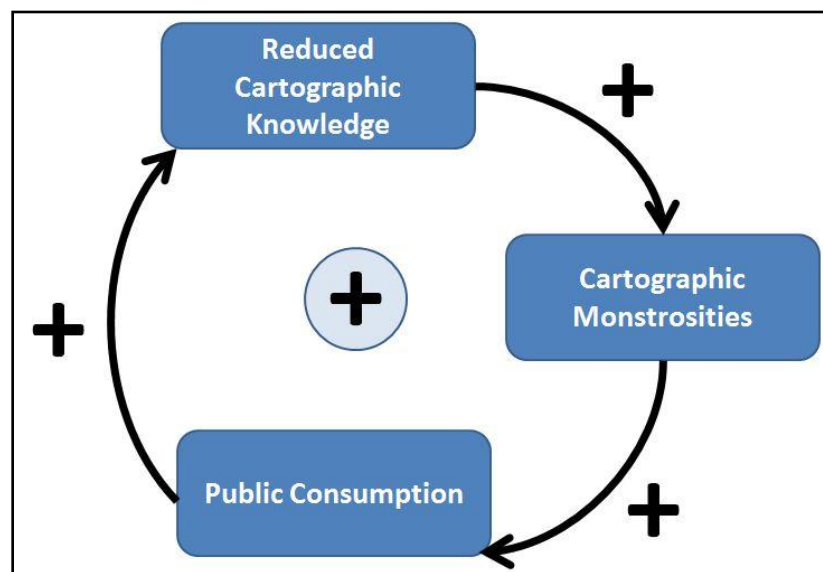


Figure 1.1: Positive feedback loop degrading map quality

To introduce negative feedback into this loop, map makers must be educated in cartography and cartographic design; however, many barriers exist in accomplishing this. One such barrier is the marginalization of cartography as a stand-alone academic topic (Goodchild 2000; Wood 2003; Krygier 2005). While cartography may not be offered as a single stand-alone course, it is being taught as a section of other courses, but not with the same depth and attention as if it was being taught in a stand-alone course (Olson 2004). While the threading of cartography through other courses continues to be popular, cartography must be taught as a stand-alone course to provide students with the core foundations of cartographic principles that will be applied throughout the remainder of their academic and, more importantly, their professional careers.

The academy is not only at fault for failing to provide education on cartographic principles but also geospatial software companies must share the burden of blame and move to remove the barriers for training users in cartographic design principles. Current software packages offer very little, if any guidance to users when it comes to designing a map. This lack of guidance does not provide an opportunity for users to learn or increase their chances of producing a well-designed map. With no feedback from the software, and no expert to review the map, users may not realize that their map is cartographically sub-par. Bad workers can blame their tools, but if the workers are cartographically ignorant, then the software companies providing the tools to the workers should take some responsibility to help train them (Robinson and Jackson 1985; Muller, Johnson, and Vanzella 1986). While it can be appreciated that software programs offer a blank slate for users to express their creativity, guidance for new (and even experienced) users will provide value by adding negative feedback in the aforementioned feedback loop.

It is common for GIS software packages to offer help topics on data operations (*e.g.* spatial analysis, network analysis), but help topics concerning cartographic design are often content absent or shallow with respect to the importance of a well-designed map. With mapmaking being a routine GIS operation (Chang 2008), the lack of available help pertaining to cartographic design in mapping software is unacceptable.

The irony of this situation is that users often do not read software manuals and, instead, opt to solve the problem by either trial-and-error, or asking an available expert (Novick and Ward 2006), thereby rendering any available software help unused. Novick and Ward continue to explain that while users do sometimes utilize online help, the help content is often organized similar to a printed manual, which is a format that has proved to be unpopular with users. Additionally, if a user is unfamiliar with a subject, they may not know the terminology required to effectively search the help system; thus creating another barrier to cartographic education. The combination of scant cartographic documentation and the resistance of users to utilize documentation, present an opportunity for an expert system to provide assistance through a non-passive help system.

An expert system is a non-passive help system that emulates the knowledge and decision-making ability of human experts through the passing of facts to, and receiving expertise from, the expert system. An expert system can differ from passive help systems by monitoring, and assisting in design decisions with the user, rather than just providing information in an on-line help system. Because an expert system contains a knowledgebase of domain knowledge, it can intelligently apply this information to a user's mapping problem. Expert systems offer many benefits to the casual and expert map maker. Many users prefer to ask an expert when available instead of using the

software's help system. An expert system provides the user with the expert knowledge they seek without needing access to a human expert. An expert system is always available whereas a human counterpart might not be. Additionally, an expert system can contain the knowledge of multiple human experts, thus, acting as a human expert multiplier. Like a human expert, instructions can be supplemented with detailed explanations covering the "why" aspect of the operation. Answering the users "why" questions elevate the expert system to the role of tutor, thus, teaching the user new skills while simultaneously solving their problems. The expert system's role of tutor can remove the barriers between the user and effective cartographic design education.

To successfully design and build a cartographic expert system, two supporting research steps are undertaken. First, an ontology is created to explicitly scope and define the area of expert knowledge that will be acquired from experts and converted into a cartographic knowledgebase. An ontology is a formal specification of a shared understanding of a knowledge domain that facilitates accurate and effective communications of meaning (Gruber 1993; Agarwal 2005). More directly, an ontology is composed of a taxonomic structure which contains objects and their relationships to other objects within a particular domain of knowledge. In practice, the ontology is a framework on which concepts can be formalized.

Storing the knowledge of human experts and cartography texts into an expert system requires the conversion of human knowledge into an expert system readable format. In order for an expert system to store and apply this knowledge, a set of concepts must first be defined so that the cartographic knowledge that is converted into a knowledgebase that can be utilized by the expert system. In addition to storing

cartographic concepts, this cartographic knowledgebase must also represent relationships and restrictions between the stored concepts. An ontology will act as the guide for the cartographic knowledgebase data structure used by the expert system. An ontology will model the concepts required to build a concept of a map, and will also diagram the relationships and restrictions between concepts, thereby providing the information required by an expert system to make inferences. By building a map ontology and, subsequently, a knowledgebase that is able to store the expert cartographic knowledge, an expert system will have all the required knowledge to assign appropriate map symbology.

In addition to the benefits provided to the expert system, building an ontology that represents the concept of a map assists the translation of knowledge between humans and a knowledgebase. The ontology acts as a definer, scoping the body of expert knowledge. Defining the body of knowledge to be acquired assists in defining the questions to ask experts and topics of texts to research. During the knowledge acquisition process, an ontology provides a well-defined structure to store the acquired knowledge.

The second supporting research step to developing a cartographic expert system is the data mining of freely-available geographic information. As discussed earlier, users resist using help systems for many reasons. An expert system provides many benefits in comparison to passive help systems, but since expert systems may be unfamiliar to many users; this may pose as a barrier to a user's utilization of an expert system. To address this barrier, further automation of the expert system is accomplished through the data mining of freely-available geographic information. The purpose of this mining is to find common patterns that will be exploited by the expert system in order to automatically identify a geographic dataset's theme. Specifically, keywords and field names found in

metadata documents of geographic datasets will be data mined. The fields, keywords, and field names were chosen as they provide short, concise words with the purpose of describing the contents of the dataset. The automatic identification of geographic dataset themes will reduce the amount of human-to-expert system interaction, thereby removing a barrier between the user and expert system.

Together with the supporting research steps, this research develops a new expert system designed to assist both expert and novice map makers with creating cartographically sound map. This expert system will assist in partially automating the cartographic design process, instructing the map maker in cartographic concepts, and attempting to disrupt the positive feedback loop affecting the quality of maps.

Significance of the Research

General reference maps will be the focus of the ontology, data mining, and expert system development. A general reference map displays natural or manmade objects with an emphasis on location of elements. Typical elements on a general reference maps are roads, boundaries, cities, water, points of interest and other similar objects (Mackaness 1986). General reference maps were selected for the following reasons: i) general reference maps are the type of map that map makers of all levels engage in creating; ii) the general public is most familiar with general reference maps, and, therefore, feel more comfortable creating these types of maps; and iii) general reference maps primarily depict location and, therefore, do not require analysis or calculation of the information displayed on the map. Eliminating analysis will simplify automating the cartographic process.

There are three objectives of this research: construct a map ontology, and data mine geospatial metadata, and design a cartographic expert system. The first objective, construction of a map ontology, provides a knowledge framework on which the expert system will draw its knowledge. In order for the expert system to assist in designing a map, the concept of a map and how the map parts make a whole must be defined; this is provided by an ontology. At present, no map ontology built with sufficient detail currently exists; requiring one to be constructed for this research. The second objective, data mining geospatial metadata, provides information required by the expert system to automatically identify datasets. It is the role of the computer to self-identify datasets to remove as many barriers between the user and expert system. The information required by the expert system to identify data sets will be created in this research through the data mining of metadata. Metadata documents contain descriptions of the contents stored inside of the associated geospatial dataset. The descriptions in the metadata, when data mined, yield useful patterns that can be leveraged by the expert system in determining the theme of a dataset. The third objective, design of a cartographic expert system, will partially automate the cartographic design process and to educate users in proper cartographic principles.

Thesis Statement and Research Objectives

How can cartographic knowledge be successfully expressed in an artificial intelligence system to partially automate the cartographic design process?

The research has three objectives that are addressed in order to evaluate the thesis statement:

- Analyze and construct a map ontology to contain cartographic principles and knowledge extracted from texts, maps and cartographers. This is discussed in Chapter 3.
- Utilize data mining and statistical techniques to analyze and model useful information and patterns from geospatial metadata. Chapter 4 discusses the techniques used and the findings.
- Develop an expert system that can intelligently apply cartographic principles of map design and symbolization. The development of the expert system is discussed in Chapter 5.

CHAPTER 2

LITERATURE REVIEW

Introduction

A critical literature review was made to assist in defining the research problem of this dissertation. This chapter is a review of interdisciplinary interaction between geography, artificial intelligence, data mining, and ontology development in the context of automating cartography. Each of the three interactions with geography and how they relate to the automation of cartography will be reviewed and discussed in this chapter.

Interactions between Geography and Artificial Intelligence

The automation of cartography has been an active research topic by geographers and cartographers more than forty years. Over this period of time, the research has taken many different forms and traversed many different meanings of “automation of cartography”. Through these distinct research efforts, many conceptual and technical breakthroughs were contributed to the scientific community making the possibility of truly automating cartography a more plausible goal. A review of these conceptual and technical breakthroughs will be covered in chronological order.

The automation of cartography initially referred to the use of computers in cartographic design process (Dobson 1979; Tobler 1959). The 1950s saw the first successful attempts to produce graphics from computers. The mid-1950s saw maps produced on line printers, followed by cathode ray tubes and tabulating equipment

(Rhind 1977). In the 1970's the production of digital maps now commonly known as computer assisted design (CAD) were introduced. The CAD view sought to research how a computer and its input and output devices are tools that, in effect, replace pencil and paper; the then common tools of the cartography trade (Monmonier 1982). In this era of cartographic automation, the *speed* at which the map was produced using digital tools was the result of the research, as CAD programs allowed cartographers to quickly construct and update maps compared with paper and pencil.

By the late 1970s to early 1980s, CAD had emerged as a useful technology in many disciplines. Simultaneously, companies like Intergraph, Autodesk, and Bentley Systems, (created in the early 1980's) were introducing CAD software not only to university and government labs, but also to personal computers thereby making CAD accessible to a large group of users, including cartographers (Carlson 2003).

In a CAD system, the computer software does not offer any advice or knowledge, but instead serves as a canvas for the user's input. While CAD systems matured and were increasingly used by a growing and diversifying user base, a second meaning of "automation of cartography" formed based on the promises of the emerging field of artificial intelligence (AI).

AI is the effort to make computers think, reason, and act rationally (Russell and Norvig 2003). Typically AI systems are modeled after human behavior, but it should be noted that the emulation of *any* intelligent behavior is desired; meaning that the intelligence being emulated need not necessarily be human.

The 1950s and 1960 saw an era of great enthusiasm about the prospects of AI. AI researchers saw successes in creating AI programs to solve problems. Initial successes

led to many millions of dollars being dedicated to ambitious AI research projects. Some such projects aimed to mimic human thinking and behavior within a matter of years (read (Feigenbaum and McCorduck 1983) for a well-known and interesting account of this ordeal). As time progressed however, AI disappointed the much of the scientific community as the task of programming an AI system was found to be much more difficult than originally expected. This was, in part, a result of the scale of the many of the problems that AI was focused on solving. Many of these problems were in the context of a “microworld” where the number of facts and objects were quite small. While microworlds were good test cases, AI was not able to successfully scale up to real world problems where facts, objects, and possibilities were essentially limitless (Russell and Norvig 2003). These combined problems resulted in a sharp decline in AI research by the scientific community in the late 1970s.

While scientists were not able to create a system that would mimic all human knowledge through AI, research began focusing on creating systems that would emulate a human expert on specific subjects; subjects on which a single human could provide as an expert. Essentially, if microworlds were too simplistic for real-world applications, and the real world was too complex for AI to operate successfully, then a single field of expertise was seen as an attainable compromise. This narrowing of goals (and scale) to bounded domains of knowledge led to the idea of expert systems.

An expert system is a computer program that represents and reasons with knowledge of some specialist subject with a view of solving problems or giving advice (Jackson 1990) and this is achieved through the passing of facts to and receiving expertise from the expert system; shown in Figure 2.1 (Giarratano and Riley 2005, 6) .

Expert systems have been defined as “an intelligent computer program that uses knowledge and inference procedures to solve problems that are difficult enough to require significant human expertise for their solution” (Edward Feigenbaum in Giarratano and Riley 2005, 5). That is, the goal of an expert system is to emulate the human thought process to arrive to the same solution as a human expert. A human “expert” is recognized as a reliable source of information in a knowledge area unknown by most.

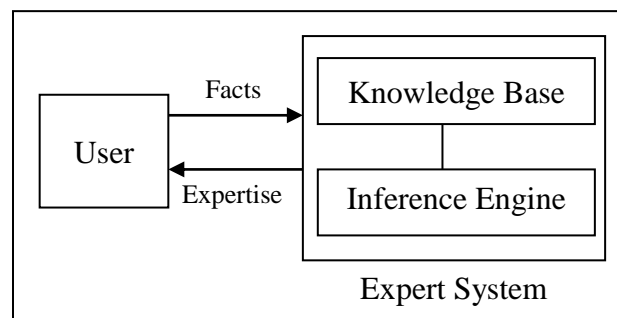


Figure 2.1: Basic Function of an Expert System (Giarratano and Riley 2005)

At a high level of abstraction, an expert system is composed of two parts: a knowledge base, and an inference engine. The knowledge base contains the knowledge that comes from the human expert in the knowledge domain of the human expert. The inference engine acts on the knowledge contained in the knowledge base and the facts presented to the expert system by the user to return a logical, *possible* answer. The answer returned from the inference engine is viewed as expertise, much like the expertise a human expert would return after considering facts against their knowledge.

The user of an expert system provides inputs, in the form of facts, and receives outputs in the form of emulated human expertise. The inputs to expert systems are considered “facts” about the state of the world on which the expert system will operate.

The facts, coupled with knowledge stored in the knowledge base are sent to the inference engine which returns an output. The output that the user receives will vary in form, but can generally be considered a *possible* expert answer, or also, a query for additional facts.

To better understand and describe an expert system, one can compare them to a conventional computer program. Expert systems often differ in design to conventional computer programs because the problem that is to be solved does not have a satisfactory algorithm and, instead, relies on inferences to achieve a reasonable solution (Giarratano and Riley 2005). An algorithm is the ideal solution to a problem as it guarantees a solution to a problem in finite time (Berlinski 2001). However, algorithms may not scale to larger problems as gracefully as AI nor be able to supply a satisfactory solution. To further illustrate the differences, consider how an expert system and a conventional program might approach the problem of site selection.

The expert system approaches a problem much in the same way that a human expert would. The human expert would first consider the larger picture of site selection, followed by an evaluation of smaller subsections bring in other supporting information to assist in making an informed decision. Intuition and past experiences may also be used by the human expert to come to a conclusion. Researchers have argued, and demonstrated, that expert system techniques are useful for determining site selection (Ortolana and Perman 1989; Wright 1989; Curry and Moutinho 1992; Arentze, Borgers, and Timmermans 1996).

Alternatively, conventional computer programs often use all-or-nothing screening criteria to narrow down the number of potential suitable sites, and often, even after this screening, there may still be too many potential sites remaining. Further screening may

take place, but this would risk the elimination of some or all of the best sites. Without the compliment of expert knowledge, the algorithm may never reach a suitable solution in the same way that an expert would (Arentze, Borgers, and Timmermans 1996).

Another difference between expert systems and a conventional computer program is the representation and storage of human knowledge. Expert systems separate the operational logic from the representations of human knowledge. The representations of human knowledge are stored in a knowledge base in the form of production rules. These production rules are composed of a name, antecedent, and consequent. If the operational logic portion of the expert system infers that a rule's antecedent evaluates to true, then the consequent action is performed. In contrast, conventional programs perform numerical calculations with data using inflexible algorithms embedded directly into a programs code. Conventional programs are procedural in nature, requiring the programming to provide the exact steps in an exact order to solve a certain problem. In contrast, an expert system is considered declarative programming. A declarative program separates goals from methods of attaining the goal. The programmer does not specify how the goal will be achieved by the program, instead, the expert system activates when facts entered by the user trigger a rule that tries to satisfy the goal. The order of the rules in the expert system program do not matter as any rule can be triggered at any time if sufficient conditions exist.

A last difference between expert systems and a conventional computer program is the use of heuristic knowledge (often referred to as 'rules of thumb'). An example of a heuristic is "On a color map, water bodies are blue with probability 0.99." This heuristic may be expressed as a production rule in the knowledge base "If the map is in color and

water bodies are shown on the map, then the water bodies are blue with a probability of 0.99.” Heuristic rules, such as this one, are typically captured from interviews and experts, reviewing publications, and/or field observations. A practical limitation of expert systems is a lack of casual knowledge (Giarratano and Riley 2005). It is easier to program expert systems with shallow knowledge based on experience and empirical knowledge than the complex underlying causes. Heuristic methods allow the expert system to remain shallow in knowledge as a heuristic does not require perfect data or perfect solutions; it aids in the solution with a guarantee of success. However, in many fields, such as medicine, heuristics still play an important part in problem solving. Even if an exact solution is known, a heuristic may still be the preferred method of solving the problem because of time or cost constraints.

Expert systems are considered to be subfield of AI. Scientists believed that modeling a small, specific set of knowledge (using an expert system) were more feasible goal to attain than modeling an open world, and were right. “Although a general-purpose problem solver still eludes us, expert systems function very well in their restricted domains.” (Giarratano and Riley 2005, 5)

Expert systems enjoyed a modest success in many academic fields throughout the 1970s, 1980s, and early 1990s. While expert systems are not currently at the forefront of the AI research agenda, expert systems are considered to be mature and reliable form of AI for constrained problems.

AI and expert systems have been used in geographic and cartographic research for many applications. “Within geography, the subfields of geographic information science (GISc), cartography, remote sensing, spatial analysis, and behavioral geography have

been fast to tap the potential of AI” (Mozolin 1997, 6). The earliest application of AI to geography and cartography can be found in the 1970s (Openshaw and Openshaw 1997). The applicability of AI to the field geography is generally agreed to be its applicable to geographic problem solving (Smith 1984). Up to the mid-1990s, expert systems had been the most popular type of AI application among geographers, and many geographers believed that expert systems and artificial intelligence were useful tools (Mozolin 1997). There have been many applications of AI and expert systems in cartography and geography. Cartographers had attempted to develop expert systems for automatic label placement (Christensen, Marks, and Shieber 1995; Doddi *et al.* 1997), cartographic generalization (Ying and Li 2005), spatial decision support systems (Armstrong *et al.* 1990; Arentze, Borgers, and Timmermans 1996), cartography (Bossler *et al.* 1988; Nyerges and Jankowski 1989; Murakami 1990), geographic information systems (Robinson, Frank, and Blaze 1986; L. Leung and Leung 1993; Fischer 1994), and thematic map design (Muller and Zeshen 1990; Skidmore 1996).

In the field of cartography, there have been multiple attempts and systems designed to provide cartographic expertise through expert systems. Bossler discusses two situations where expert systems were used to increase productivity, reduce subjectivity, and provide consistency of nautical charts (Bossler *et al.* 1988). The first situation pertains to the representation of shipwrecks on a nautical chart. Two expert cartographers were chosen to provide their expertise in an IF-THEN rule format for the expert system to ingest. It was found that while the conversion of expert cartographic knowledge to IF-THEN rules was easier than expected, the cartographers underestimated the number of rules required to accurately represent their expertise. The second situation

involved displaying overhead cables on nautical charts. The system was designed and found to be 93% accurate in the first trials. The authors also noted that entry level cartographers sought advice from the expert system rather than a supervisor so as to not appear unknowledgeable. The authors concluded that expert systems for cartographic design are useful for the purposes of facilitating cartography. This finding ties in nicely with the findings outlined by Novick and Ward (2006) concerning users' reluctance to use online help but feeling freer to solicit advice from an expert system.

One of the largest cartographic expert systems attempted was Map-Aid, which was designed with the purpose of assisting users in creating thematic map products (Muller and Zeshen 1990). This project was a huge undertaking and involved many agencies and researchers in the UK. However, this project was never completed. Other less ambitious, thematic map expert systems were also created, however these were often limited in scope and functionality (Howard 2004).

Revisiting the automation of cartography and its applicability to AI, the expert system saw success in partially automating some aspects of cartography. The last major works of expert systems in cartography concentrated on formalizing knowledge for cartographic expert systems performed in the late 1990s thru early 2000s (Forrest 1999a, 1999b, 1993). Forrest's research focuses on the classification and categorization of geographic knowledge for ingestion into a cartographic expert system. Forrest identifies three basic elements to be considered when describing spatial datasets: 1) nature of the phenomenon being mapped, 2) location information, and 3) nature of measurement of the characteristics. With these basic elements described, Forrest argues that a cartographer (or expert system) will be able to determine which cartographic representation(s) is/are

appropriate for the data. This research provides a starting point for the research of this dissertation as both Forrest's and this research have in common the need to first represent geographic information in a form that a computer can "understand". Forrest's research ultimately provides a classification of spatial phenomenon in order to formalize the process of describing data in the form of a taxonomy. This dissertation continues on the same path, but deviates when formally representing spatial phenomena in ontologies specifically for use in cartographic design decisions.

In addition to expert systems, machine learning, another aspect of AI will be utilized in this dissertation. Machine learning is the process in which a computer program infers new rules from experience. Tom Mitchell (1997) describes machine learning as a computer program that is said to learn from experience 'E' with respect to some class of tasks 'T' and performance measure 'P', if, its performance at tasks in 'T', as measured by 'P', improves with experience 'E'.

There are many machine learning methods. Some commonly used machine learning methods are neural networks, decision trees, supervised and non-supervised classification, and genetic algorithms. Within the machine learning methods, six machine learning paradigms exist. They are connectionist (neural network) learning, empirical learning (rule induction, decision trees), genetic algorithms and classifier learning, analytic learning (problem-reduction search, state-space search), and case-based methods (nearest neighbor) (Schlimmer and Langley 1991). This dissertation focuses on the empirical learning method, specifically, the construction of decision trees.

Empirical Learning

Empirical learning methods are inductive in that they move from specific data to some general rule or description. Empirical methods aim to move beyond training instances to make predictions about unknown cases (Schlimmer and Langley 1991). Empirical learning methods employ propositional knowledge representation for both experiences and acquired knowledge.

Propositional knowledge representation assumes that there exist objects (facts) in the world that are described by characteristics. These objects are represented by triplets in the form of object:property:value. “Earth:equatorial diameter:7926.41” miles is an example of such a propositional knowledge in triplet form. In this example, “Earth” is the object, “equatorial diameter” is the property, and “7926.41” is the value. Containing knowledge in this form allows for easy translation to table format and, thus, further translation into computer code by rule induction (Giarratano and Riley 2005).

To increase the complexity of a computer’s representation of reality, relational representations are used. A relational representation links multiple propositional knowledge triplets through a relationship. For example, “Megan and Maggie are sisters” represents the relationship between the two objects (Megan and Maggie) as being sisters. The use of propositional knowledge and relational representations provide the basic knowledge representation data structured used for empirical learning.

Multiple empirical learning methods exist in the field of AI. While many have been applied to geography, the Decision Tree empirical learning method will be focused on as it is directly applicable to this research.

Decision Trees

A decision tree is a predictive model that utilizes observations to make a linked, hierarchical set of decision points which are then used to derive a conclusion about input data. Decision trees offer many advantages including insight into data structure, hierarchical data representation, and ease of interpretation by users. Empirical learning methods are used to construct decision trees from observed attributes from training data. A decision tree is composed of two parts: branches and nodes. Branches represent groupings of dataset objects that share similar attributes. Branches make connections between nodes and operate as paths through the decision tree. Nodes represent locations in the tree. Nodes are pieces of information from which decision can be made.

There are three unique types of nodes present in every decision tree: the root node, the decision node, and the answer node (Giarratano and Riley 2005). The root node acts at the point of entry into the decision tree and resides at the top of the tree hierarchy. It is vital that a decision tree only have one root node as this allows for complete traversal of the decision tree as the root node is connected to every node in the decision tree. Decision nodes are decision points based on attributes in the training dataset. Each decision node has a test and connects to branches based on the result of the decision node's test. In simple decision trees, the test may be a yes or no question, such as "Is the slope greater than 10%?" In general, a decision node may use any criteria to determine which branch to follow, provided each selection process only yields one branch to follow. The next type of node, the answer node, is a node from which node branches lead to child nodes. Instead of a test condition, answer nodes contain a conclusion about the input data and serve as a termination point for the decision tree. A

decision tree will often contain many answer nodes, thereby allowing the decision tree to arrive at many different conclusions.

To illustrate the operation of a decision tree and how the nodes and branches work together to derive a conclusion, consider the heuristic, displayed in Figure 2.2, in determining what type of map is being presented.

```
IF the map focuses primarily on location of objects
    THEN the map is a General Reference Map

IF the map DOES NOT primarily focus on location of
    objects AND displays aspects of numerical data
    THEN the map is a Quantitative Map

IF the map DOES NOT primarily focus on location of
    objects AND DOES NOT display aspects of
    numerical data
    THEN the map is a Qualitative Map
```

Figure 2.2: Decision Tree Heuristic for Map Type Determination

The map type heuristic is visualized as a decision tree in Figure 2.3. Each decision node in this decision tree assumes the answer to each test is “yes” or “no”. Traversing the decision tree is a simple process, which is why decision trees are a popular choice with researchers. To use the decision tree to arrive at a conclusion, first, set the current location in the decision tree. If the current location is a decision node, then the test must be evaluated for an answer. Based on the answer to the test, the corresponding branch is followed to the next location. This process continues until an answer node is set as the current location. With an answer node as the current location, the conclusion stored in the answer node is returned as the decision tree’s conclusion.

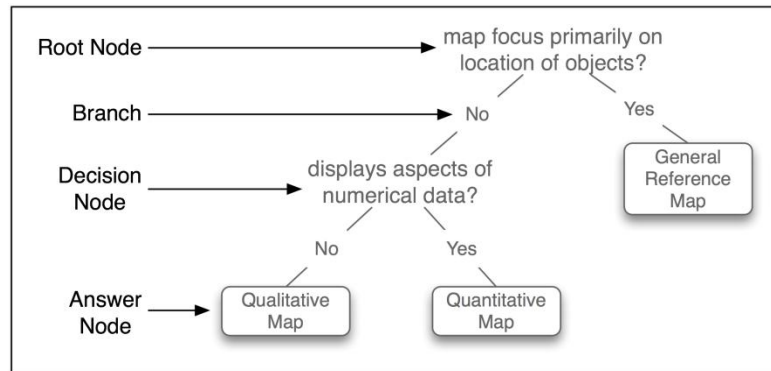


Figure 2.3: Decision Tree Anatomy

There are many types of decision trees employed in data mining. Of these types, the Classification and Regression Tree (CART) is used the most often in geographic research than other types of tree induction algorithms (Breiman *et al.* 1984). The advantage of CART is that it can deal with categorical and continuous dependent variables thus allowing for both categorical classification and regression in the data mining process.

Data Mining and Knowledge Discovery

Technology has enabled the collection of an unprecedented volume of data. Whether an image from a remote sensing satellite, or a demographic dataset from the U.S. Census Bureau, huge amount of new data are collected daily. Databases are constantly expanding to contain the abundance incoming data. This explosion in data collection can provide much potential for knowledge discovery; however, from this cache of data it is often difficult to discern information as traditional database tools only handle the storage and retrieval of data, not extraction of information. This problem calls for

new methods and tools that can transform data into information, which can then be synthesized into knowledge (Buttenfield *et al.* 2000).

A multidisciplinary approach by artificial intelligence, statistics, and database technology has yielded approaches in deriving useful information from the large databases of information – data mining. Data mining is the “non-trivial process of identifying valid, novel, potentially useful and ultimately understandable patterns in data” (Fayyad, Piatetsky-Shapiro, and Smyth 1996). Data mining techniques are inductive techniques meant to reveal patterns in the data being mined (Mennis and Liu 2005). Data mining’s inductive techniques extract potentially useful information from large data sets allowing for hypothesis generation and knowledge formulation.

There are two goals associated with data mining: prediction and description. Prediction is using variables to predict unknown values of target variables. Examples of data mining prediction tasks are classification and regression functions. Both of these tasks aim to learn and map data items into classes or variables. Description is the uncovering of interpretable hypotheses derived from describing the data. Examples of data mining descriptive tasks are clustering and summarization. Both of these tasks aim to identify a set number of categories that compactly describe the data.

As data mining deals with complex data sets, a number of supporting steps must be undertaken to prepare data for input and interpret the output data. The supporting steps surrounding (and including) data mining are known as knowledge discovery in databases (KDD). KDD is a secondary data analysis of databases where “secondary” refers to the database which was never originally designed for data analysis (Sowa 1998). The purpose of KDD is to perform secondary analysis on databases in order to transform

information into knowledge through uncovering new facets of the information and formulating new hypothesis, or patterns. To perform secondary analysis, several steps must be taken before and after data mining to uncover the patterns in the data.

KDD and data mining are distinct terms and describe different processes. KDD is the overall process of preparing the database, discovering useful patterns, and interpretation of the results. Data mining is only the selection and application of the data mining algorithm. Therefore, data mining is one of many steps in the overall KDD process.

The KDD Process

The KDD process is comprised of nine steps as discussed in Fayyad, Piatetsky-Shapiro, and Smyth (1996). The first step in the KDD process is the development of an understanding of the application domain, relevant prior knowledge, and goals. This first step serves to develop a well-formed research question. The second step is the creation or identification of an existing dataset. Should an entire dataset not be required, or unattainable, a subset of rows and/or columns may be extracted and used. The third step is cleaning and preprocessing the data. During this step, tasks such as removal of noise and outliers and handling missing data or fields should be undertaken. The fourth step is the transformation of the data into a data mining algorithm-friendly format. This is typically completed by restricting the number of variables to those which contain potentially useful information for the data mining task. The fifth step is choosing the data mining task (prediction or description) to be run on the data. With the task chosen, the next (sixth) step is to choose which data mining algorithm and parameters will be appropriate for the overall goal of the KDD process. The seventh step is to execute the

data mining algorithm to search for patterns of interest. Eighth is the interpretation of the results of the data mining through visualization and modeling of the results. Based on the results of this step, it may be required to revisit previous steps to make adjustments to the data, data mining task, or data mining algorithm. The final, ninth, step is comparing the discovered knowledge with existing knowledge and acting on the knowledge.

Table 2.1: Nine Steps of the KDD Process

Step	Action
1	Develop understanding of application domain and goals.
2	Creation or identification of existing data.
3	Cleaning and preprocessing the data.
4	Transformation of data into data mining friendly format.
5	Choose data mining task.
6	Choose data mining algorithm and parameters.
7	Execute data mining algorithm.
8	Interpretation of data mining results.
9	Compare discovered knowledge with existing knowledge.

For example, a problem that could be solved by completing the KDD process might be to find why a particular product is selling very poorly in a part of a city. The data analyst has to first identify and acquire a detailed database of consumer purchasing habits in the city that has been collected by retailers. Using the database, the data analyst will clean the dataset and select only variables and records that are significant for this study. Next, the data analyst will choose a data mining algorithm most appropriate to the type of problem and implement the algorithm. The output of the algorithm would be a hypothesis (or multiple hypothesis) which may be in the form of rules or labels describing the patterns uncovered in the data. With the hypothesis stated, the data analyst would verify and refine the hypothesis before finally interpreting the results and presenting them to the product sales manager. This process may require iteration through

the selection and application of the data mining algorithms to adjust for the characteristics of the data in the database.

Research and Uses of Ontologies in GIS

The term ‘ontology’ began as a philosophical device that “deals with the nature and the organization of reality” (Guarino and Giaretta 1995, 32). The field of AI co-opted the term ontology to act as an explicit formal specification of a shared conceptualization (Gruber 1993) and “a logical theory which gives an explicit, partial account of a conceptualization” (Guarino and Giaretta 1995, 32). In the philosophical field, ontology cannot be pluralized as it deals with the entirety of reality. In AI, ontology can be pluralized, and divided into multiple ontologies so that an individual ontology will contain a conceptualization regarding a specific object or knowledge area which can later be linked to other ontologies to construct a more complete conceptualization of reality.

A key difference between the philosophical ontology and AI ontology is the reason for defining reality. A philosophical ontology attempts to record and catalog all of reality by breaking it down into concepts, relations, and rules (Audi 1995) into a single ontology. In AI, ontology construction is for the designation and definition of terms and ideas to define a consensual foundation for inter-operability (Agarwal 2005). Additionally, ontology construction in AI aims for the general goal of interoperability and to achieve a widespread acceptance as a description of reality. This description of reality should be as complete, objective, and unbiased as possible. Multiple ontologies build a basis for the exchange of information by making vocabulary of a certain domain

of knowledge explicit (Initiative 2008), specify the kinds of concepts or entities that exist or may exist and relationships between them (Timpf 2002) and are used for specification of a conceptualization, typically specialized to one domain (Gruber 1993; Uschold 1996; Chandrasekaran, Josephson, and Benjamins 1999). An ontology containing an exhaustive list of concepts representing the world would be immense in size and likely never complete in both the AI and philosophical domains. As this complete concept of reality may never be reached, the more practical goal of defining top-level concepts of reality first, followed by more narrow concepts (linking back to top-level ontologies) seems to be the current and preferred practice in both the AI and GIS fields among many others.

The use of ontologies in GIS is a widely researched topic within the academy. In the year 2000, the University Consortium for Geographic Information Science (UCGIS) identified ontology as an emerging research theme (Mark *et al.* 2000). This UCGIS white paper called for the formalization of phenomena representations and discusses how the semantics of geographic information is critical for interoperability. Many research projects and papers about the use of ontologies in GIS have been completed in the 1990s and 2000s, and ontology research continues to be a healthy research area in GIS and other fields.

While research on the use of ontologies in GIS is seen as promising by many GIS scientists, there are those who have begun to critique ontology and its use in GIS. Kuhn for example discusses the nature and construction of ontologies and argues that to make geographic information more useful, ontologies should be designed with a focus on human activities. Kuhn raises concerns about ambiguities of natural language and

questions whether it is possible to model domains of knowledge independent of human activities (Kuhn 2001). His question of whether it is possible to model domains of knowledge objectively is appealing. For instance, if one were to attempt the formalization of the concept of a flower, how would they attempt this without having a bias of purpose? An individual could build an ontology describing the concept of a flower in every possible way, but 1) is this even possible? 2) is that much detail required? Additionally, if there is a need for an ontology, then a purpose must have already been identified, therefore bias is inevitable. While Kuhn's criticisms are valid and should be considered, there still exists a need for ontologies that are independent of human activity as to be accepted by as wide an audience as possible. While it is impossible to separate the conceptualization of reality from a human perspective, not every ontology should be linked to a specific task, as this would pigeon-hole ontologies into narrow use-areas, thus, contradicting the purpose of ontologies.

Several scholars interested in incorporating ontologies into GIS research are of the opinion that, while building a complete ontology is desirable, the purpose of the ontology should be evaluated as to contain the scope of the ontology, thereby saving time and money (Chandrasekaran, Josephson, and Benjamins 1998). The ontology constructor should still act as objectively as possible and build as complete an ontology as possible without exceeding the point of diminishing returns.

Another relevant research project sought to formalize the environment of an airport for an agent based wayfinding model (Raubal 2001). This project conceptualized an ontology of the airport, provided goals to the agent, and ran simulations to see where the agent (individual) gets stuck on its way from the ticket counter to gate. The

formalization of knowledge provided in this project allowed the computer to “understand” or, at least intelligently interpret its environment in order to achieve its goals.

Knowledge sharing is an active research area within the realm of ontology research. The heterogeneity of formats, systems, non-standard and ambiguous language, all pose challenges for scientists attempting to retrieve data (Lutz and Klien 2006). Currently, most geographic information portals index data based on information contained in associated metadata files. This method of search is often insufficient as the search algorithm relies on keyword searches of non-standard ambiguous language. Ontology and semantic reference systems can offer a solution to this problem, using ontologies to enrich descriptions of information services. Computer scientists as well as those in the GIS community are working on the problem of having too much data to effectively search through. There are some members of the computer science community who feel that the semantic web is the next step in the evolution of the world wide web (Berners-Lee, Hendler, and Lassila 2001). This vision is shared by some GIS scientists working towards information being machine-interpretable through the use of global and local ontologies, and a semantic searching interface (Lutz and Klien 2006; Lutz 2007).

CHAPTER 3

CONSTRUCTION OF THE MAP ONTOLOGY

Introduction

Working towards the automation of cartography requires the storage, retrieval, and application of cartographic knowledge by a computer program (in this case, an expert system). In order for a computer program to store and apply this knowledge, a set of concepts must first be defined so that the cartographic knowledge can be converted from human experts and cartography texts into a digital data structure that the computer can access. In addition to storing cartographic concepts, this data structure must also represent relationships and restrictions between the stored concepts. Building an ontology that represents the concept of a map and its related pieces assists the translation of knowledge between humans and computers, and, acts as a basic framework from which to build the digital data structures used by computer programs. By building a map ontology and subsequently a digital data structure that is able to store the cartographic knowledge, relationships, and restrictions provides the required knowledge an expert system needs to make inferences to automate cartography.

Overview of Ontologies

From domain-specific perspective, an ontology is a formal specification of a shared understanding of a knowledge domain that facilitates accurate and effective communications of meaning (Gruber 1993; Agarwal 2005). More directly, an ontology

is composed of a taxonomic structure which contains objects and their relationships to other objects within a particular domain of knowledge. In use, the ontology is a framework on which concepts can be formalized.

In understanding what an ontology is and what its function is, four key concepts must be discussed. The first idea, *concept*, means an abstraction of the world that is to be modeled. The world is abstracted as objects which have properties and relationships to other objects inside and outside the domain of knowledge being abstracted. Additionally, if we choose to specify constraints, we can construct an ontology that validates objects and their relationships.

The second idea, *formal*, means that the ontology is specified in a structured, logical, unambiguous fashion allowing for machine processing. This is accomplished through the use of a structured language built to be handled consistently and maintain structure.

The third idea, *explicit*, means that the ontology's concepts are clearly defined in a carefully selected vocabulary. The vocabulary should utilize words that describe the concept in the most precise manner and are commonly used by domain experts. Using unfamiliar or controversial words as the basis for the concept is counter-productive to the purpose of ontology construction.

The fourth idea, *shared*, means that the ontology represents concepts agreed upon by a community of experts. The word *shared* also refers to the formal language utilized as the ontology's dissemination vehicle. Using a formal language with a common syntax allows for the ontology to be shared between systems and users without losing structure or meaning.

Four broad categories of ontologies have identified by Guarino (1997): 1) top-level ontologies, 2) domain ontologies, 3) task ontologies, and 4) application ontologies. A top-level ontology covers a wide range of concepts independent of an individual problem. Top-level ontologies are concerned with abstract concepts and often serve as a foundation for more specific ontologies and sharing of knowledge across domains. Domain ontologies represent objects and their relationships within a specific domain of knowledge. The domain ontology is tasked with defining the vocabulary and relationships of the domain. Task ontologies deal with a specific task or activity within a certain domain. It is unlikely that the knowledge represented in the task ontology will be useful outside of its specific domain. Application ontologies link the domain and task ontologies acting as a mediator.

The map ontology will be constructed as a domain ontology. The domain ontology was chosen because a top-level ontology would have to be too general for our purposes, and the concept of a map is already (broadly) defined in some top-level ontologies. Our ontology will tie into the top-level ontology's map concept but provide more specific details.

In the context of this research, the ontology will serve as a specification of the concept of a map to facilitate knowledge acquisition from experts and design of the expert system to enable it to understand and intelligently apply the acquired expert knowledge. Development of an ontology has benefits to expert systems development. The first benefit, re-usability, shares its name with a major tenant of effective computer programming; re-use as much as possible and do not “re-invent the wheel.” An exhaustive search has been performed by the author in the attempt to find an existing

cartographic ontology, but the search yielded no immediately useable domain ontologies. The lack of immediately useable cartographic domain ontologies prompted this research, thus, this research will create the needed cartographic ontology and will be shared for future re-use and expansion by others.

The second benefit of ontology development, knowledge acquisition, speeds the acquisition of knowledge as the ontology provides a basis for knowledge engineers to ask questions of the domain expert.

Reliability is the third benefit and provides the automation of consistency checking, which would be difficult if done manually as ontologies can become quite complex. If assertions in the ontology are inconsistent (for instance, stating that a dog is both a type of plant and animal), then a computer reasoning with this inconsistent ontology may produce erroneous conclusions.

This research aims to have the ontology enable machine understanding of cartographic concepts with respect to the construction of maps. The development of this ontology will allow computers and cartographers to share their knowledge, thus making this knowledge accessible to a larger audience.

Methodology of Ontology Construction

Constructing an ontology is a multi-step process that is necessarily iterative and difficult to progress through linearly. In an effort to streamline construction of the ontology, multiple ontology development methodologies were researched and evaluated. Many of the ontology building methodologies shared similarities, and in the end, METHONTOLOGY (Fernandez, Gomez-Perez, and Juristo 1997) was the chosen

methodology as it incorporated the best practices elicited by many of the other methodologies.

In METHONTOLOGY, the development life-cycle is composed of specification, conceptualization, formalization, integration, implementation, evaluation and documentation. The ontology life-cycle does not specify the order in which the tasks should be undertaken. During the initial iteration of the ontology development, some tasks will naturally come before others, such as specification of the problem before implementation, however as the ontology is iteratively constructed, task order becomes less relevant. Each of the tasks will now be discussed as they were completed for creation of this map ontology followed by a discussion of the resultant ontology in the results section of this paper.

Specification

The process of creating an ontology initially involves consulting references (such as texts and experts) to a) locate previous attempts at constructing a related ontology, and b) frame the domain to be conceptualized. The specification task requires a document to be drafted in natural language that includes such information as the purpose, level of formality, scope, vocabulary, competency questions and sources of knowledge. This document guides the development process and allows for brainstorming and then bracketing of the domain to be modeled. This specification document provides guidance throughout the ontology development process by keeping the problem bounded and on topic. Multiple text sources and experts were consulted during the specification phase to define and bound the problem. Each section of the specification document will now be discussed.

Purpose: The purpose of the ontology is to describe the constructs required to build a map. It will assist a knowledge engineer in eliciting cartographic expertise from cartographers and texts to convert this information into a structured manner. This structured information can then be inserted into an expert system with the help of a computer scientist. The expert system will use the ontology, along with the inserted structured information, to maintain the structure of a map and determine appropriate selection, placement and symbolization of graphic variables. The ontology must also describe the interaction between the map, the objects which compose it, and the real-world phenomena that it will represent. Ultimately, the scenario in which the ontology will be used is to create the basis for questions to elicit knowledge from cartographers and to assist in building the framework for the expert system's facts and rules.

The audiences of this ontology are cartographers, computer scientists, and an expert system. With three audiences, the ontology needs to strike a balance between formality and accessibility. The ontology should be accessible enough to be understood by a cartographer with some study, and the help of a knowledge engineer. The ontology needs to be sufficiently formal for the computer scientist to encode the knowledge in a way in which the expert system can read. For the expert system, the ontology must be rigid, formal, and include even the most basic knowledge as it cannot be assumed that the expert system has any pre-existing knowledge in this domain.

Level of Formality: Four types of ontologies have been distinguished by Uschold and Gruninger (1996) and are: highly informal ontologies, semi-formal ontologies, formal ontologies and rigorously formal ontologies. A highly informal ontology is expressed loosely in a natural language, such as a narrative written in the

English language. This type of ontology is a poor fit as computers are not able to reliably and easily interpret such an ill-structured document and ambiguous language. A semi-informal ontology is expressed in a more structured form of a natural language with a well-defined vocabulary, however, this type of ontology, too, is a poor choice much for the same reasons the highly informal ontology type is a poor choice (ambiguous language, for instance). A semi-formal ontology is written in an artificial and formally defined language that is machine interpretable because of the strict structure and language requirements. This type of ontology is a good fit for computers as it is well structured and provides a bounded vocabulary. A *rigorously* formal ontology is a precise document formed by well-defined terms, semantics, theorems and proofs. This ontology, too, is a good fit for computers as it is well structured and provides a bounded vocabulary. Additionally, it includes theorems, proofs, and other supporting information.

For this research, the semi-formal ontology was the chosen type of ontology. A semi-format ontology can be interpreted by both a human and computer. The structured nature of this type of ontology provides many advantages for the computer. With a rigid structure, the computer can determine relationships and hierarchies, thus providing the information necessary to validate the ontology and represent knowledge in a useful manner. The bounded vocabulary reduces the number of words that are used to represent a topic being modeled, thus, removing the ambiguity of natural language. In addition to advantages to the computer, the semi-formal ontology offers many benefits to humans. Again, the rigid structure that this type of ontology requires, allows the human to model and determine relationships and hierarchies as they exist in the knowledge being

modeled. The bounded vocabulary requires that the human removes natural language ambiguity thereby allowing the knowledge to transfer easily to the computer.

A *rigorously* formal ontology (being the most formal type of ontology) would also be a fit for this research, however, with cartography being a domain of knowledge based on general rules and experience instead of rigid rules (such as mathematics), proofs and theorems do not necessarily apply in this knowledge domain. Additionally, a rigorously format ontology provides many more barriers to the human, which, considering the three audiences that need to be able to understanding the ontology, may not be accessible.

Scope: Scope defines the context and domain in which the ontology will operate. It is important to have a bounded scope and domain as to keep the ontology focused. The map ontology is bounded to a tangible or virtual form that can be viewed by multiple people. Additionally, the ontology will only cover the basic structure of a map and the representation of space thereon. The level of granularity is purposely coarse as to allow flexibility in subsequent contributions to the ontology.

Vocabulary: To assist in determining the scope and requirements of the ontology, a list of terms of objects or concepts a cartographer might use when designing a map was compiled by referencing cartography texts and cartographers (See Appendix A). Once the terms were identified, they were classified into logical groups and served as an initial structure of the ontology. These terms were also used as a basis for determining how the elements interact with each other to compose a map, and formed a basis for questions to start building the expert system's knowledgebase.

Competency Questions: The role of the competency questions are to define the scope of the ontology and serve as a set of guiding thoughts when constructing the ontology. The ontology is deemed successful if it can be used to satisfactorily address the questions. For the map ontology, four competency questions were established:

- What are the components of a map?
- How do map components relate to compose a map?
- How do objects in the real world become transformed to representations on the map?
- What visual elements are applied to representations of objects?

Sources of Knowledge: Multiple cartographers, maps, cartography texts, and ontologies were consulted in determining the terms and scope of the ontology. A wide range of sources was deemed to be the most appropriate as the aim of the map ontology is to capture the basic concept of a map. In order to determine the vocabulary, questions, and scope, many different sources of knowledge were analyzed to find commonalities that could be abstracted into the ontology.

Knowledge Acquisition

The knowledge acquisition step extends across the entire ontology development life cycle. Initially, knowledge acquisition is used to create the initial specification of the ontology. This can be done through brain storming, interviews, literature review or other useful methods. Later in the ontology development life cycle, knowledge acquisition provides the ontology validation service. In this research, the knowledge acquisition step progressed in parallel with the initial specification step and ran concurrently throughout the remainder of the ontology development task.

Four relevant sources of information and knowledge were identified for development of the map ontology: expert cartographers, maps, cartography texts and ontologies. Each of these sources of information was tapped throughout the construction of the ontology and was used to validate the ontology. Additionally, each source was used to corroborate knowledge acquired from the other sources.

The first source sought after was existing map ontologies. A search for existing map ontologies bore few results and only one that provided enough detail to be useful as a source of information. This ontology was in the form of a conference paper and provided a skeleton for a basic domain cartographic ontology (Iosifescu-Enescu and Hurni 2007) and yielded some useable information and structure. Even though the ontology was not complete, it added to vocabulary and structure for development of the map ontology.

Next, discussions were conducted with cartographers to build upon the preliminary specification document and vocabulary. The concept and construction of a map were discussed to determine what knowledge and information a cartographer must possess to construct a map. These discussions also helped to form the competency questions.

After the discussions, the specification document and vocabulary were expanded upon using informal analysis of texts to further flush out the structure of the specification document. The knowledge gained up to this point was then evaluated against a plethora of maps as a parity check and also to identify areas that had been ill-defined in the specification document.

The final step was taking the vocabulary, specification document, and initial constructed ontology back to the expert cartographers for another round of discussions and evaluation of the ontology.

Conceptualization

The conceptualization step deals with the structuring of the domain knowledge discovered in the specification step. The vocabulary was expanded to a complete glossary of terms which included the acquired vocabulary, verbs and properties. The glossary was compiled from the specification document and initial vocabulary and then was structured more formally for use in developing the ontology. The conceptualization step started and ran concurrently with the second half of the specification step.

Integration

The integration step proposes the use of other ontologies to speed the construction of the new ontology. The use of a meta-ontology, such as OpenCyc (CyCorp 2010) is recommended as meta-ontologies cover a large range of topics and provide a standard set of terms used by other ontologies. This step started concurrently with the initial specification step and yielded one domain ontology (previously mentioned) that provided some basis for a starting point.

Implementation

The implementation step is where the ontology is developed using a well-structured format. The web ontology language (OWL) was chosen which is recommended by the W3C (W3C 2004) and requires structure and semantics on objects and supports the specification relationships between objects specified in the language. OWL and RDF are well established, widely used and can support a formal ontology.

Additionally, OWL is stored in XML format which can be consumed by the expert system quite easily.

To build the ontology, the software program Protégé (<http://protege.stanford.edu/>) a free, open-source ontology editor was used. Protégé fully supports OWL and provided the functionality required to create this ontology. Additional functionality of Protégé that was useful was lexical and syntactic checking, search functionality, and visualization.

Evaluation

Evaluation step is where a technical judgment of the ontology is performed with respect to the domain of knowledge being modeled in the ontology. This step requires verification of the correctness and the validation of the structure of the ontology. The ontology was verified by expert cartographers, cartography texts, and maps. The ontology structure was validated by the software program Protégé. The evaluation step was performed throughout the ontology development life cycle in an effort to catch incomplete, redundant and inconsistent knowledge and structures.

Documentation

The documentation step requires explanatory documents to be created throughout the ontology development life cycle. The purpose of documentation is to record decisions and the information those decisions were made with respect to for future reference and justification. The documentation for the map ontology takes many forms such as the specification document, vocabulary, internal explanatory documents, reference managers and documentation within the ontology itself.

Results: The Map Ontology

In this section, the map ontology developed in this study will be discussed first as a whole, and then by the eight top-level classes of objects. Before we discuss the formal ontology in detail, however, some introductory terms and concepts need to be presented so the reader can follow the discussion.

Formal Ontology Concepts

Class: A class represents tangible or intangible things that exist. Classes state the requirements for which a thing would be considered to be a type. For instance, if we had a class to represent a dog, the requirements would be (a) animal, (b) has four legs, (c) barks, and (d) warm blooded. A dog that exists would then be considered to be a type of dog as specified by the dog class.

Property: A property is a relationship between two classes. A property may be used to define restrictions between classes as well, such as restricting the cardinality.

Inheritance: In an ontology, classes are placed in a hierarchical structure with respect to whether a class is a subclass (child) of another superclass (parent) class. A child class must have all of the properties of its parent class and must be more specific of a class than its parent without exception.

Top level classes of the map ontology

The top-level (most general) classes in the map ontology are: Attribute Element, Graphic Element, Layout Element, Map, Map Scale, Map Projection, Production Medium, Spatial Phenomenon, and Visual Variable. These nine classes encompass all of the concepts and information required to consider an object to be a map. It is important

to keep in mind the purpose, scope, and competency questions of the ontology when evaluating the grouping of concepts in the ontology.

When exploring the ontology, any class could be an entry point as all classes are related, however, when thinking of the concept of a map and the components of a map, the natural point of beginning in the ontology is the Map class. For the discussion of this ontology, the Map class will be the focus and point of beginning for discussion.

Overview of the Map Ontology

To assist in gaining an overview of the map ontology, an overview of the map ontology will be presented to provide a general view of how the classes relate and interact to form the concept of a map.

Figure 3.1 illustrates how the top-level classes and a few important subclasses work together to construct the concept of a map. A map (represented by the Map class) is a communication device that displays representations of Spatial Phenomena. The Spatial Phenomena are represented by graphics such as points, lines, polygons, and rasters and visualized with Visual Variables such as color and opacity. Spatial Phenomena are described by Attributes which are visualized by Labels. Layout Elements are placed on a Production Medium, such as paper. The Map Body class (subclass of Layout Element) contains at least one Visual Variable. The Map Body has one Map Scale dictating the ratio at which the Spatial Phenomena have been reduced to display on the Map Body. The Visual Variable is placed on the Map Body using a Map Projection.

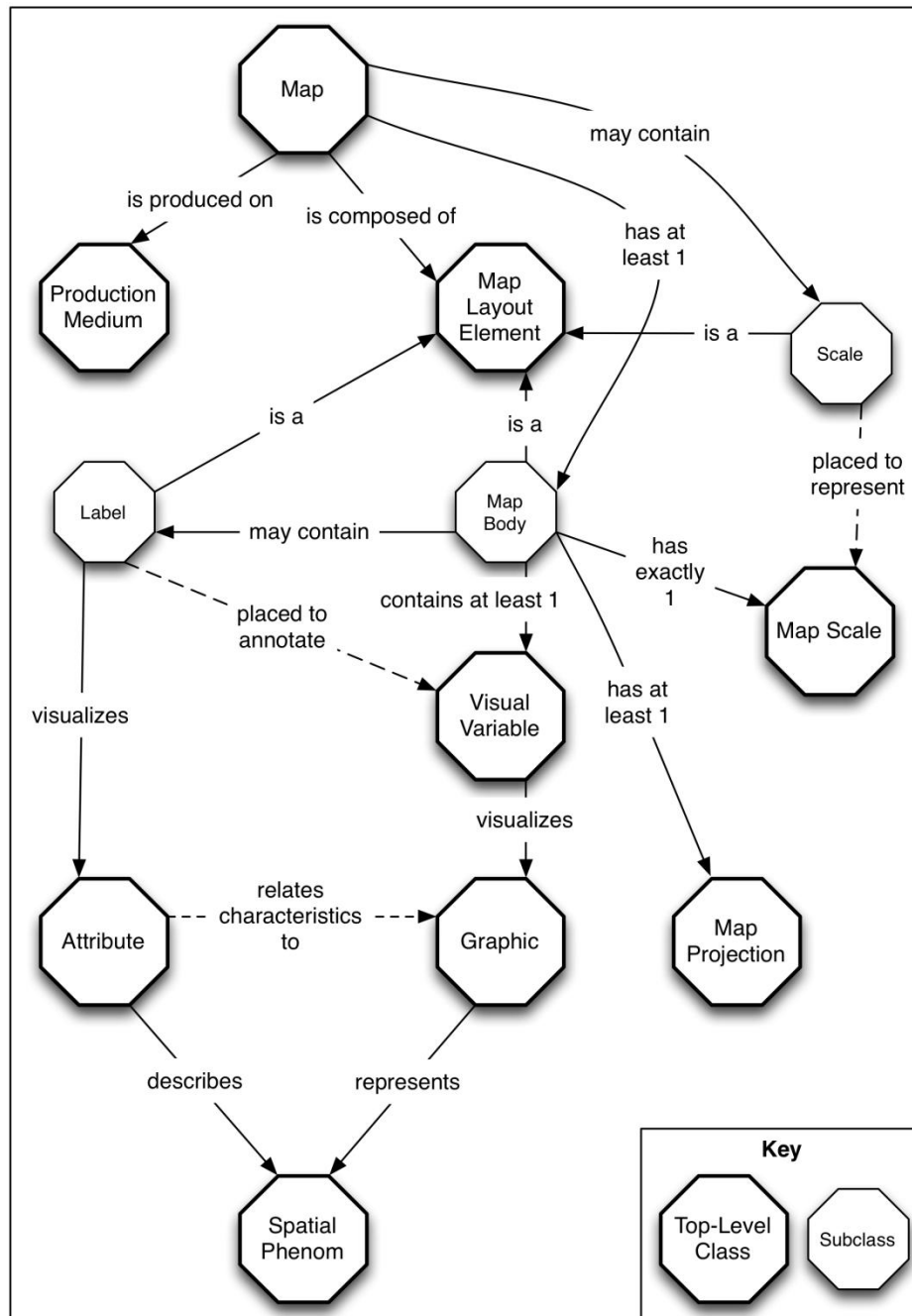


Figure 3.1: Overview of the Map Ontology

The Map Ontology

The discussion in the preceding section provided a general overview of how the top level classes in the map ontology work together to build the concept of a map. This

section will discuss, in detail, each of the 56 classes in the map ontology and provide context for why the class was chosen and why it is necessary for inclusion in the Map Ontology. Again, while there are multiple points of entry for the map ontology, the discussion will attempt to work logically through the classes and how they relate back to the overall concept of a map.

To aid in readability, classes and properties are written in the Courier New font ‘Like This’. Additionally, each figure displaying how classes in the Map Ontology related should be read from left to right, with the left-most oval representing the highest, most general, class in the ontology’s hierarchy, and the right-most oval representing the lowest, most specific, class in the ontology’s hierarchy. To reinforce the hierarchy and parent-child relationships, the figures displaying the hierarchy are also illustrated with directional arrows annotated with the phrase “is-a”, which is short for “is a type/subclass of.”

Thing: The `Thing` class (Figure 3.2) is the base class for every ontology stored in the OWL language. The `Thing` class represents some object or concept that exists. The `Thing` class is purposely broad as it serves as a single point of beginning for an ontology of which every other class is a type. Additionally, the `Thing` class serves the important task of providing a way for one ontology to relate to other ontologies. If both ontologies have the same base class, then the two ontologies can bind to a common class. Additionally, within the Map Ontology, the `Thing` class provides a way for subclasses of the `Thing` class to relate to each other. Also, for the computer to traverse from one class to another, having all classes inherit the `Thing` class provides a convenient tree structure for traversal.



Figure 3.2: The `Thing` Class

Map: The `Map` Class (Figure 3.3) is the container for the basic concept of a map. All other classes in the ontology will related directly or indirectly to the `Map` class. In order for a thing to be considered a map, it must communicate something about a spatial phenomenon on a physical medium and be viewable by multiple people. This communication can range from a simple map showing the path from a store to a house to a very complex map showing the economic interactions between countries. Whether the map is simple or complex, all maps have a basic requirement before they are considered to be a map. The requirement in the ontology is that the map shows a representation of at least one spatial phenomenon (bona-fide or fiat) on a medium visible by other people. Whether the map communicates effectively is irrelevant as even poorly designed maps can still meet the requirements to be considered a map, and, therefore, is not considered a requirement in the ontology.

In the map ontology, the `Map` class has two children: `ReferenceMap`, representing reference maps, and `ThematicMap`, representing thematic maps. These two types of maps meet the basic set of requirements discussed above, but display different facets of spatial phenomenon. Reference maps display objects from the environment and show features of spatial phenomenon. Thematic maps present a graphic theme about a physical or abstract subject. Thematic maps are further subdivided as qualitative and quantitative depending on which type of attribute is being mapped (Muehrcke and Muehrcke 1998; Dent, Torguson, and Hodler 2009).

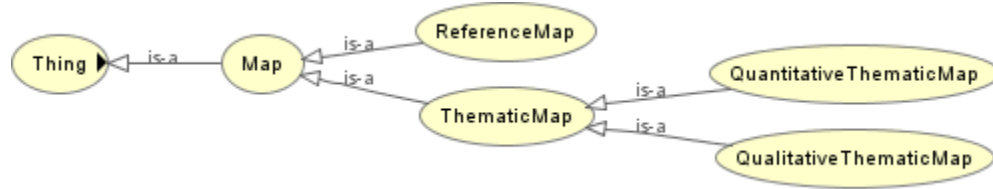


Figure 3.3: Map Class and Subclasses

MapProjection: The MapProjection class (Figure 3.4) represents the task of placing the representations of spatial phenomenon on a surface for visualization. Any surface for visualization requires a projection whether it is a flat piece of paper or a virtual globe. Every map body (some maps containing more than one map body) must have exactly one map projection making the map projection a requirement for a map.

In the map ontology, properties of map projections were chosen as the delimiter for child classes. The five properties used are: azimuthal, conformal, equal area, equidistant, and compromise (Snyder 1987). These properties are widely used and recognized which is the main reason they were the chosen delimiters for classes. Each of these five properties are subclasses of the MapProjection class using their respective property names.

Although a map body must contain at least one map projection, but there are some map projections that contain multiple properties. To accommodate this situation, the Compromise class can contain elements from zero to three of the other MapProjection classes, but cannot contain all four classes at once.

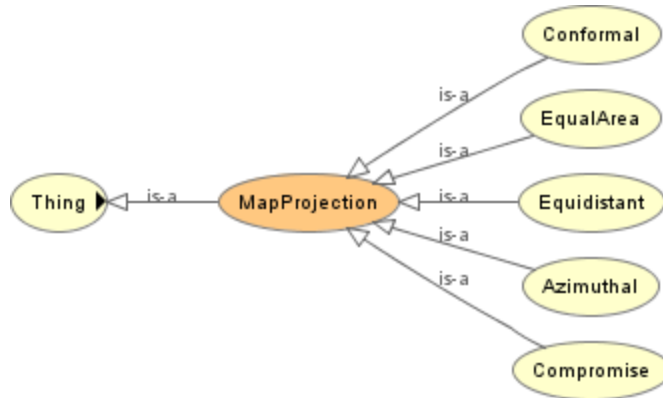


Figure 3.4: MapProjection Class and Subclasses

MapScale: The MapScale class (Figure 3.5) represents the ratio at which the Spatial Phenomena have been reduced to display on a map. Each MapBody has exactly one MapScale. The MapScale is often a function of the size of the production medium of the map. In the case of a paper production medium, the map body will have one static scale. If the map is produced digitally, the map scale may change depending on whether the user is provided the tools to increase or decrease the map body's scale. The Scale LayoutElement may be placed on the Map to reference the scale of the MapBody.



Figure 3.5: MapScale Class

ProductionMedium: The ProductionMedium class (Figure 3.6) represents the medium upon which the map will be produced for dissemination. As the scope for the map ontology is restricted to physical or virtual maps that can be viewed by

multiple people, the map must be produced on viewable medium. Subclasses were not added to this class to provide flexibility for future expansion. While it would be easy to define a “digital” subclass for maps stored in digital format, classifying “non-digital” maps is more problematic. A “non-digital” map could range from a map drawn on paper to a map carved into stone. A problem occurs when trying to classify such a wide range of possible map mediums generally. Are paper maps and rock maps to be classified together as “analogue” maps? Is it fair to define the medium of a map based off of its difference with respect to another medium? For instance, is the class of “non-digital” considered an adequate class which is based on its children simply not being digital? Another possibility is to create two classes of “concrete” and “digital” maps. This causes a problem in considering whether a hard drive containing digital bits is still a concrete item. Based on these philosophical issues and the potential rabbit hole dealing with this particular class, the choice to keep it general was considered the best option.

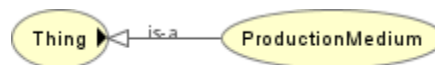


Figure 3.6: ProductionMedium Class

MapLayoutElement: The MapLayoutElement class (Figure 3.7) represents the objects and containers that construct the map. The individual layout elements, working together, construct the concept of a map (represented by the Map class).



Figure 3.7: MapLayoutElement Class and Subclasses

The layout element class has eleven subclasses: MapBody, Legend, DirectionalIndicator, Metadata, Neatline, Scale, Title, Label, AncillaryText, AncillaryObject, and Graticule. Of the eleven classes, only one, PrincipalMapBody (which is a child of MapBody), is required by the Map class. While it is preferable for additional layout elements to be used on map to provide supporting information for the map body, the scope of the ontology set forth earlier dictates that the most basic type of map be conceptualized; thus a map consisting of a single map body that contains at least one visual variable would be the most basic type of

map and supported by this ontology. To illustrate this point further, Figure 3.8 displays an example of how a map is composed using the concepts represented in this ontology.

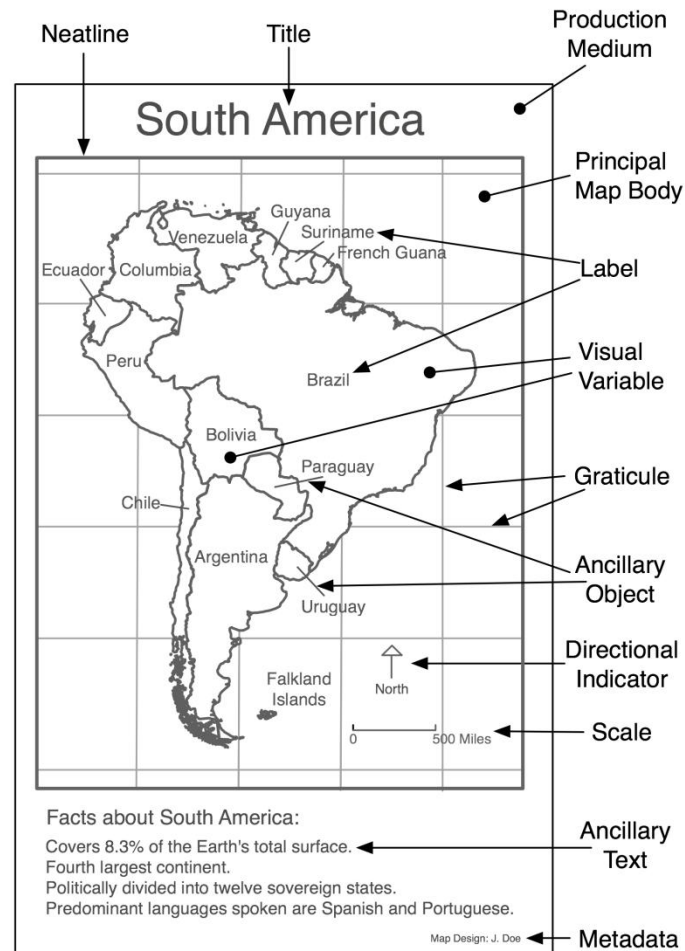


Figure 3.8: Map Composition using Concepts in Map Ontology

The eleven subclasses were chosen based on a review of multiple texts. A large number of texts created a consensus on what elements are required to construct a map; therefore those identified map elements determined the classes. Each one of the classes will now be reviewed.

`MapBody` represents a map body layout element. The purpose of a map body is to contain at least one visual variable which visualizes the graphic representation that, in turn, represents the spatial phenomenon being mapped. `MapBody` has three child, equivalent classes: `PrincipalMapBody`, `IndexMapBody`, and `InsetMapBody`.

`PrincipalMapBody` is the focal point of a map. As the principal map body is the focus of a map, and is the basis on which other layout elements are created, the `PrincipalMapBody` is the only class required by the `Map` class. A `Map` may contain more than one principal map, but at least one principal map is required to construct the concept of a map.

The `InsetMapBody` represents an inset map. An inset map, at its basic conceptual level is considered a map, however, an inset map takes a supporting role to the primary map body.

The `IndexMapBody` represents an index map. An index map displays where a principal map geographically covers with respect to a larger, encompassing, geographic area.

`Title` represents the title of a map. The purpose of a map title is to draw attention to the primary content of the map.

`Legend` represents the map legend. A map legend describes and contains a sample of a visual variable found on a map body.

`DirectionalIndicator` represents a graphic placed on a map to illustrate a particular direction so the map user can orient the map appropriately for their needs. Typical examples of a directional indicator are a north arrow, compass rose, and a magnetic north declination graphic. The name of this class deviates from the name of

layout elements found in the majority of the texts reviewed. In the texts, “North Arrow” was the most commonly used term when discussing this concept. While a north arrow is a very common directional indicator, using this term shows too much bias towards only one of the possible types of directional indicators. Therefore, the term “Directional Indicator” was chosen as it includes the multiple types of graphics that show a direction without being too specific and restricting.

`Scale` represents a scale which conveys the rate at which the spatial phenomenon being mapped has been enlarged or reduced. The `Scale` class is composed of three subclasses: `GraphicalScale`, `RepresentativeFractionScale`, and `VerbalScale`. `GraphicalScale` represents the graphic scale, which shows a reference graphic and text to be used for measuring based on the size of the scale graphic.

`RepresentativeFractionScale` represents a representative fraction text expressing the rate of enlargement or reduction. `VerbalScale` represents a verbal scale which is text explaining the rate of enlargement or reduction.

`Neatline` represents a neatline used for framing of layout elements. A neatline must encapsulate another layout element, otherwise it would be considered decoration, and, thus, a member of the `AncillaryObject` class.

`Label` represents labels placed on the map. A label will visualize the text stored in an attribute on the map. Typically a label will be placed in a way in which the map reader will related the attribute label to the visualized graphic, but this is not a requirement of the `Attribute` class or `Label` class.

`Metadata` represents metadata (descriptive information) with regards to the map itself. Examples of metadata elements are: map author name, map production date, map projection, and copyright information.

`Graticule` represents a grid overlaid on the a map body that shows references lines related to lines of constant coordinate value or delineate a grid reference system.

`AncillaryText` refers to explanatory text displayed on the map. Such text could be descriptive or flavor text aimed at supporting the message of the map.

`AncillaryObject` refers to objects such as graphs, photographs, and videos placed on the map. Such objects support the message of the map.

SpatialPhenomenon: The `SpatialPhenomenon` class (Figure 3.9) represents all bona-fide and fiat objects suitable for mapping. These spatial phenomena are represented abstractly by humans and are stored in graphic representation form for modeling and visualization. For the purposes of mapping, spatial phenomena are divided into two child classes relating to the way in which they are abstracted: `Discrete` and `Continuous`. Discrete objects have clearly defined boundaries and location. Continuous objects do not have clearly defined boundaries and



Figure 3.9: `SpatialPhenomenon` Class and Subclasses

Graphic: The `Graphic` class (Figure 3.10) describes the graphic devices uses to represent and store the abstract representations of spatial phenomena. Graphics are the drawing primitives available to a cartographer when creating a map. The primitives are point, line, and polygon (vector) and matrix (raster). These primitives are mirrored in the map ontology as subclasses of `Graphic`.

Graphics do not hold any visualization information other than basic shape. The basic shape may be constructed from coordinate pairs, by the hand of a cartographer, or both. The pairing of the graphic and visual variable provides the vehicle for visualization on the map body by the cartographer.

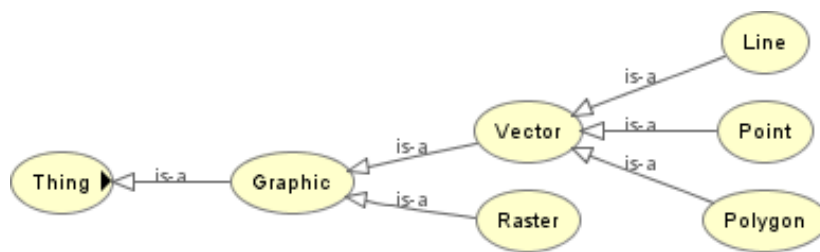


Figure 3.10: Graphic Class and Subclasses

VisualVariable: The `VisualVariable` class (Figure 3.11) represents the concepts cartographers use to visualize the graphics (which abstract reality) onto the map body. When used together, or separately, “these symbols serve as visual variables from which the reader gathers information and interprets the map” (Dent, Torguson, and Hodler 2009, 71). This class has nine children covering the most widely accepted visual variables: Arrangement, Focus, Hue, Orientation, Saturation, Shape, Size, Texture, and Value.

`Arrangement` class represents the arrangement visual variable that is used to represent the location of objects. Bertin first identified this visual variable as the “planar dimensions” (Bertin 1983). The planar dimensions represent the X,Y positioning in two-dimensional space. For contemporary applications, the third dimension, Z, should also be considered as part of this arrangement visual variable. However, for the purposes of this map ontology, no stance on the number of dimensions is taken to allow for maximum flexibility and future expansion.

`Orientation` class represents the orientation visual variable. The orientation visual variable represents the rotation of an object. Additionally, this visual variable can be used to create a perception of groups, or likeness of objects based on uniform orientation. Conversely, misaligned objects can create a perception of disparity.

`Size` represents the size visual variable. Size represents the size of the object and can be used to imply relative levels of importance.

`Shape` represents the shape visual variable. Shape is used to display similar elements and facilitate object identification (Bertin 1983).

`Texture` represents the texture visual variable. A texture is usually a variation in dot or line density and arrangement placed as an overlay.

`Focus` represents the focus visual variable. The focus visual variable provides for crispness of edges of an object. Focus is often related to representations of uncertainty.

`Hue` represents the hue visual variable. Hue can be thought of as the name of a particular color pulled from the visible portion of the electromagnetic light spectrum.

`Saturation` represents the saturation visual variable. Saturation is the dominance of the hue. This ranges from a pure hue, to no hue where no hue dominates (white).

`Value` represents the value visual variable. Value is the intensity, or strength of the light; ranging from light to dark.

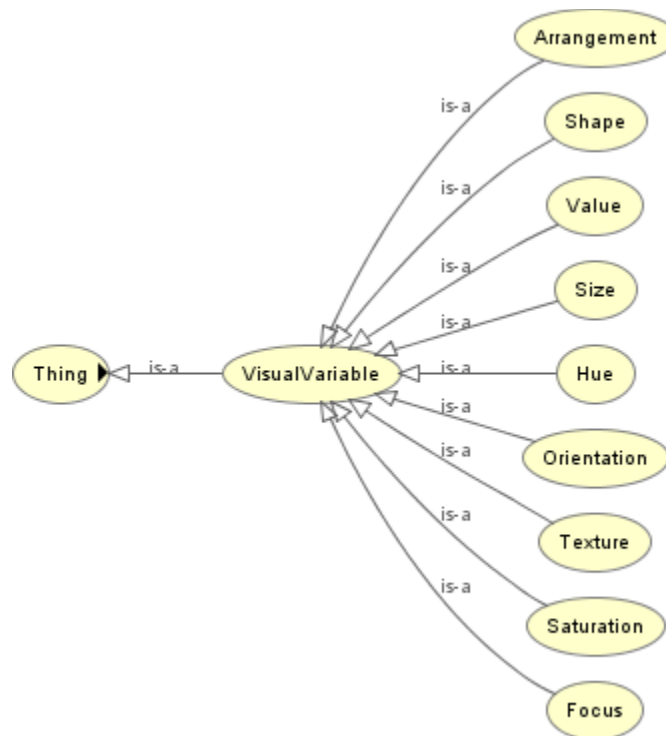


Figure 3.11: `VisualVariable` Class and Subclasses

Attribute: The `Attribute` class (Figure 3.12) contains the descriptive information related to spatial phenomena. Attributes may be visualized on the map body through the `Label` class. Four subclasses, `Nominal`, `Ordinal`, `Interval`, and `Ratio` compose the `Attribute` class.

Nominal represents nominal attributes. A nominal attribute value contains a descriptive value. These descriptive values can be used to only describe and determine different kinds of things. Mathematical and Boolean operations are not possible on nominal attributes.

Ordinal represents ordinal attributes. An ordinal attribute contains a rank value. This rank value can only be placed in a hierarchy and only determine rank and not magnitude between values.

Interval represents interval attributes. An interval attribute value contains rank and magnitude information; however, there is no natural origin.

Ratio represents ratio attributes. A ratio attribute value contains both rank and magnitude information. Additionally, the ratio magnitudes are based on an absolute zero value as its starting point.

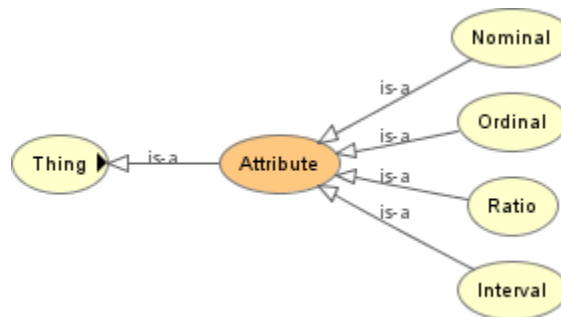


Figure 3.12: Attribute Class and Subclasses

Chapter Summary

The development of this map ontology plays an important role in working towards the partial automation of the map production process. Formalizing the vocabulary, structure, and relationships of a map for use by a computer program provides

a foundation for software developers, cartographers, and software engineers to build systems upon.

Choosing to represent the complex subject that is a map in an ontology provides many advantages. Of the many advantages, two significant advantages are software independence, and accessibility. Software independence means that the knowledge of maps and mapping are represented independently of any software package. This ontology was kept general enough to be used by many other technologies in various situations. Accessibility means that formalizing the concept of a map in a structured, well-defined and widely used language will facilitate shared expansion of the ontology. Changes and improvements by the community will benefit those interested in modeling the map.

CHAPTER 4

DATA MINING GEOSPATIAL METADATA

Introduction

To automate as much of the cartographic design process as possible, user input should be kept to a minimum. One key area where the user would typically need to provide input to the cartographic expert system is setting the theme of the dataset. The theme of the dataset (road, park, political boundary) is what the expert system will utilize to determine the appropriate symbol for rendering the dataset on the map. Currently, there is no algorithm or tool available that automatically determines the theme of an input dataset. Since no algorithm or tool exists, and a goal of this research is to identify and symbolize a dataset automatically, a new algorithm for automating theme identification was created and is discussed in this chapter.

Selecting the Inputs to Data Mine

To work towards the automation of detecting the theme of a dataset, the research question “what information can be found in this dataset that would assist the task of identifying its theme?” was formed. Having the computer emulate the identification process that a user makes through examining a geospatial dataset is the ultimate goal of this portion of the research. However, before emulating the identification process made by a user, determining what the inputs that the computer will use to autonomously identify the theme of the geospatial dataset must be decided. The challenge is having the computer

autonomously identify a dataset's theme based only on the dataset itself and parameters of the data mining algorithm.

Relying on the dataset to identify theme poses the question, "what parts of a geospatial dataset are good candidates for providing nominal information useful for theme differentiation by a computer?" To answer this question, the format of a typical geographic dataset must be considered. The majority of geospatial datasets are segregated into three major parts: the geometry, the attributes, and the metadata. The geometry represents the spatial conceptualization of reality modeled in a discrete or continuous fashion, such as a vector, or raster format. The attributes are the non-spatial characteristics often stored in a tabular format. The last part, metadata, is essentially ancillary data about a data set. It contains a wealth of information such as keywords, purpose, contact information, lineage, and attribute field names to name a few. In addition to the three major parts of a geospatial dataset, there are other, minor/supporting parts that include information such as attribute field indices, geometry indices, and coordinate system information, among other information.

While the geometry portion of a dataset holds the spatial representation and might initially seem a good candidate for determining a dataset's theme, its purpose is simply to define the locations where features exist and the geometry portion itself does not provide any descriptive information. As an example, two points from two separate datasets may have identical coordinates, but one point represents a coffee shop, and the other point represents the location of a wifi hotspot; the geometry part holds no descriptive information.

The attribute table does have the potential to provide a large amount of uniquely identifying information per dataset. However, it was not chosen as a source for two reasons: 1) the information stored in the attribute table may be *too* specific for a dataset to match with other datasets of the same theme. For instance, consider two road datasets covering two cities of relatively the same size. The attribute tables for both road networks contain road names and road lengths. The road lengths are not likely a useful attribute as the length of a road (simply seen as a decimal number to the computer) would not provide a meaningful data point for comparing whether other attributes from other datasets share the same theme; length would only verify that two numbers from two different datasets match. The street name attribute will probably yield more matches between the two cities, but it is not possible to state that two road names (again, seen as letters by the computer) that match between cities is enough evidence to point towards a similar theme. There are a few cases in which an attribute may contain information that could be used to identify a similar theme. For instance, the suffix of a road (*e.x.* St., Dr., Road) could be used to infer a similar theme, however, this assumes that the suffix is separated into a separate field and a majority of datasets contain some field that contains some common descriptor. In short, the values stored in attribute tables serve to differentiate between records in a dataset, but are typically not stored to make any statements about the larger point, or theme, of the datasets that contain them.

Taking the shortcomings of the geometry and attribute portion of a geospatial dataset into account, what is needed for the computer to make suitable identifications of dataset themes are information whose express purpose is to educate the user on the facets of the dataset. Metadata is the natural choice for dataset theme identification as the

purpose of metadata is to describe facets of the dataset to users and to differentiate one dataset from another. The major uses of metadata are to organize, maintain, catalog, and aid in data transfer (Hand 1998; General Geographic Data Committee 2000). The information stored in the metadata provides descriptive, information in a structured format which makes it ideal for automation, and, therefore, is considered a good source for potentially useful information in automating theme identification. Specifically, keywords, and field names found in metadata documents of geographic datasets were targeted as candidates for data mining. Keywords and field names were chosen as they provide short, concise words with the purpose of describing the contents of the dataset and are easily consumable by data mining algorithms.

Note that machine learning and pattern matching is not expressed as a major use of metadata, thus, this research will be utilizing metadata in a novel way. Exploring the relationship between knowledge discovery techniques and metadata can yield knowledge which will assist in the automation of the identification of a dataset's theme.

With metadata being identified as the part of a geospatial dataset that is the best candidate for input into the data mining algorithms, the question of which part of a dataset is useful posed at the top of this section is now answered. This leads us to a new question that will be addressed in the results section: “do metadata of freely available GIS datasets provide sufficient information for automating the identification of datasets commonly found on general reference maps?”

Methods

Fayyad, Piatetsky-Shapiro, and Smyth (1996) identified nine steps that comprise the knowledge discovery in databases (KDD) process (of which data mining is one step of the nine). These nine steps are introduced in Chapter 2. Data mining is the 7th and “core” step of the KDD process that derives the knowledge to be interpreted for fitness of use. For a complex knowledge discovery task like the one in this research, the preceding six steps must be completed in preparation for the data mining and the following two steps will provide the interpretation and evaluation of the results. The first seven steps will now be discussed, and the last two steps, interpretation and knowledge, will be discussed in the results section of this chapter.

KDD Step 1: Understanding the Application Domain

This step has been completed and discussed in the preceding section. It was decided that metadata of geospatial datasets provide information likely to be useful in the data mining process.

KDD Step 2: Identification and Collection of Existing Datasets

As this research is restricted to general reference maps (for reasons discussed in Chapter 1), datasets that would likely be used by the general public to make maps were targeted for data mining. Eight requirements were defined for the purpose of determining whether a dataset would be included in the population of input into the data mining algorithm. The eight requirements are:

- 1) Data must be a base map dataset commonly found on a general reference map.
- 2) Data must contain completed metadata.
- 3) Data must be freely available to the public, or, widely used by the public.

- 4) Data must be of vector type (point/line/polygon).
- 5) Data must be in digital format.
- 6) Data must contain only one theme of information.
- 7) Data may only cover the geographic extent of the United States of America.
- 8) Data and metadata must be in the English language.

Each of these requirements will now be discussed in detail.

Requirements

Requirement 1 – Base Map Dataset

Base map data commonly found in a general reference map are the focal datasets. Eleven dataset themes were targeted for this research: airports, boundaries, contours, fire and police departments, hospitals, parks, railroads, roads, trails, water features, and zip codes. Government produced data will provide the bulk of this data as it typically meets all eight data requirements set forth above. Potential data sources for this data are the NSDI Clearinghouse Network, state government geospatial data portals, local government geospatial data portals and commercial geospatial datasets.

Requirement 2 – Completed Metadata

The data mining process relies on metadata as the mechanism for decision-making and pattern analysis. All datasets to be analyzed must contain completed metadata, or at least the Dublin Core Metadata Element Set (Initiative 2008). Specific metadata elements of interest are: originator, title, abstract, keywords, description, geographic extent, geometry type, attributes, attribute domain values, attribute description, and ISO Topic Category. These selected metadata elements provide an excellent source of information required for theme classification.

Additionally, the metadata must be in a structured data format that is conducive to traversal by a computer program. Examples of such formats are extensible markup language (XML/.xml), hypertext markup language (HTML/.html), or text files converted from well-structured formats (.txt/.met).

Requirement 3 – Freely Available or Widely Used Datasets

To help foster the democratization of cartography, proprietary and/or expensive datasets were excluded from the project unless it was in widespread use. Freely available, widely used geospatial datasets had preference during the selection phase of this research. This preference reduced the cost of the research and still allowed for a very large population of datasets.

Requirement 4 – Data must be of Vector Data Type

Vector datasets are the only type of data used in this research to contain the scope.

Requirement 5 – Data must be in Digital Format

Only digital data is used in this research. Hardcopy data that has not been digitized is purposely excluded for two reasons: 1) no digital metadata is likely to exist for hardcopy data, and it would be too time consuming to research the information required to fill out the metadata properly; and 2) hardcopy datasets are not likely to be widely used as they are not in a computer-readable format yet, and, therefore, puts hardcopy data in contention with requirement 3 and would require too much time to convert to digital format.

Requirement 6 – Data must only Contain One Theme of Information

While it is possible for one dataset to contain multiple themes of information (for instance, an emergency services dataset containing police, fire, and hospitals), including

these datasets in the study might create a situation of inconsistent manual classification. Additionally, typically datasets contain one theme of information and these multi-theme datasets are considered rarities and, therefore, are excluded for this research but would should be included in future research.

Requirement 7 – Data May Only Cover the Geographic Extent of the United States of America

To contain the scope of this project, the geographic area was constrained to the United States of America.

Requirement 8 – Data and Metadata Must be in the English Language

To contain the scope of this project, the data and metadata was constrained to the English language.

Data Acquisition

Online and offline media were the target source media for extracting geospatial datasets and metadata. A multitude of sources at private companies and differing levels of government were searched to find geospatial datasets that met the eight requirements listed above. Tapping as many disparate sources of data was preferred to decrease the potential for bias towards one data producer or type of data producer. Additionally, gaining a sample of geospatial data and metadata from as many locations as possible allows the resulting product to be applicable to the widest possible audience.

Once a dataset was downloaded, it was placed into a directory coinciding with a theme. Four top-level themes were chosen that categorized the downloaded data. These were, “transportation”, “boundaries”, “physical features”, “points of interest”. Each top-level theme was further broken down into sub-themes. For example, the top-theme

“points of interest” was broken down into sub-themes such as “airports”, “hospitals”, “monuments”, “museums”, “parks”, “police” and “fire”. This categorizing of datasets was done so that programs that would be authored later could handle datasets by theme separately.

KDD Step 3: Preprocessing and Cleaning the Data

After acquisition, the geospatial datasets and metadata needed to be cleaned and preprocessed. The goal of this step was to remove noise and outliers, and remove datasets that did not fit the eight requirements. The resulting datasets would be considered to be clean and suitable for the next step.

Preprocessing

Once all datasets were decompressed, the datasets needed to be converted into a homogenous format to ease the process of automatically extracting useful information. The format chosen to have all datasets converted into was the shapefile format as is a very popular format, is an easy format to convert datasets into, and many programming functions exist that can access the format.

Cleaning

With the datasets decompressed and converted into the ESRI Shapefile format, datasets that did not meet the requirement of completed metadata must be removed from the population of acquired datasets. A script was written that visits each downloaded dataset and searches for a matching metadata file. If a dataset was found to not have a related metadata document, then the dataset was removed from the population. The pseudo-code for the script is displayed in Figure 4.1.

```

metadataExtensions = [".xml", ".met", ".txt", ".html", ".htm"]
For each directory in path:
    For each shapefile in directory:
        For each metadataExtension in metadataExtensions:
            If (shapefile[without .shp extension] +
                metadataExtension) does not exist:
                Remove shapefile from population

```

Figure 4.1: Pseudo-code of Script that Removes Shapefiles without Metadata Files

As shown in Figure 4.1, the script was designed to search for commonly used extensions for geospatial metadata documents that are in a text format. These three formats are: extensible markup language (XML) (W3C 2008) format which has the extension .xml, text format which has the extensions .txt or .met, and hypertext markup language (HTML) (W3C 1999) format which has the extensions .html or .htm. With the common extensions defined, the program looks for each shapefile inside of a provide directory path. Then, for each shapefile found inside a directory, the .shp extension is removed from the shapefile name and replaced with each one of the metadata extensions, one at a time. With the metadata extension replacing the .shp extension, the program checks for the existence of a file with that name (hence, a metadata document). If the file does not exist for all of the possible metadata extensions, it is assumed that no metadata exists for the shapefile, and the shapefile is removed from the population acquired geospatial datasets.

Once the population of datasets was reduced to only datasets with metadata documents, the next stage was to crawl through each metadata document and to 1) search for documents that had formatting problems and remove them from the population; 2) compile descriptive statistics about the metadata documents to uncover commonalities and anomalies within the data; 3) remove keywords that do not lend themselves to theme

identification such as “at”, “the”, “none”, or “and”; 4) combine plural and singular forms of keywords into one entry; and 5) extract keywords found in each metadata document and write the total of each keyword, compiled by theme, to an output file.

These five stages were written into a single computer program using the Python programming language and ESRI’s ArcPy application programming interface (ESRI 2011). The complete program listing can be found in Appendix B. Each one of these five stages will now be discussed and the relevant section of the computer program will be discussed using pseudo-code.

KDD Step 3 - Stage 1 – Find formatting problems

In order for the computer to crawl through the metadata documents, the documents must follow a common format. The most common metadata format of the metadata downloaded was the Content Standard for Digital Geospatial Metadata (CSDGM) version 2 (Federal Geographic Data Committee 1998). Metadata documents that follow the CSDGM must be well-formed and utilize the same structure and terms set forth by standard. Of all of the metadata stored in the CSDGM metadata documents, the theme keywords were the target data to be extracted. These keywords were stored in the ‘themekey’ portion of the document which is nested within the document as shown in Figure 4.2.

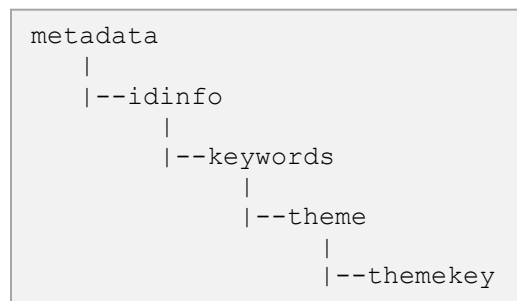


Figure 4.2: Hierarchy of Target Components of CSDGM Metadata Standard

The CSDGM does not specify which file format the metadata must reside in (for instance, XML or TXT), but metadata is typically found in one of the three formats identified by the program shown in Figure 4.1. Of these three formats, (XML, HTML, and text), XML and HTML are considered markup languages. In a markup language, information is surrounded by markup tags which are intended to be interpreted in a special way by another program that reads the file. Additionally, these tags build a hierarchical structure and can be checked for consistency against the CSDGM. The XML and HTML files can be loaded into a tree structure which allows for efficient traversal through the branches of the tree to find a specific set of data. If, with the XML or HTML file loaded into the tree structure, a particular branch does not exist that is required by the CSDGM, the metadata can be determined to be invalid, or, at least, incomplete, and, therefore contradictory to requirement 2 set forth above.

There is a caveat with HTML files with respect to checking for consistency against the CSDGM, however. XML files allow for the author to create custom markup tags; the terms/tags required by the CSDGM. The HTML standard does not allow for this flexibility as it has a set of predefined tags. Because of this, metadata stored in the HTML format does contain the terms set forth by the CSDGM, but not in the markup tags, therefore, it cannot be checked for consistency against the CSDGM standard, only the HTML standard. Because of this issue, all of the tags in the HTML file are stripped away from the HTML file, rendering it to be simply, the same as a text file. Because of this caveat, XML and HTML files must be processed in a different way even though they are both markup languages.

Text files do not contain any special markup tags and cannot be checked for consistency in the same way that XML and HTML files can. Therefore, text files cannot be checked for hierarchical structure consistency, but the terms set forth by the CSDGM can be found within the text document through pattern matching. If the term and associated data matches a defined pattern inside the text file, the data can be reliably extracted.

If a XML metadata file was found to be valid for the current shapefile, and HTML and/or text metadata files were also found for the same shapefile, then the HTML and/or text files would not be checked or used in the research. The reason for this is that the XML file can be checked for consistency of hierarchy whereas the HTML and text files cannot. Since the XML file can have one additional check, this puts the XML file at a higher priority for data extraction.

With the three formats defined in more detail, the method used to find formatting problems will now be discussed. For each format, pseudo-code will be shown in a figure and the code will be discussed in narrative.

XML

The pseudo-code for the traversal of the XML document is show in Figure 4.3. The first task in this part of the program was to load the XML file into memory. XML files can be checked for structural consistency by loading it into a tree data structure within computer memory. A Python software module named ElementTree (Lundh 2005) was leveraged to allow for the easy loading and traversal of the XML metadata document in computer memory. With the XML metadata file loaded into a tree structure, the tree was traversed to check for the existence of these required elements: idinfo, keywords,

theme, and themekey. If the themekey part of the metadata document is found properly nested within the XML document, then the document is checked for the existence of keywords stored within the themekey part of the document. If both the themekey part is found and contains at least one valid keyword, then the metadata file is considered valid, otherwise it is reported as invalid.

```
import ElementTree
For XML file in directory:
Parse XML file into Tree structure using ElementTree
Traverse the tree to find the following required nodes for this
research:
    • idinfo
        o keywords
            ▪ theme
                • themekey
If the themekey node is found and is not empty:
    Metadata file is valid
Else:
    Metadata file is invalid
```

Figure 4.3: Pseudo-code Checking XML Metadata file for Required Components

HTML

The pseudo-code for the traversal of the HTML document is show in Figure 4.4. This portion of code will only run if a XML metadata was not found, or was found and found to be invalid for the current shapefile. Next, all HTML markup tags are stripped (removed) from the document. The tags are removed to simplify the document for subsequent pattern matching. Next, the terms “Theme_Keyword” or “Theme Keyword” are searched for. In both the HTML and text documents, these are the terms used to indicate that keywords follow. If one of the two terms is found, and keywords follow on the same line, then the metadata document is deemed valid.

```

If XML file is not found or valid for current shapefile:
For HTML file in directory:
Strip all HTML markup tags from the file
Search for "Theme_Keyword" or "Theme Keyword" term
If term is found and keywords exist on the same line:
    Metadata file is valid
Else:
    Metadata file is invalid

```

Figure 4.4: Pseudo-code Checking HTML Metadata file for Required Components

Text File

The pseudo-code for the traversal of the text document is show in Figure 4.5.

This portion of code will only run if XML or HTML metadata documents were not found, or were found and found to be invalid for the current shapefile. The terms "Theme_Keyword" or "Theme Keyword" are searched for. If one of the two terms are found, and keywords follow on the same line, then the metadata document is deemed valid.

```

If XML and HTML files are not found or valid for current shapefile:
For text file in directory:
Search for "Theme_Keyword" or "Theme Keyword" term
If term is found and keywords exist on the same line:
    Metadata file is valid
Else:
    Metadata file is invalid

```

Figure 4.5: Pseudo-code Checking Text Metadata file for Required Components

KDD Step 3 - Stage 2 – Create Descriptive Statistics

The goal of stage two is to create descriptive statistics to uncover any commonalities that may be leveraged, or issues that need to be addressed before proceeding further. Descriptive statistics were collected for:

- Number of shapefiles...
 - with XML metadata document
 - with HTML metadata document
 - with text metadata document
- Average number of keywords per metadata file
- Total number of keywords
- Number of unique keywords
- Number of instances of each keyword

The pseudo-code show in Figures 4.3, 4.4, and 4.5 were extended slightly to include counters for each one of these statistics.

```
If metadata document valid, increase appropriate counter
For each keyword found in the themekey or "Theme_Keyword" section:
    Convert each keyword to lowercase
    Remove any leading or trailing spaces
    Increase counter for total number of keywords
    If keyword already exists in unique keyword list:
        Do not add to unique keyword list, but increment
        individual keyword counter
    Else:
        Add keyword to unique keyword list and increment
        individualkeyword counter
Divide total number of keywords counter by number of metadata
documents to derive average number of keywords statistic
```

Figure 4.6: Pseudo-code of Descriptive Statistics Creation

Figure 4.6 displays pseudo-code that compiles these statics. This pseudo-code is placed at the bottom of the previous three snippets of code. The first line of pseudo-code shown in Figure 4.6 increases a counter that holds the total number of valid metadata documents found. If two metadata documents are found for one shapefile, the counter is only incremented once since only one of the metadata documents are used. Next, each keyword is converted to lowercase to ease checking for string equivalencies later. Leading and trailing spaces are also removed as they are not considered meaningful characters. Next, a counter that holds the total number of found keywords is incremented and a check is made to see if the keyword has been found before in the current or any other metadata document that has been checked. A unique keywords list has been created to keep only one entry for each word found. If a keyword is found multiple times, it will only show up one time in the unique list, but it will also record how many times the unique keyword has been found. Lastly, the program divides the total number of keywords found by the total number of metadata documents that keywords have been extracted from to derive the average number of keywords per metadata document.

Normally, the results of these statistics would be discussed in the results section of the chapter, however, the results of the descriptive statistics did uncover issues that needed to be addressed through additional cleaning before moving forward in the KDD process. Stages 3 and 4, discussed next, were inserted because of the issues uncovered in the descriptive statistics. These sections will discuss the relevant descriptive statistics that uncovered the issues. The remaining descriptive statistics will be discussed in the results section of this chapter.

KDD Step 3 - Stage 3 – Remove Non-Meaningful Keywords

In Stage 2, one of the descriptive statistics created was the number of instances of each keyword across all metadata documents in a theme. After reviewing this statistic, it was discovered that a larger number of non-meaningful words were included in the keywords collection. For example, the word “none” was found 455 times across all metadata documents in the collection. It was discovered that, in a particular software package commonly used to create metadata, if no keywords were entered, then the software would record “none” (to designate no keywords entered by the user) in the keyword part of the metadata document. The term “none” is not a meaningful word for describing the theme of a dataset and, therefore, should be removed for the purposes of this research.

In addition to the term “none”, 23 other terms were deemed as non-meaningful terms and needed to be removed from the metadata documents. These terms are listed in Table 4.1 below. In Table 4.1, the term *space* represents one or more spaces, and the terms *email address* represents a unique email address found in the keywords section of metadata documents.

Table 4.1: Non-Meaningful Terms to be Removed from Metadata Documents

none	required	and	or	not
the	a	this	that	in
of	is	has	on	at
by	for	please	was	sake
also	*space*	*email address*		

The scripts in Part 2 were modified to ignore the non-meaningful terms listed in Table 4.1. After unique keywords were compiled into a data structure, the script would

remove any entries deemed non-meaningful. The pseudo-code listed in Figure 4.7 displays the method used to remove the non-meaningful keywords.

```
meaninglessKeywords = ["and", "or", "not", "the", ..., "also"]
for each uniqueKeyword in uniqueKeywords list:
    if uniqueKeyword is found in list of meaninglessKeywords:
        remove meaninglessKeyword from uniqueKeywords list
```

Figure 4.7: Removal of Non-Meaningful Keywords

KDD Step 3 - Stage 4 – Combine Plural and Singular Forms of Keywords

The descriptive statistics compiled in Stage 2 also uncovered a large amount of keywords that existed in both the plural and singular forms. For example, “park” and “parks” both had a large number of entries in metadata documents. It is clear that they both are identifying the theme of the dataset as a park, however, having two separate entries for the same theme weaken the computer’s ability to correlate these keywords to a single theme. As the purpose of the keywords in a geospatial metadata document is to identify the theme of a dataset, it is assumed that the singular and plural form of a keyword can be considered to identify the same theme. With this assumption, singular and plural forms of keywords can be combined to represent one entry in the unique keywords list thereby reducing the number of duplicate entries, and, therefore, combine geospatial datasets that only have “park” as a keyword with other datasets with only “parks” as a keyword.

Three common forms of plurals were targeted for removal in this research: addition of letter “s”; addition of letters “es”; and replacement of letter “y” with “ies”. Special cases of plurals, such as plurals that maintain their Latin form in plural form, or

mutated plurals (*e.x.* “person” into “people”) were ignored as the vast majority of plurals fell into the three common forms, and for scripting purposes, the small number of special case plurals were considered acceptable error.

In this research, only if both the singular and plural form of a keyword was found in the list of unique keywords, would the plural keyword be removed in favor of the singular keyword. If a plural was removed, the number of plural keywords would be added to the total of the singular form. The code used to accomplish this task is listed in Figure 4.8.

```
#Replace "ies" from end of word with "y" to see if base word found
for keyword in keywordsHash:
    if last three letter in keyword are "ies" and ...
    if keyword-"ies"+"y" exists in keywordsHash:
        keywordsList[keyword-"ies"+"y"] +=
            keywordsHash.pop(keyword)

#Remove "es" from end of word to see if base word found
for keyword in keywordsHash:
    if last two letters in keyword are "es" and ...
    if keyword-"es" exists in keywordsHash:
        keywordsList[keyword-"es"] += keywordsHash.pop(keyword)

#Remove "s" from end of word to see if base word found
for keyword in keywordsHash:
    if last letter in keyword is "s" and ...
    if keyword-"s" exists in keywordsHash:
        keywordsList[keyword-"s"] += keywordsHash.pop(keyword)
```

Figure 4.8: Removal of Plural Keywords if both Singular and Plural Found

Figure 4.8 displays the three algorithms employed to combine plurals to found singulars. The first of the three algorithms will now be explained to clarify the pseudocode. The first block of code is the algorithm that replaces the last three letters “ies” from the end of a keyword with the letter “y” to see if the modified keyword has

also been found in the explored metadata documents. The first line of the algorithm selects a keyword from all keywords found in the geospatial metadata documents. The second line checks to see if the last three letters of the keyword are “ies”. If the last three letters of the keyword are “ies”, then the algorithm replaces “ies” with “y” and searches for the modified keyword in the list of all keywords. If there is a match, the plural form of the keyword is simultaneously removed from the list of keywords found, and the total instances of the plural form found are added to the total instances of the singular forms of the keyword.

KDD Step 3 - Stage 5 – Compile Keywords and Keyword Counts into Output Files by Theme

With Stages 1 through 4 completed, the geospatial metadata is considered clean enough to begin compilation into final output files. One final output file was created for each theme. Separating the outputs by theme allow for easier creation of training files for the data mining algorithm. For each theme, three output files were created: extractLog.txt, fieldNames.txt, and keywords.txt.

extractLog.txt

The output file extractLog.txt contained the descriptive statistics created in Stage 2. This file displays statistics of interest to the user and provides an initial peek at the state of the geospatial metadata included in the theme. Figure 4.9 displays the extractLog.txt for the Roads theme.


```
Number of SHP files parsed: 275
Number of XML files parsed: 214
Number of XML files with keywords matched: 153
Number of HTML files parsed: 20
Number of HTML files with keywords matched: 3
Number of TXT/MET files parsed: 17
Number of TXT/MET files with keywords matched: 8
Plurals removed: 16
Average number of keywords per metadata file: 6
Total number of keywords found: 1119
Number of unique keywords:169
Total number of field names found: 7350
Number of unique field names:2119
Average number of field names per shapefile: 26
```

Figure 4.9: extractLog.txt for Roads Theme

fieldNames.txt

The second output file, fieldNames.txt, contains a delimited list of unique field names extracted from the geospatial datasets, and how many instances of the unique field names found expressed as a percentage of all unique field names found in the geospatial datasets in the same theme. The number of instances was expressed as a percentage so that inter-theme comparisons of the most popular field names could be fairly compared. Initially, total counts were utilized, however, if there was a large disparity in the number of geospatial datasets collected between two themes being compared, the total counts did not provide for a fair comparison. Figure 4.10 displays a portion of fieldNames.txt for the Trails theme. There are two columns delimited by two semi-colons. The column on the left is a unique field name. The column on the right is the number of times that unique field was found expressed as a percentage of all found field names.

```
surface_ty;;0.520
trailid;;0.260
nht_office;;0.260
road_name;;0.260
railway_t;;0.260
objectid_1;;0.520
opened;;0.260
riverwalk_;;0.260
co_name;;0.260
comments;;0.260
fac_10;;0.260
access;;0.260
trailprior;;0.260
miles;;1.041
```

Figure 4.10: fieldNames.txt for Trails Theme

keywords.txt

The third output file, keywords.txt, has the same format as fieldNames.txt. It contains a delimited list of unique keywords extracted from the geospatial datasets, and how many instances of the unique keywords found expressed as a percentage of all unique keywords found in the geospatial datasets in the same theme. The number of instances was expressed as a percentage so that inter-theme comparisons of the most popular keywords could be fairly compared. Figure 4.11 displays a portion of keywords.txt for the Airports theme. There are two columns delimited by two semi-colons. The column on the left is a unique keyword. The column on the right is the number of times that unique keyword was found expressed as a percentage of all found keywords.

```
airport;;27.4509803922
towers;;0.980392156863
landing;;2.94117647059
environmental;;0.980392156863
connecticut;;0.980392156863
protection;;0.980392156863
aviation;;0.980392156863
faa;;0.980392156863
data;;0.980392156863
codes);;0.980392156863
structure;;2.94117647059
database;;0.980392156863
acais;;0.980392156863
runway;;14.7058823529
```

Figure 4.11: keywords.txt for Airports Theme

KDD Step 4 – Transformation of Data into Data Mining Algorithm-Friendly Format

The fourth step of the KDD process is the transformation of the preprocessed data into a data mining algorithm-friendly format. This will be accomplished by creating input files for the data mining algorithm that only contains potentially useful variables and training information in a structured format.

The input files will be created in three stages: calculating goodness of fit, addition of geometry variable, and addition of a training variable. These three stages were written into a single computer program using the Python programming language and ESRI's ArcPy application programming interface (ESRI 2011). This input to this program is the output files created in Step 3 of the KDD process. The complete program listing can be found in Appendix C. Each one of these three stages will now be discussed and the relevant section of the computer program will be discussed using pseudo-code.

KDD Step 4 - Stage 1 – Calculating Goodness of Fit

In KDD Step 3, two variables were extracted from the geospatial metadata and preprocessed: theme keywords and field names. These two variables were stored in final output files expressed as how many instances were found as a percentage of all instances found for a theme. These two variables will be utilized as metrics to determine how good a dataset of a particular theme matches with the keywords and field names. For instance, if my input is a geospatial dataset that represents a city park, I will compare all of the keywords found in the dataset against the keywords extracted in KDD Step 3. For each keyword that is found in both the input dataset and the extracted keywords, the percentage associated with the keyword will be summed for a single total. The higher the total percentage match, the more closely the input parks dataset matches with existing datasets' keywords. This same process is repeated for field names. The results of this matching operation are written out to an output file along with the shapefile name so that the totals can be linked back to a unique shapefile. The pseudo-code for this operation is shown in Figure 4.12.

```

For each theme:
    For each fieldName in fieldNames.txt:
        extractedFieldNames[fieldName] = percentage
    For each keyword in keywords.txt:
        extractedKeywords[keyword] = percentage

    For each shapefile:
        For each fieldName in shapefile:
            fieldNameScore += extractedFieldNames[fieldName]
        For each keyword in shapefile:
            keywordScore += extractedKeywords[keyword]

        Write shapefile name, keywordScore,
            fieldNameScore to output file

```

Figure 4.12: Match Dataset Keywords and Field Names to Extracted Keywords and Field Names

In Figure 4.12, the first block of code chooses a theme and creates a hash data structures keyed by field names and keywords found in the output files created by Step 3 of the KDD process. The second block of code then compares the keywords and field names of all shapefiles from all themes against the extracted keywords and themes from the current theme. If a keyword or field name from the current shapefile is found in the hash, then the percentage is added to a running total to be written to the output file. Once all shapefiles have been compared against a single theme, the third block of code writes the results of those comparisons to an output file for the current theme. The next theme is then selected and the code runs again until all keywords and field names from each theme have been compared against all shapefiles.

A sample of the output from this code is shown in Figure 4.13. This sample displays a sample of the results of matching all shapefiles' keywords and field names to the keywords and field names extracted for the Airport theme. The first line in the output file is the headers for the three columns in the output: FileName for the shapefile file name, MatchField for the percentage of field names matched for the theme, and

MatchKey for the percentage of keywords matched for the theme. Subsequent lines show the results. As this is the results of the matching against the Airports theme, as expected, the two shapefiles of airports (ok_airports.shp and hgac_airports.shp) have significantly higher matches against the field names, keywords, or both when compared to the shapefiles of county boundaries (JimHoggCounties.shp, LimeStone County.shp), water features Albany Lakes.shp), and railroads (railroads.shp) shapefiles.

FileName	MatchField	MatchKey
JimHoggCounties.shp	4.100	0.980
LimeStone County.shp	4.100	0.980
Albany Lakes.shp	4.497	0
ok_airports.shp	27.513	0
hgac_airports.shp	6.216	27.450
railroads.shp	0.925	0

Figure 4.13: Sample Output from Keyword and Field Name Comparisons for Airport Theme

KDD Step 4 - Stage 2 – Addition of Geometry Variable

In addition to keywords and field names, a third variable will be included as a potentially useful variable for the data mining algorithm: geometry (point/line/polygon). Geometry is being included as a potentially useful variable because it is believed that certain themes will be represented by a single geometry in most cases. For instance, county boundaries are most likely to be represented as a polygon rather than a point or line, even though both of those cases are possible. By including geometry in the list of input variables, this provides the data mining algorithm with a third potentially useful variable that makes logical sense. Figure 4.14 displays the pseudo code for recording the geometry type to the output file created in Stage 1 above.

```
For each shapefile:
    geometryType = arcpy.describe(shapefile).shapeType

Write shapefile name, keywordScore,
    fieldNameScore, geometryType to output file
```

Figure 4.14: Extracting Geometry Type from Shapefiles

The code shown in Figure 4.14 is replaces the last code block in Figure 4.12. The geometry type for the current shapefile is reported by the Describe function available in Esri's ArcPy API and stored in the geometryType variable. The geometryType is then added to the output file. The resulting output file is displayed in Figure 4.15 and is an updated version of the Airport theme output shown in Figure 4.13.

```
FileName, MatchField, MatchKey, ShapeType
JimHoggCounties.shp, 4.100, 0.980, Polygon
LimeStone County.shp, 4.100, 0.980, Polygon
Albany Lakes.shp, 4.497, 0, Polygon
ok_airports.shp, 27.513, 0, Point
hgac_airports.shp, 6.216, 27.450, Point
railroads.shp, 0.925, 0, Polyline
```

Figure 4.15: Addition of Geometry to Output File

KDD Step 4 - Stage 3 - Addition of a Training Variable

The third and final stage was to include a training variable to the output file. As the goal of the data mining operation is to determine the theme of an input dataset, the data mining algorithm must be trained to know which shapefiles are of which theme and the results of matching keywords and field names to the extracted keywords and field names for that theme. The idea behind this is that if the data mining algorithm knows how shapefiles of a theme match keywords and field names to all extracted keywords and

field names of that theme, and it knows how shapefiles not of the current theme perform, then it will be able to determine what facets of the matching can be used to determine the most probably theme for an input shapefile.

The training variable is the Boolean value of “Yes” or “No”. “Yes” designates that the current shapefile is a member of the theme it is currently being evaluated against. “No” designates the opposite case. The data mining algorithm can use the Boolean value to train itself on how an input shapefile matches against its own theme, and other themes.

The pseudo code of this operation is shown in Figure 4.16. For each theme iterated through for matching keywords and field names, the current shapefile’s theme is matched against the current theme to see if they match. If they match, the trainingValue is set to “Yes”, otherwise it is set to “No”. The training value is written to the output file along with the other variables calculated in Stages 1 and 2.

```
For each theme:
    For each shapefile:
        if the shapefile's theme equals theme:
            trainingValue = "Yes"
        else:
            trainingValue = "No"

    Write shapefile name, keywordScore, fieldNameScore,
        geometryType, trainingValue to output file
```

Figure 4.16: Determining Value of Training Variable

The resulting output file from the code shown in Figure 4.16 is displayed in Figure 4.17 and is an updated version of the Airport theme output shown in Figure 4.15. A new column named “Correct” is added to the output file. Training Variable values are entered on each line in the “Correct” column.

FileName, MatchField, MatchKey, ShapeType, Correct
JimHoggCounties.shp, 4.100, 0.980, Polygon, No
LimeStone County.shp, 4.100, 0.980, Polygon, No
Albany Lakes.shp, 4.497, 0, Polygon, No
ok_airports.shp, 27.513, 0, Point, Yes
hgac_airports.shp, 6.216, 27.450, Point, Yes
railroads.shp, 0.925, 0, Polyline, No

Figure 4.17: Addition of Training Variable to Output File

Figure 4.17 is the final data mining algorithm-friendly format that will be used as input to the data mining algorithm. Each theme will have one output file in this format so that the data mining algorithm can be trained for every theme individually.

KDD Step 5 – Choosing the Data Mining Task

The data mining task in this research is a prediction task. A prediction task is one that moves from some observations from specific data to a more general rule or description. The goal of a predictive task is to make predictions about unknown cases based on the learned patterns from the training datasets. As the ultimate goal of the data mining process of this research is to have the computer autonomously identify (predict) the theme of an input dataset, prediction is the appropriate task.

KDD Step 6 – Choose Data Mining Algorithm and Parameters

The predictive algorithm chosen for this research was the random forest of decision trees. A decision tree is a predictive algorithm that utilizes observations to make a linked, hierarchical set of decision points which are used to derive a conclusion about input data. Decision trees offer many advantages including insight into data structure, hierarchical data representation, and ease of interpretation. Decision trees are constructed from observed attributes in training data. Ultimately, the resulting decision tree will contain nodes and branches which classify the training dataset into objects that share

similar attributes. The nodes serve as providing the predicted value of an inputted dataset that traverses through the tree branches to terminate at the node.

The random forest algorithm creates many classification trees. For each split on each tree, only a random subset of input variables are available for subsequent splits. The output tree contains the mode of the splits from all of the trees in the forest. The main purpose for choosing the random forest algorithm is that it allows for quick experimentation for detecting variable interactions without having the user manually change the parameters. Through multiple runs of the random forest algorithm, the resulting trees show a convergent behavior of optimum decisions for branching in the trees.

For this research, the following parameters were chosen for the random forest algorithm and will be discussed below:

- Number of trees: 10
- Criterion: gain ratio
- Minimal gain: .1
- Minimal size for split: 4
- Minimal leaf size: 2
- Maximal depth: 20
- Pre-pruning: True
- Pruning: True

Number of trees

This parameter determines how many learned random trees will be created by the algorithm. Ten trees were chosen as to not overwhelm the user by the number of trees

generated. The user will compare the generated trees for commonalities to determine the values of important variables and where branches should occur in the tree.

Criterion

The criterion parameter specifies how the attributes and splits will be selected. The gain ratio calculation ranks a branch in a tree based on how cleanly the branch separates the data (Myatt 2007). This calculation is based on the impurity of a node before and after a split.

Minimal Gain

The minimal gain parameter sets the minimum amount of progress a split makes towards having a pure/perfect split in order for it to be considered an acceptable split. The gain parameter may have a value between 0 and 1. A gain of 0 represents a split with an equal number of contradictory observations in each resulting node. This is considered the most impure of splits as no imbalance exists. A gain of 1 represents a split with resulting nodes with no contradictory observations in each node; a pure split.

For this research, a minimal gain of .1 was chosen after a number of runs of the data mining algorithm. A minimal gain of .1 requires a reasonable amount of progress for each split, but not too much progress that no splits can meet the requirement.

Minimal Size for Split

Minimal size for split sets the required number of observations to be in a node for it to be considered split-able. For this research, the minimal size for split was set at ten. Through iterative runs of the data mining algorithm, choose ten for this criterion produced a tree that did not seem to over fit the input by creating small nodes very specific to individual input datasets.

Minimal Leaf Size

The minimal leaf size parameter sets the minimum size of all leaf nodes in the tree. Splits that would create a leaf below the threshold set by this parameter would not be allowed. A minimum leaf size of two was chosen for this research. Like the minimal size for split parameter, a leaf size of two was chosen so that the produced tree would not over fit the input data.

Maximal Depth

The maximal depth parameter sets the maximum number child splits from the root node. For this research, a maximal depth of 20 was chosen as the maximal depth to keep the decision trees manageable in size but not too restrictive in the case that a deep node produces a significantly pure split.

Pre-pruning

Pre-pruning stops splitting the decision tree when additional splits cannot improve the predictive performance. This is done through determining which input attributes are significantly correlated to an output class, and only using those attributes. For this research, pre-pruning was enabled to stop splitting when splits become unreliable.

Pruning

Pruning is completed after the decision tree has been generated. This pruning process removes splits that do not add to the predictive ability of the tree. Pruning simplifies the tree without sacrificing the predictive abilities of the decision tree. For this research, pruning was enabled.

KDD Step 7 – Execute Data Mining Algorithm

To classify the geospatial datasets, the software package Rapid Miner (Mierswa *et al.* 2006) was utilized. Rapid Miner is an open-source software package that provides analysis and data mining algorithms and visualization. Rapid Miner has the capability to perform data modeling and develop Decision Tree and Random Forest data mining algorithms.

To develop the decision trees, the training datasets built in Step 4 of the KDD process were used as inputs into Rapid Miner. Rapid Miner includes a random forest data mining algorithm that was used in this research. The parameters set in KDD Step 6 were inputted and the training datasets were set as the input data. The data mining algorithm would then be executed and interpreted in KDD Step 8.

KDD Step 8 – Interpretation of Data Mining Results

This last step of the KDD process is to interpret the results of the data mining operation. Previously, in Step 7 of the KDD process, ten decision trees were yielded per theme. Each decision tree in a forest shows one way in which the input dataset can be classified based on the input variables. Each of the decision trees must be manually evaluated by the data mining operator to determine which decision tree(s) yield the most useful classifications. In this research, the decision trees were evaluated based on the following two criteria: minimize false-positives, and resist over-fitting of decision tree to training data.

Minimizing false-positives was chosen as a criterion to provide a pathway for a prediction with the highest confidence. As the goal of the data mining process is to have a computer autonomously identify the theme of a dataset, classes in a decision tree that

are not able to create clean splits of a theme were not chosen. If a node of a tree only contains a small number of false positives and a large number of positives, then the node would be chosen because of the productiveness of the classification.

The second criterion used for choosing the best decision tree(s) is to resist over-fitting decision trees to the training dataset. Decision trees that have branches yielding nodes that only have a very specific dataset in the class were pruned from the decision tree. The branches and nodes of a good decision tree should represent the significant variable values that best represent the makeup of the majority of the input datasets.

With the two criteria set, the data mining operator would consider the output of the data mining algorithm to choose the most effective trees to be used for automating the identification of geospatial dataset themes. To illustrate how the output from the data mining algorithm is used for interpretation, an example case will now be presented.

Understanding the Data Mining Algorithm Output

The output of the algorithm is ten trees in the random forest in a graph and text view. Figure 4.18 displays an example output from the data mining algorithm in two views. The left half of the figure displays a graphic view of one of the ten resulting decision trees. The right half of the figure displays the related text view.

In the graphic view of the decision tree, the rounded rectangles represent a split, or branch, in the tree based on the attribute inside the rounded rectangle. Directly below the rounded rectangles are the values that are used to split the observations into two resulting nodes. Lastly, the rectangles (with square edges) are leaf nodes that contain the split observations. Inside the leaf node rectangles are the words “Yes” or “No” and a bar. The word “Yes” represents a leaf node that contains all observations that have been

identified as a member of the current theme. The word “No” represents a leaf node that contains observations that have been identified as not a member of the current class. The bar is composed of red, blue, or a combination of both colors. The color red represents observations that are a member of the current theme, and blue represents observations that are not members of the current theme.

In the text view of the decision tree, each line represents a split in the decision tree and should be traversed from the top to bottom. On each line, the first word is the variable that the split is being based on. Next follows an equality operator ($>$, $<$, $>=$, $<=$) followed by the value the variable is being compared against and a colon. After the colon, the word “Yes” or “No” will represent whether the resulting node will contain all observations that were classified in the current class or not. Lastly, a set of curly braces will contain the number of observations that are in the class that are not part of the current theme (designated by the word “No”) and the number of observations that are in the class that are part of the current theme (designated by the word “Yes”). Lines that do not have the colon followed by curly braces means that no leaf nodes were created from the split and the subsequent two lines will be the splits that take place as child nodes of the current node. Children nodes are denoted by a pipe character (“|”) and an indentation at the front of the line and are children of the line immediately above that is indented one level to the left.

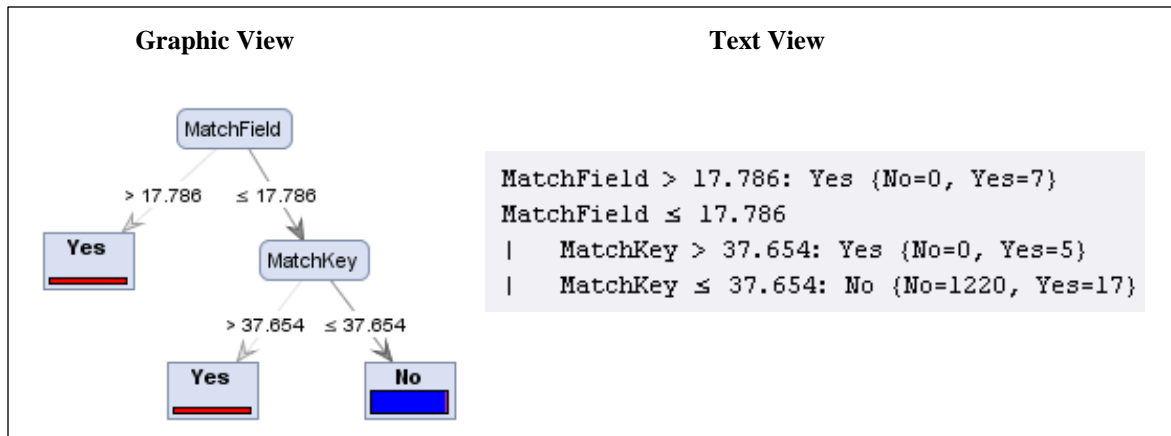


Figure 4.18: Example Output from Data Mining Algorithm

Figure 4.18 represents a single decision tree in the forest. In addition to this decision tree, the data mining operator would also consider the output of the other nine trees in the forest looking for commonalities among all ten trees to come to a final decision. However, for brevity, only this single tree will be used to explain the evaluation process.

Example Interpretation of Data Mining Algorithm Output

Using the example output shown in Figure 4.18, and the explanation on how to interpret the views from above, the data mining operator would interpret the decision tree in this way: First, leaf nodes designated as “Yes” that are the purest will be identified (remember that a pure node has no false-positives). Next, the location of the pure leaf node will be considered. The closer to the top of the tree a leaf node, the split that created the leaf node is typically more significant predictor. If a pure node is found much further down the tree, then it is most likely an over-fitted node as it will often only represent one or two datasets.

In Figure 4.18, the leaf node that is created from choosing all geospatial datasets where field names match with at least a score of 17.786 seems to be a significant node as it correctly identified 7 datasets as belonging to the current theme and had no false-positives. Additionally, the leaf node that is created from choosing all geospatial datasets where field names scored less or equal to 17.786 and keywords matched with a score greater than 37.654 successfully identified five datasets correctly and had no false-positives. Both pure leaf nodes are near the top of the tree and do not seem to be over-fitting the training dataset.

The remaining node created by choosing all geospatial datasets where the field names scored less or equal to 17.786 and keywords matched with a score less than or equal to 37.654 correctly identified 1,220 datasets as not part of the current theme, however, it also incorrectly identified 17 datasets as not part of the current theme when, in fact, they were part of the current theme. In the graphic view, the related node has a mostly blue bar, but with a small amount of red, which designates an impurity in the node. While the node correctly excludes a large number of datasets from the current theme correctly, it also excludes some datasets that should not be excluded.

The result of this interpretation is using the two splits that created the two pure nodes should perform well in identifying datasets that are members of the theme. As the two criteria are to minimize false-positives and resist over-fitting, these two splits are ideal for the purposes of this research and would be used to predict the theme of an input dataset.

However, 17 datasets were not able to be correctly identified based on the splits presented by the decision tree. This means that out of 29 datasets that were part of the

theme, 12 were correctly classified and 17 were not, which means that 59% of the datasets in the current theme were not correctly identified. The possible reasons for this lower performance will be discussed later in the chapter. However, even with a 59% failure rate, that does not mean that one of the 17 incorrectly classified themes cannot be correctly classified eventually. As will be discussed next, each input dataset will be scored against all possible themes, and then ranked based on score. Even if the dataset was not correctly classified into its theme, it may have scored the highest for that theme compared to all other themes, thereby causing its theme to be the most likely.

With all ten trees considered and significant splits recorded, the data mining algorithm would be run again and a new forest of ten trees would be considered. The data mining algorithm would be run multiple more times until the data mining operator could identify splits that were pure and representative of the emergent behavior of the forest of decision trees.

Prediction Program

The final splits that produced the purest nodes were synthesized into a set of rules and incorporated into a program that predicts the theme of an input dataset. The primary function of this program is to assess an input dataset provided by a user and to compare it to each theme's rules to determine which theme the dataset most likely belongs based on the theme it scores the highest. The program then prompts the user to confirm or reject the theme that the program predicted. If the user rejects the predicted theme, the next most likely theme will be proffered, until either a) the user accepts the proffered theme, or b) no more theme predictions meet a minimum score to be considered a candidate theme.

The program is designed around one algorithm that compares the input dataset against all rules from all themes. The pseudo-code for the algorithm is listed in Figure 4.19 and the complete code listing for this program is in Appendix D. In this program, the first block of code has the program load the list of keywords and field names and their associated percentage value from lists stored on the computer. The percentage value represents what percentage of the total population of the training dataset contained a particular keyword or field name. For each match of a field name or keyword, the percentage score is summed. The summed value is then divided by the minimum score required to consider the input a member of the theme. If a theme has multiple rules, all rules are evaluated and the highest score is the only score stored for future reference. In the second code block, after all themes have had their rules evaluated against the input, the scores are ranked highest to lowest in a list. The list is then iterated through where it prompts the user until the user agrees to the theme suggestion, remaining scores are less than .1 (10% confidence), or all scores have been considered. If the user agrees to the theme suggestion for the input dataset, then the input dataset theme is set to the current theme and the program ends.

```

For each theme:
    Load keywords and associated percentage score
    Load field names and associated percentage score
    Compare keywords, field names, and shape (as appropriate)
        with rules for theme.
    Sum and save scores as percentage match with theme.

Order theme match scores from highest to lowest
For each theme score:
    Loop until scores lower than .1, or user agrees
        Ask user if input is of theme of current score
        If "Yes":
            Set input theme equal to current theme

```

Figure 4.19: Prediction Program Main Algorithm

Evaluating the Method with Experiments

This section evaluates the developed KDD process of real world geospatial metadata. First, the downloaded datasets will be described and discussed. Next, the results of the data mining algorithm and resulting rules are presented. Finally, the accuracy of the dataset theme identification will be assessed.

Downloaded Datasets

In total, 2,099 vector datasets were downloaded from over 85 local, state, federal, and private agencies totaling about 60 gigabytes of memory. However, 805 of the downloaded datasets were discarded because they had either invalid metadata, or metadata that was not representative of the content stored in the geospatial dataset. For example, in the latter case, one data producer re-published portions of the U.S. Census TIGER files but all with identical metadata documents that did not accurately represent the content. This homogeneity in metadata documents across multiple themes of data was considered to be invalid metadata, and, thus, rejected.

In total, 1,294 vector datasets were downloaded from over 85 local, state, federal and private agencies totaling about 35 gigabytes of memory. With the datasets, 1,272 metadata documents were included. Of the 1,272 metadata documents, 1011 were in XML format, 150 were in HTML format, 91 were in text format, and 9 were in .met format. Additionally, there were 11 .PDF files, however the PDF files were excluded because there were so few and they could not be reliably traversed programmatically.

Of the 1,272 metadata documents, there were 166 instances of multiple metadata documents being provided for the same dataset, but in different file formats. The duplications means that 192 metadata documents are redundant thereby reducing the number of datasets with metadata documents down to 1,080, or 83%. Of the remaining 1,080 geospatial datasets, 7,484 keywords were extracted and 81 keywords were determined to be plurals of other keywords and were added to the singular form's total. This yielded 7,403 keywords which reduced further to 695 unique keywords at an average of 6 keywords per metadata document. Of the 1,294 datasets, 28,019 field names were extracted making 5,183 unique field names at an average of 21 field names per dataset. Table 4.2 further breaks down the descriptive statistics of the extracted keywords and field names by theme.

Table 4.2: Descriptive Statistics of Extracted Keywords and Field Names by Theme

Theme (Count)	<u>Keywords</u>			<u>Field Names</u>		
	Total	Unique	Average	Total	Unique	Average
State/County (366)	2,843	124	8	13,374	634	36
Zip Code (39)	190	71	6	200	82	5
Contours (98)	155	36	3	531	163	5
Water (211)	1,198	233	7	1,755	670	8
Airports (38)	130	51	3	990	486	26
Hospitals (25)	200	65	7	697	408	27
Parks (63)	247	104	5	935	595	14
Police/Fire (50)	690	96	15	977	272	19
Railroads (86)	265	96	4	775	430	9
Roads (286)	1,431	203	5	7,401	2,128	25
Trails (32)	135	66	5	384	269	12

Interpretation of Data Mining Algorithm Results

The training sets prepared following the procedures described earlier were used for the random forest classification algorithm. The result of each run of the algorithm yielded ten decision trees that were then interpreted by the data mining operator. From the interpretation of the decision trees, classification rules were created. The classification rules, graphic representation, and text representation of the decision trees deemed representative of each forest will now be presented and discussed for each of the eleven themes.

State and County Boundary Theme

Figure 4.20 displays the graphic and text view, and classification rule derived from the data mining algorithm output for the state and county boundary theme. The decision trees reported that classifying state and county boundaries by keywords found in metadata documents yielded the best results. As seen in the text view, if a geospatial dataset scores greater than a 50.703 in matching keywords with the state and county boundary training dataset, a pure classification occurs with 280 datasets being correctly identified and 0 false-positives. The split where keyword score ≤ 19.574 and Shape Type = Polygon created a fairly clean class (23 correct, 3 false-positives). However, this split was not considered strong enough because another split (keyword score < 13.349) classified 895 datasets as not members of the state and county boundary theme, and with both keyword scores being significantly less than the first split, being close in scores, and the seeming over-reliance on shape type being polygon to determine theme this split was not used for determining the classification rules.

The resulting classification rule from interpreting the random forest is: If Keywords Match Score > 50.703 , then theme is State/County Boundary. This rule meets the criteria discussed earlier and performs well against a sample.

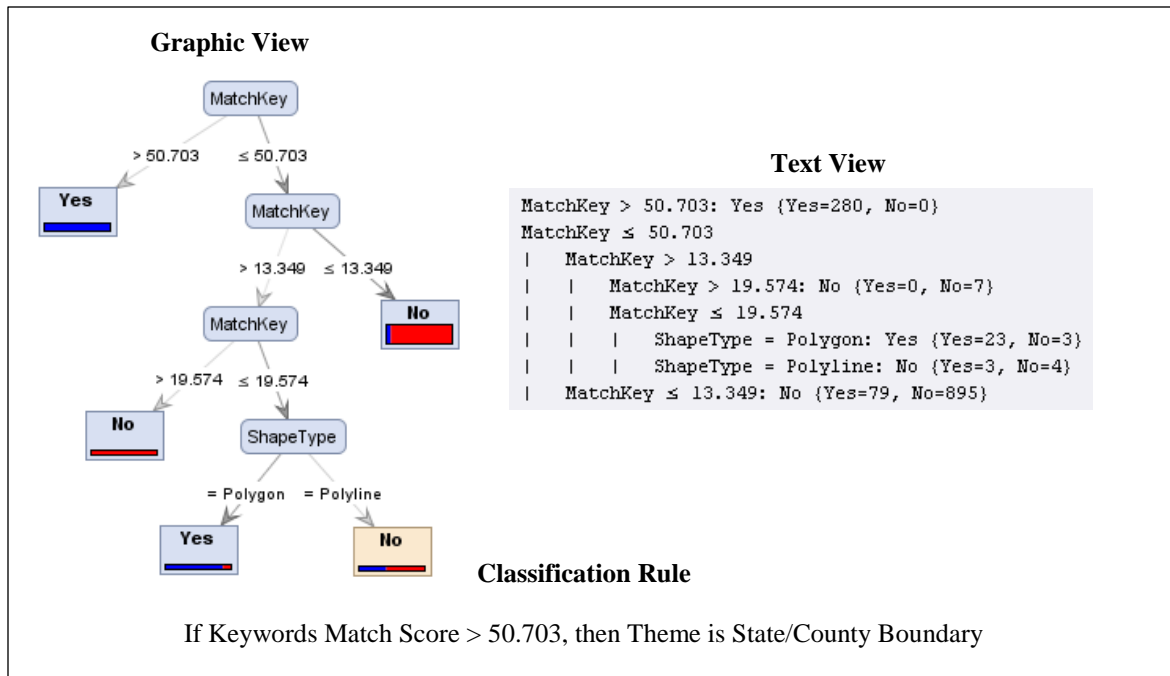


Figure 4.20: Decision Tree and Classification Rule for State and County Boundary Theme

Zip Code Theme

Figure 4.21 displays the graphic and text view, and classification rule derived from the data mining algorithm output for the zip codes theme. For this theme, the forest yielded three different pathways for classification. Because of these three pathways, Figure 4.21 displays two decision trees that represent the three pathways derived by the random forest.

The first pathway for classification uses keywords found in metadata documents that match with a score greater than 14.583 and shape type of “Polygon” produces a reasonably clean classification (15 correct, 2 false-positives). This pathway can be seen in the decision tree designated as “Tree 1” in Figure 4.21.

The second pathway, found in the decision tree designated as “Tree 2” in Figure 4.21, relies on keywords found in metadata documents matching with a score greater than

25.893 to consider a dataset a member of the zip code theme. This split produced a pure class (13 correct, 0 false-positives) in the sample.

The third pathway is also found in the decision tree designated as “Tree 2” in Figure 4.21. This pathway deems a dataset a member of the zip code theme if fields found in the geospatial dataset match with a score greater than 32.031. This split creates a pure class (3 correct, 0 false-positives).

The resulting classification rule from interpreting the random forest is: If Keywords Match Score > 25.893 or (Keywords Match Score > 14.583 and Shape Type = “Polygon”) or Fields Match Score > 32.031, then theme is Zip Code. This rule meets the criteria discussed earlier and performs well against a sample.

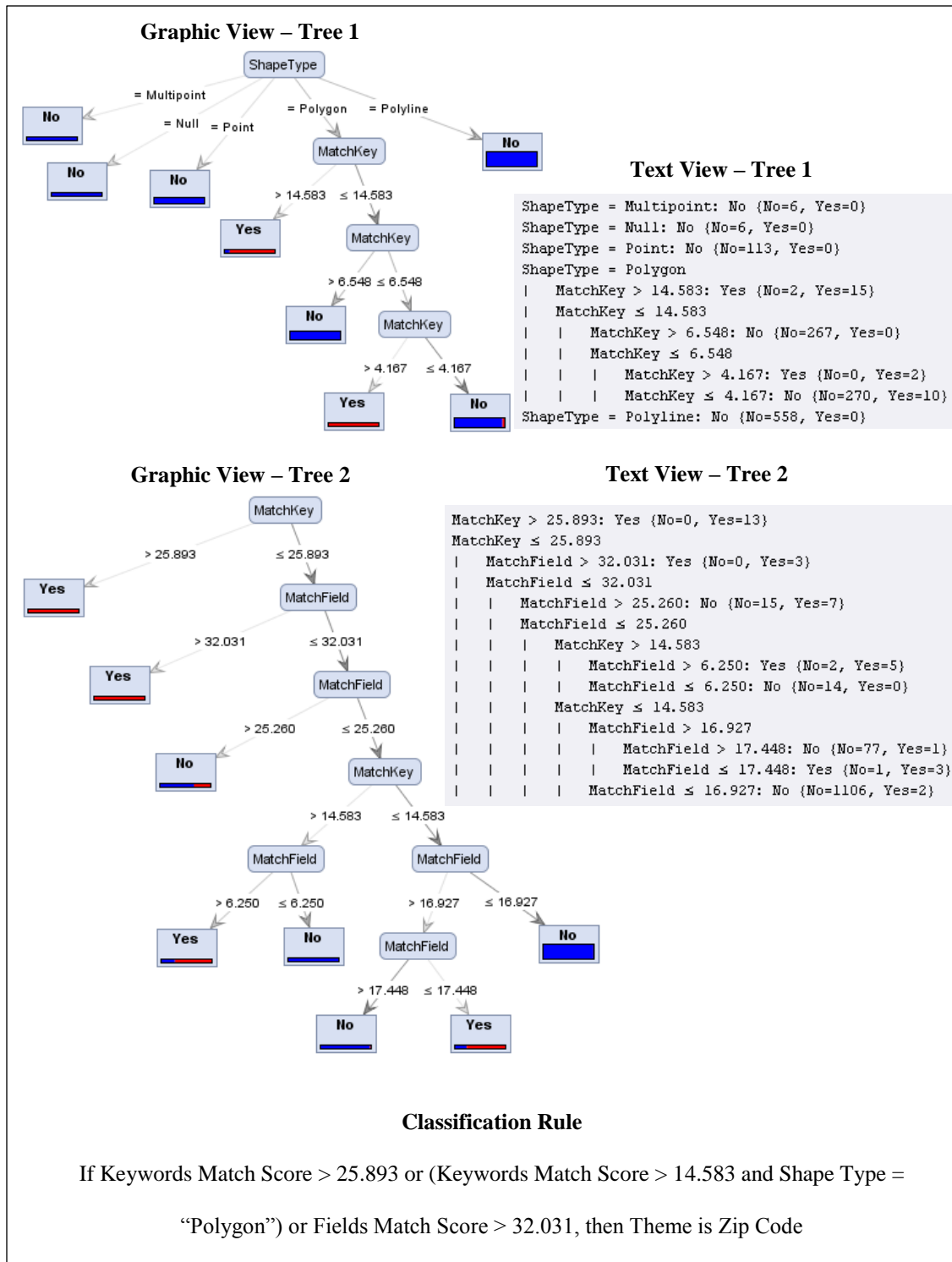


Figure 4.21: Decision Tree and Classification Rule for Zip Code Theme

Contour Theme

Figure 4.22 displays the graphic and text view, and classification rule derived from the data mining algorithm output for the contours theme. The decision trees reported that classifying contours by field names found in the geospatial datasets yielded the best results. Two classification pathways were identified that created the purest classes and are both found in the same decision tree.

The first pathway for classification creates a pure class (21 correct, 0 false-positives) by splitting into the class if field names match with a score greater than 31.559. The second pathway also relies on field names for matching, but the score must be greater than 14.259 and less than or equal to 15.209. The second pathway does seem to target specific datasets, potentially over fitting the training datasets, however, upon further investigation, this pathway correctly identifies 19 datasets and only has 2 false-positives which share very similar fields that seem to indicate that these datasets were derived from the same, or similar sources. Because of the disparity of the 19 datasets being fit, this pathway seems relevant and not over fitting.

The resulting classification rule from interpreting the random forest is: If Field Names Match Score > 31.559 or (Field Names Match Score > 14.259 and Field Names Match Score <= 15.209), then theme is Contours. This rule meets the criteria discussed earlier and performs well against a sample.

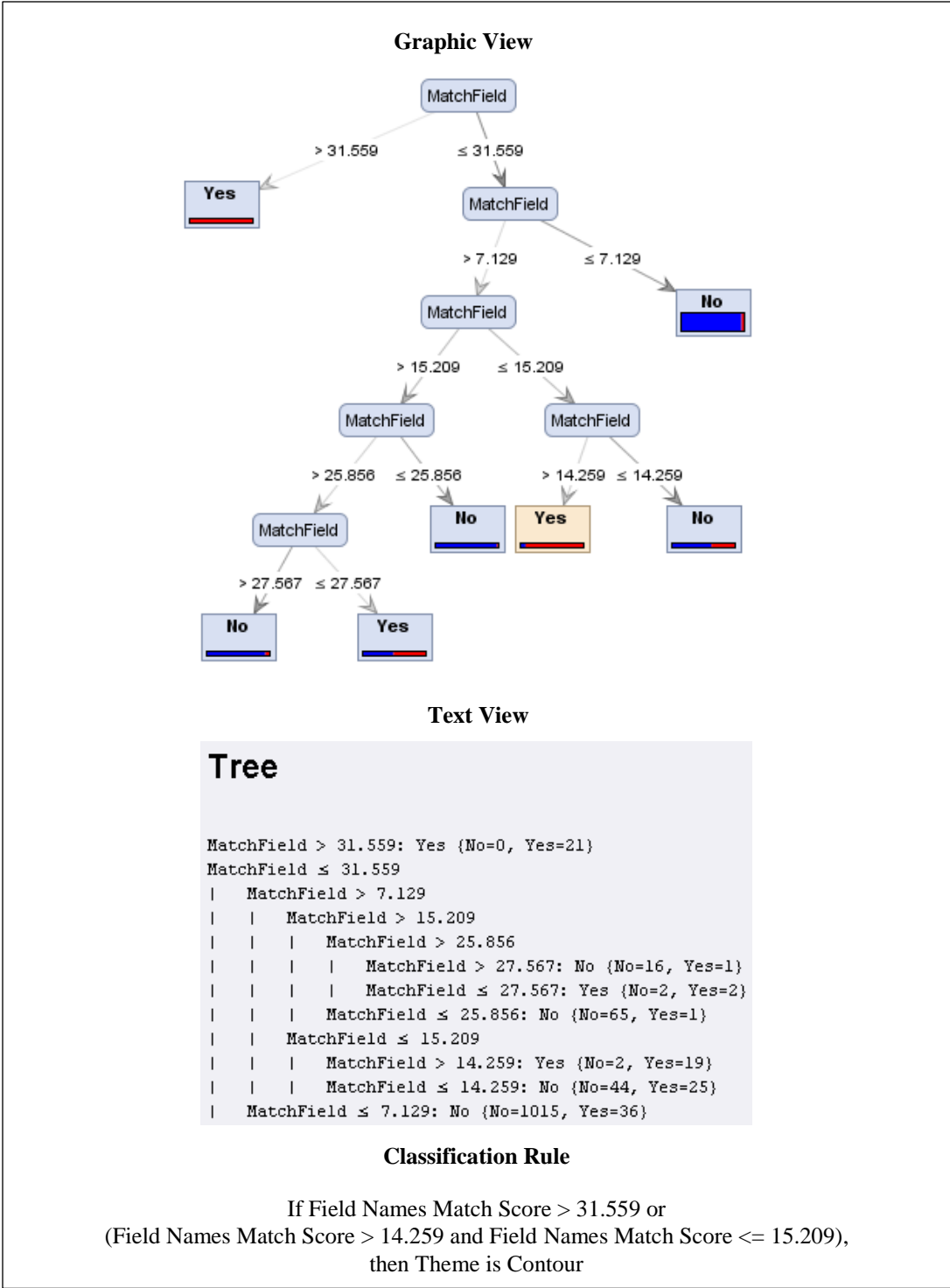


Figure 4.22: Decision Tree and Classification Rule for Contour Theme

Water Theme

Figure 4.23 displays the graphic and text view, and classification rule derived from the data mining algorithm output for the water theme. The decision trees reported that classifying water by keywords yielded the best results. As seen in the text view, if a geospatial dataset scores greater than an 8.841 in matching keywords with the water training dataset, a pure classification occurs with 39 datasets being correctly identified and 0 false-positives. What is unfortunate about this split, however, is the number of false-positives when keywords match less than or equal to 8.841. 157 false-positives occur by evaluating keywords for the water theme. The training dataset for water had a high ratio of unique keywords (233) to total keywords (1,198) which points towards a large number of words being used to describe water features with small amount of agreement between datasets' metadata.

The resulting classification rule from interpreting the random forest is: If Keywords Match Score > 8.841, then theme is Water. This rule meets the criteria discussed earlier, but does not score very well against the sample.

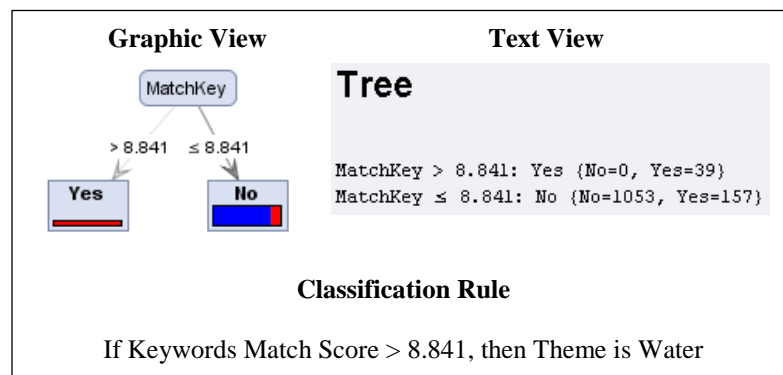


Figure 4.23: Decision Tree and Classification Rule for Water Theme

Airports Theme

Figure 4.24 displays the graphic and text view, and classification rule derived from the data mining algorithm output for the airports theme. The decision trees reported that classifying airports by both field names and keywords found in the geospatial datasets yielded the best results. Two classification pathways were identified that created the purest classes.

The first pathway for classification creates a pure class (13 correct, 0 false-positives) by splitting into the class if field names match with a score greater than 17.328. This pathway can be seen in the decision tree designated as “Tree 1” in Figure 4.21.

The second pathway, found in the decision tree designated as “Tree 2” in Figure 4.21, relies on keywords for matching, but the score must be greater than 32.716. This creates a pure class (10 correct, 0 false-positives).

Similarly to the results of the random forest for the water theme, a larger number of false-positives are yielded in both pathways. This can be attributed to the very high ratio of unique keywords (39%) and field names (49%) to total keywords and field names respectively. Like with the water theme, this points to little agreement on common terms between metadata documents.

The resulting classification rule from interpreting the random forest is: If Field Names Match Score > 17.328 or Keywords Match Score > 32.716, then theme is Airports. This rule meets the criteria discussed earlier, but does not perform well again a sample.

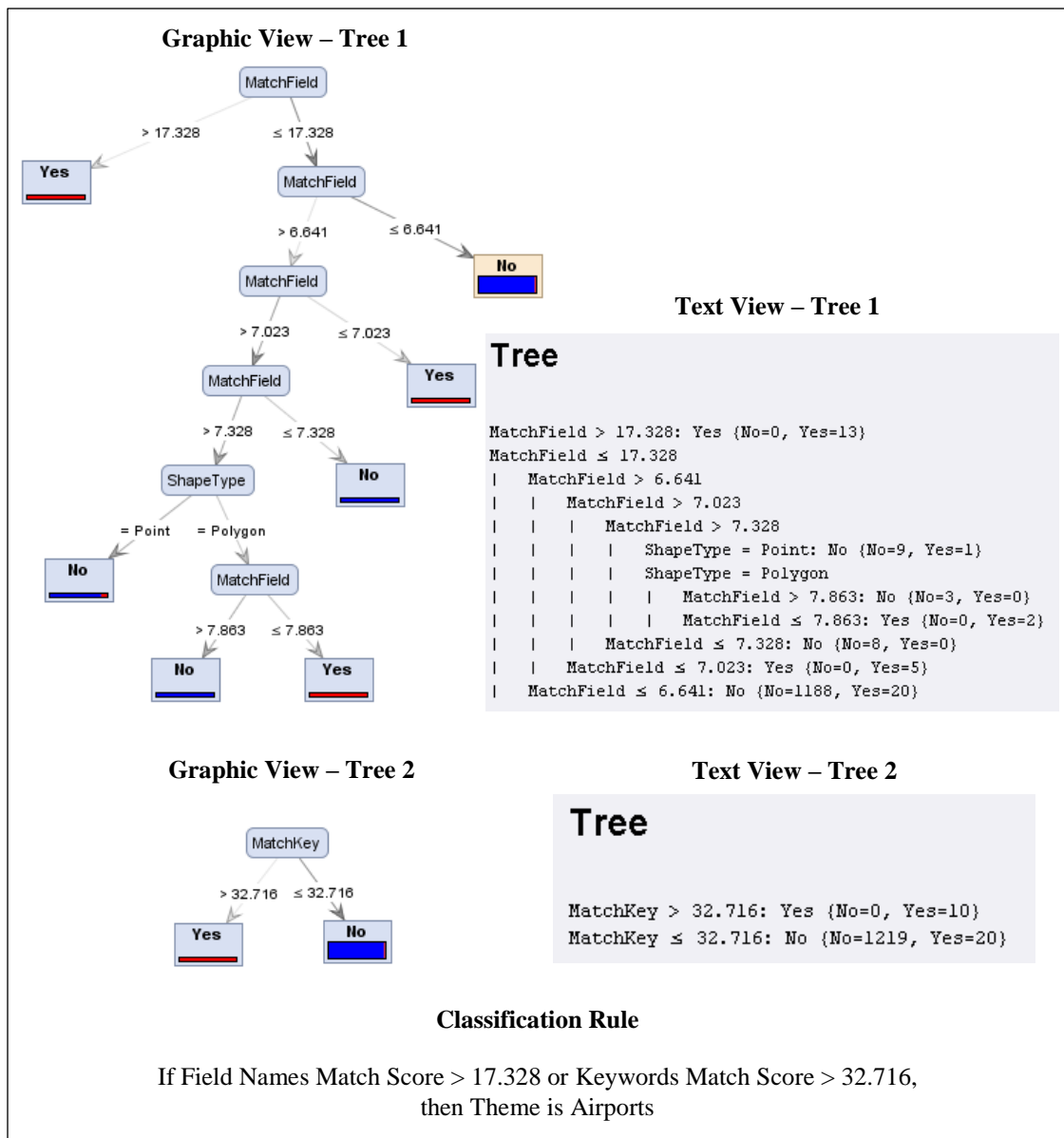


Figure 4.24: Decision Tree and Classification Rule for Airports Theme

Hospitals Theme

Figure 4.25 displays the graphic and text view, and classification rule derived from the data mining algorithm output for the water theme. The decision trees reported that classifying hospitals by keywords yielded the best results. As seen in the text view, if a geospatial dataset scores greater than 11.654 in matching keywords with the hospitals

training dataset, a pure classification occurs with 19 datasets being correctly identified and 0 false-positives. The other side of this split, however, does misclassify 12 hospital geospatial datasets as not a member of the hospital theme.

The resulting classification rule from interpreting the random forest is: If Keywords Match Score > 11.654 , then theme is Airports. This rule meets the criteria discussed earlier and scores slightly better than average against a sample.

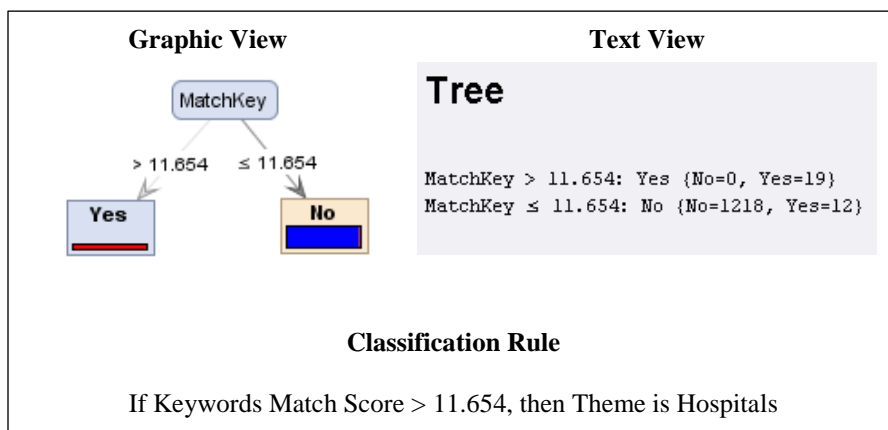


Figure 4.25: Decision Tree and Classification Rule for Hospitals Theme

Parks Theme

Figure 4.26 displays the graphic and text view, and classification rule derived from the data mining algorithm output for the parks theme. The decision trees reported that classifying parks by field names and keywords found in the geospatial datasets yielded the best results. Two classification pathways were identified that created the purest classes and are both found in the same decision tree.

The first pathway for classification creates a pure class (12 correct, 0 false-positives) by splitting into the class if field names match with a score greater than 13.743.

The second pathway uses on keywords for matching. For a geospatial dataset to be considered part of the parks theme, it must match keywords at a score above 22.407

The resulting classification rule from interpreting the random forest is: If Field Names Match Score > 13.743 or Keywords Match Score > 22.407, then theme is Parks.

This rule meets the criteria discussed earlier and performs slightly above average against a sample.

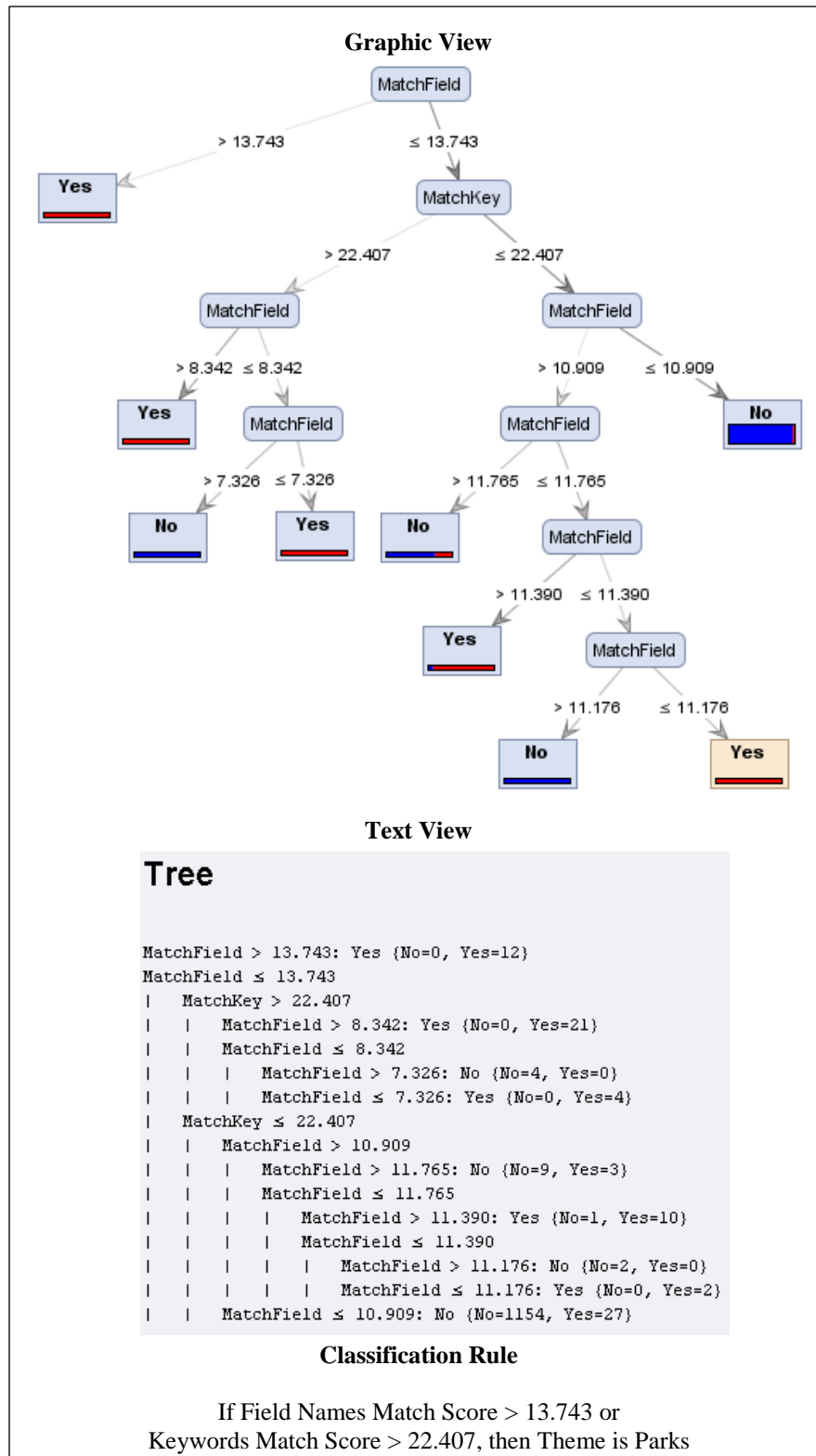


Figure 4.26: Decision Tree and Classification Rule for Parks Theme

Police/Fire Theme

Figure 4.27 displays the graphic and text view, and classification rule derived from the data mining algorithm output for the police/fire theme. The decision trees reported that classifying police/fire by both field names and keywords found in the geospatial datasets yielded the best results. Three classification pathways were identified that created the purest classes.

The first two pathways for classification are shown in the decision tree designated as “Tree 1” in Figure 4.24. The first pathway for classification creates a pure class (20 correct, 0 false-positives) by splitting into the class if keywords match with a score greater than 10.652. The second pathway creates a pure class (5 correct, 0 false-positives) by splitting into the class if keywords match with a score greater than 8.587 and the shape type is “point”.

The third pathway, found in the decision tree designated as “Tree 2” in Figure 4.21, relies on keywords for matching, but the score must be greater than 9.348. This pathway creates a pure class (28 correct, 0 false-positives). The split in this pathway does not successfully classify more police/fire datasets than it does not, however, it does provide a pure class and a reasonable amount of correctly classified datasets to be considered a valid rule.

The first and third pathways (Keywords Match Score > 10.652 and Keywords Match Score > 9.348) both rely on keywords and create pure classes. Because of this, the third pathway that requires a lower score match against keywords (9.348) allows for more correct classifications, and, therefore, the first pathway is dropped in preference for the third pathway.

The resulting classification rule from interpreting the random forest is: If Keywords Match Score > 9.348 or (Keywords Match Score > 8.587 and Shape Type = “Point”), then theme is Police/Fire. This rule meets the criteria discussed earlier, and performs average against a sample.

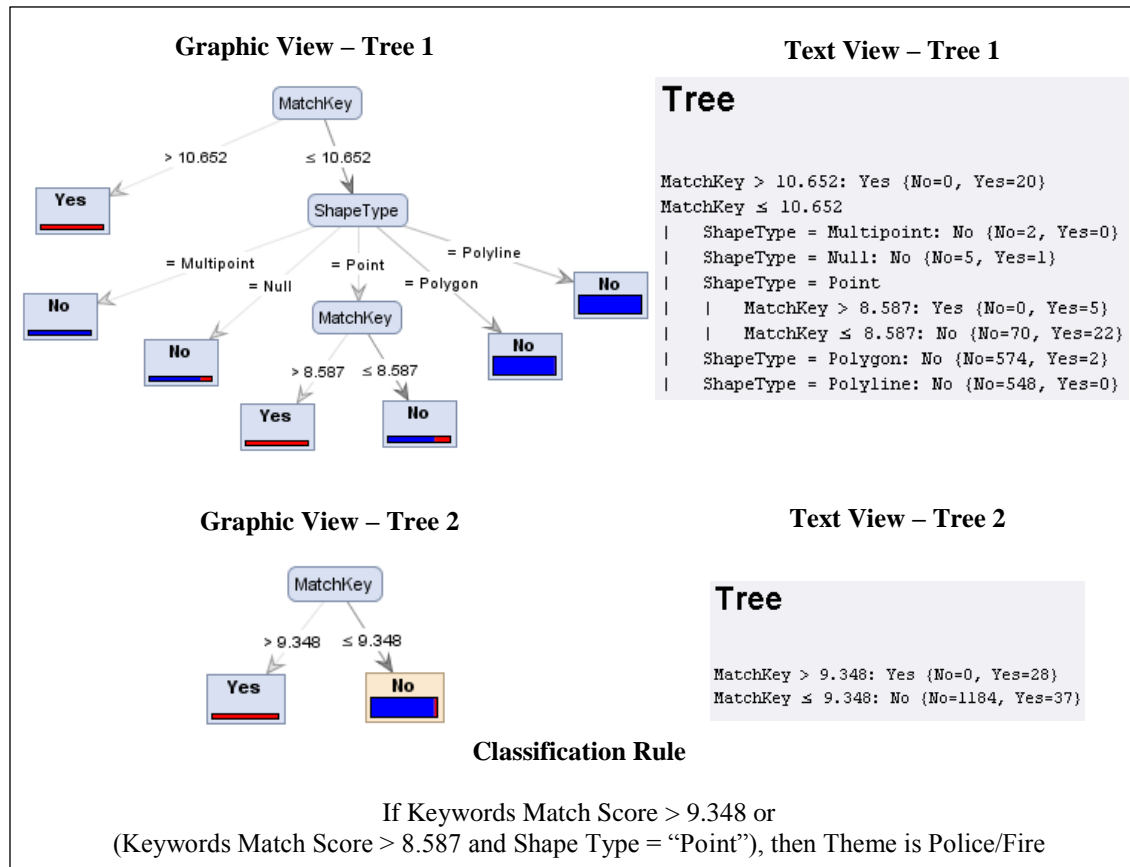


Figure 4.27: Decision Tree and Classification Rule for Police/Fire Theme

Railroads Theme

Figure 4.28 displays the graphic and text view, and classification rule derived from the data mining algorithm output for the railroads theme. The decision trees reported that classifying railroads by keywords yielded the best results. As seen in the

text view, if a geospatial dataset scores greater than 25.941 in matching keywords with the railroads training dataset, a pure classification occurs with 26 datasets being correctly identified and 0 false-positives. The other side of this split, however, does misclassify 61 railroads geospatial datasets as not a member of the railroads theme. This is likely due to the high ratio of unique keywords (36%) to the total number of keywords.

The resulting classification rule from interpreting the random forest is: If Keywords Match Score > 25.941, then theme is Railroads. This rule meets the criteria discussed earlier, however, it does not score well against a sample.

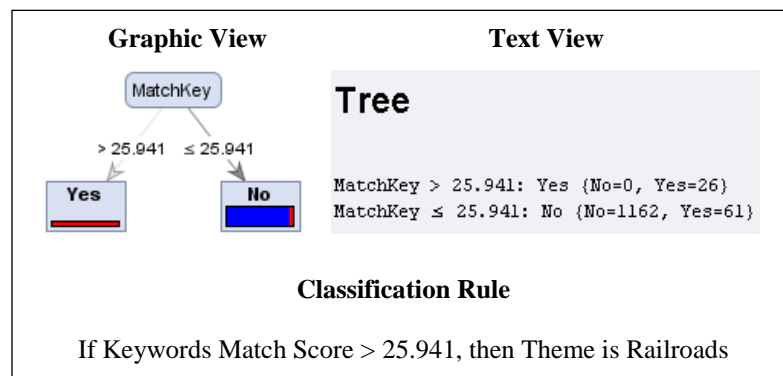


Figure 4.28: Decision Tree and Classification Rule for Railroads Theme

Roads Theme

Figure 4.29 displays the graphic and text view, and classification rule derived from the data mining algorithm output for the roads theme. Three pathways for classification were presented in the random forest.

The first pathway is identified as “Tree 1” in Figure 4.29. This pathway does not create a pure split, but the correct classifications (185) far outweigh the false-positives

(7). Datasets are considered to be a member of the roads theme if keywords match with a score higher than 8.158 and the shape type is “polyline”.

The second pathway, identified as “Tree 2”, uses field names to classify datasets into the road theme. This pathway creates a nearly pure split (108 correct, 1 false-positive) by classifying datasets into the road theme if field names match with a score greater than 5.006.

The third pathway, identified as “Tree 3” uses keywords to create a pure split (73 correct, 0 false-positives). If a dataset matches keywords with a score of at least 11.880, then it is considered a member of the roads theme.

The resulting classification rule from interpreting the random forest is: If (Keywords Match Score > 8.158 and Shape Type = “polyline”) or Field Names Match Score > 5.006 or Keywords Match Score > 11.880, then theme is Roads. This rule meets the criteria discussed earlier and scores well against a sample.

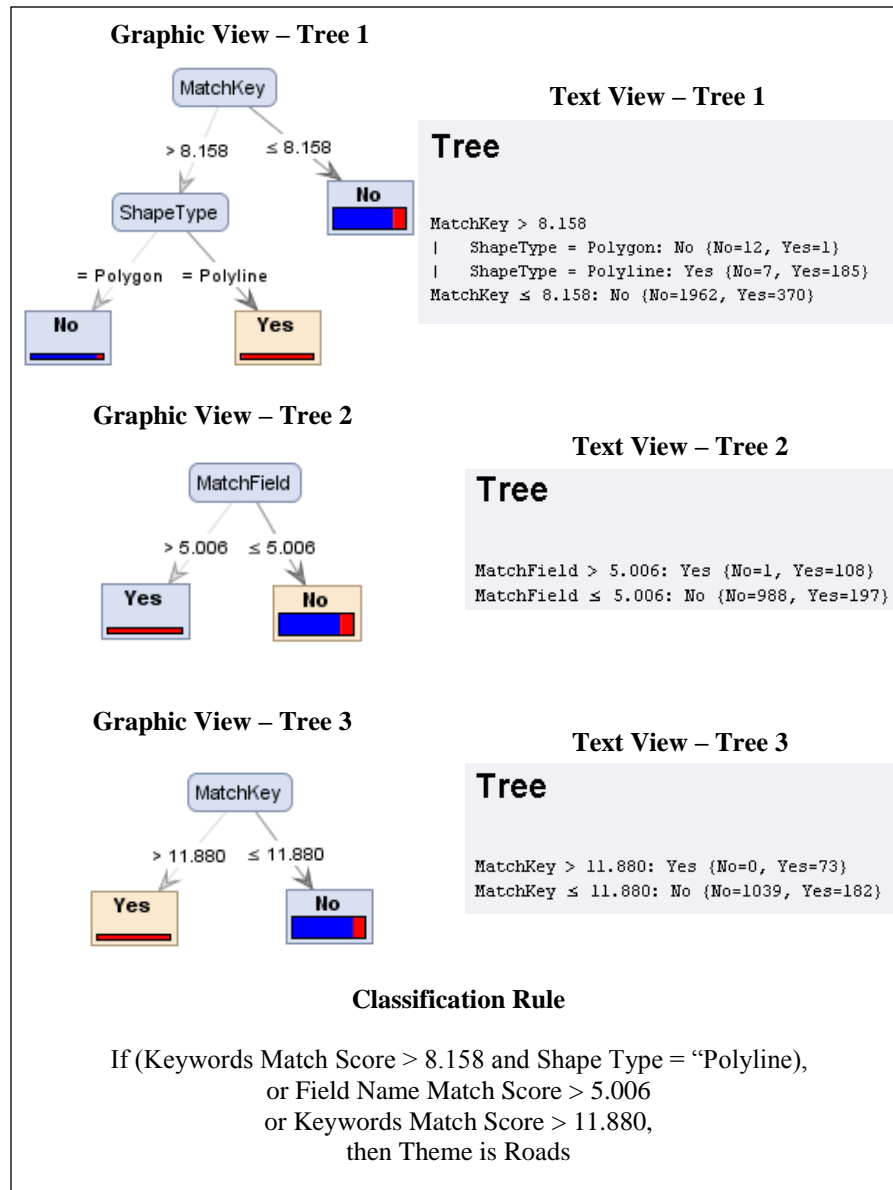


Figure 4.29: Decision Tree and Classification Rule for Roads Theme

Trails Theme

The final theme, trails, is shown in Figure 4.30 as a graphic and text view. The decision trees reported that classifying trails by keywords yielded the best results. As seen in the text view, if a geospatial dataset scores greater than 14.815 in matching keywords with the trails training dataset, a pure classification occurs with 9 datasets

being correctly identified and 0 false-positives. The other side of this split, however, does misclassify 32 trails geospatial datasets as not a member of the trails theme. This is likely due to the high ratio of unique keywords (48%) to the total number of keywords.

The resulting classification rule from interpreting the random forest is: If Keywords Match Score > 14.815 , then theme is trails. This rule meets the criteria discussed earlier; however, it does not score well against a sample.

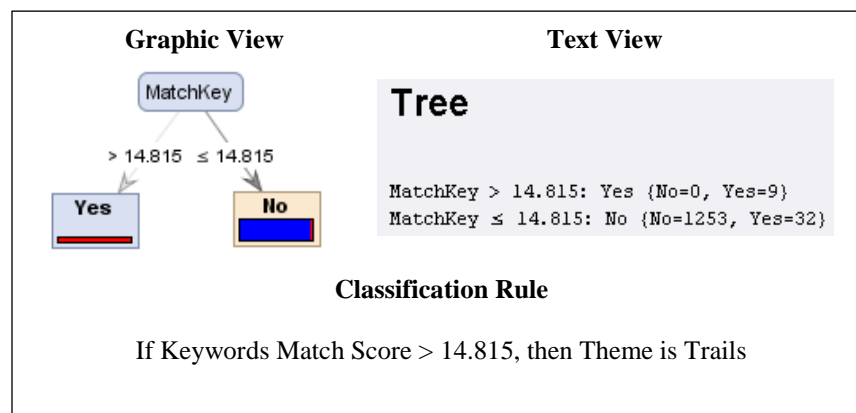


Figure 4.30: Decision Tree and Classification Rule for Trails Theme

Overall, the classification rules derived from the random forests was satisfactory in creating pure splits. Themes that had a high ratio of unique field names or keywords to total field names or keywords respectively did not seem to provide adequate information for the data mining algorithm to create pure classifications, or classifications that were able to classify a larger number of samples correctly than not. Additionally, themes with a small number of field names, or keywords did not provide much input information for successful classification. Another issue that seemed to prevent easy classification, is when a theme's (for example, the water theme) field names or keywords were generic in

description. Examples of generic field names are “name”, and “area” and examples of generic keywords are “physical”, and “environment”.

Accuracy Assessment

To assess the accuracy of the decision trees in identifying the theme of a dataset, a sample of datasets not included in the data mining process were tested against the dataset theme prediction program listed in Appendix D. For each theme, twenty datasets (120 datasets total) were tested to see if their themes would be correctly identified. Table 4.3 displays the results of the accuracy assessment and lists the percentage of datasets that were correctly identified out of the twenty tested datasets per theme.

Table 4.3: Theme Identification Accuracy Assessment Results

Theme	Correctly Identified (%)
State/County	35
Zip Code	90
Contours	80
Water	65
Airports	75
Hospitals	60
Parks	90
Police/Fire	70
Railroads	40
Roads	85
Trails	50

The results of the accuracy assessment shown in Table 4.3 displays that the majority (8 out of 11) of themes were identified correctly at a reasonable rate of 65% and above. Sample datasets of the Zip Code and Parks themes were identified correctly 90% of the time and were the highest performing themes for identification. These two themes

seemed to perform so well because they both had a few keywords that were extremely common across all input datasets. Having this common keyword that is not shared with other themes really set these themes up for high accuracy in theme identification.

Roads, Contours, Airports, and Police/Fire themes were identified correctly in the ranges of 70% - 85% which shows a good performance for identification. The Water and Hospitals themes at 65% and 60% correctly identified are still considered reasonable identification percentages, but are nearing the point where the performance is considered undesirable. These themes performed adequately as they contained enough specific shared keywords and field names that were not often found in other themes.

The State/County, Railroads, and Trails themes were identified at the rates of 35%, 40%, and 50% respectively. The low rates of identification for these three themes are not considered desirable for automatic identification. The reason the Railroads and Trails themes were identified so poorly seems to be the ambiguity in the keywords used in the metadata. The keyword “transportation” was used so commonly among the Roads, Railroads, and Trails theme that the scores for the Railroads and Trails theme could not overcome being identified as Roads because they did not have enough other uniquely identifying keywords with large scores. The State/County theme also displays a similar problem as the Railroads and Trails themes. The keywords for the State/County theme were generally very generic (boundary, polygon, information, area, etc...) and were shared by many other themes, chiefly the Parks theme.

Conclusions

The results of this research show that there is promise in data mining from geospatial datasets and metadata. However, as seen in this chapter, this process is only fruitful when themes have metadata documents with a low ratio of unique keywords to total number of keywords and a low ratio of unique field names to total number of field names. Additionally, any single keyword or unique field name that had a strong presence in a single theme made a large positive difference in accuracy of theme identification. The shape type as a predictor was useful in a few themes, but overall was not as useful as keywords or field names.

The surprising result of this research was how useful field names were in predicting dataset theme. Although the determinations based on field names were not overwhelmingly accurate, they did show significant promise to warrant additional research emphasizing field names in addition to metadata. One theory as to why field names proved useful was that since the majority of the downloaded datasets were in shapefile format, the attribute table restricted field names to eight characters. With so few characters defining a field name, supplemental information (such as a geographic modifier) were not included in the field name and instead, commonly used acronyms and abbreviations were more widely used.

With these results and lessons learned, future research should be undertaken to determine what other features of datasets and metadata could be data mined to increase the accuracy of feature identification, particularly in the State/County, Railroads, and Trails themes. Perhaps mining attribute entries for commonly used words would be useful. Additionally, increasing the number of datasets in the weaker theme categories

will be targeted to look for increases in accuracy as a result of a more comprehensive training set. Based on these findings, the promise of this research path continuing to bear fruit is promising.

Chapter Summary

The primary focus of data mining geospatial datasets was to determine whether datasets and metadata could be successfully data mined for use in identifying datasets commonly found on general reference maps to assist in determining cartographic symbol choices. The research discussed in this chapter shows that data mining metadata of geospatial datasets can lead to successful classification of datasets into themes.

Keywords and field names found in the metadata documents provided the most relevant predictors, while shape type provided some significant prediction information for a few themes.

CHAPTER 5

CONSTRUCTION OF THE EXPERT SYSTEM

Introduction

To reach the partial automation of the cartographic design process, the symbol design decisions that cartographers make when constructing a map must be emulated by the computer. The challenge in having the computer emulate a cartographer is translating the human process of symbol design into rules and heuristics that can be acted upon by the computer. Cartographic design is an ill-structured problem that cannot be solved effectively by conventional computer programming. As conventional computer programming is not suited for emulating a cartographer's decisions, this research employs expert systems technology to store, evaluate, and report facts, rules, and results with regard to map symbol design. The purpose of expert systems is to assist users in a decision on the basis of stored knowledge and is designed to deal with ill-structured problems such as symbol design choices. This chapter discusses an expert system that was created in order to emulate the decisions a cartographer makes when designing a general reference map.

Applicability of Expert Systems to the Cartographic Symbol Design

For this research, expert systems are employed to assist new and experience map makers by automatically choosing cartographic symbols for data sets added to a map. Applying expert systems technology for choosing cartographic symbols offers three

major benefits to users. The first benefit is that an expert system provides a user with the expert knowledge they seek without needing a human expert in the room. The second benefit is an expert system always being available whereas the human counterpart may be absent. The third benefit is that one expert system can contain the knowledge of multiple human experts, thus, acting as a human expert multiplier. In the context of map making, an expert system contains a knowledgebase of cartographic domain knowledge that it can intelligently apply user's unique mapping problem.

While expert systems seem to offer great benefits to their users, the question of whether an expert system is applicable and useful for solving cartographic problems must be addressed. Giarratano and Riley (2005) pose six questions that must be answered to determine if a knowledge domain is appropriate fodder for an expert system. These six questions will now be addressed to validate the application of expert system technology to the cartography domain of knowledge.

The first question posed by Giarratano and Riley is "Can the problem be solved effectively by conventional programming?" It is impossible to know all combinations of data that could be added to a map *a priori*. Additionally, the number of possible combinations may grow so large that a conventional computer program will quickly become extremely complex in structure as each combination would require a significant amount of programming structure. In an expert system, handling combinations of data does require additional facts and rules, however, it does not require additional complexity as an expert system design is flat compared to conventional programs. A second reason why conventional programming is not effective for cartographic design is because there is no efficient algorithm solution to the problem of automating a map; cartographic

design constitutes an ill-structured problem and is the type of problem that expert systems are designed to address. Expert systems are designed to compare facts presented with rules in order to infer which actions should be taken. In sum, conventional programs are not well suited to solve the complex problem of cartographic symbol design.

The second question to consider is “Is the domain well bounded?” While expert systems can hold any amount of knowledge, it is important to “stop somewhere” when enough knowledge is entered to successfully solve problems presented in the knowledge domain. The amount of cartographic knowledge available in the human brain and texts alone is far beyond the capacity of computers (Muller and Zeshen 1990) and well beyond a reasonable scope for this research. Additionally, since the expert system in this research is primarily focused on creating a proof of concept and many professionals experience with expert systems recommend, the domain of this first expert system will be constrained to a reasonably-sized domain of knowledge. For the expert system created in this research, there are two bounds creating a reasonable scope. The first bound is that general purpose maps are the only type of map being considered. The second bound is that only symbol designs will be chosen by the expert system. These two bounds will keep the size and complexity of the expert system in check for this research.

The third question is “Is there a need and a desire for an expert system?” As discussed in Chapter 1, there is a large population lacking in cartographic knowledge and training which needs to be remedied. More people are making maps today than ever before, however, the opportunities for people to learn proper cartographic techniques is diminishing (Goodchild 2000; Wood 2003; Krygier 2005). An expert system will provide the needed knowledge to untrained users who wish to create a general purpose

map but may not know how to properly symbolize the graphic elements. By providing cartographic expertise to map makers through an expert system, expert cartographic knowledge can be conveyed to a wide audience in a proactive manner.

“Is there at least one expert who is willing to cooperate?” is the fourth question to consider. Yes, there are multiple cartographers willing to assist in the development of this expert system. In addition to cartographers, cartography texts, and published maps can be referenced for design knowledge.

The fifth question is “Can an expert explain the knowledge so that it is understandable by the knowledge engineer?” Yes, as discussed in Chapter 2, cartographic expert systems have been successfully produced thereby proving possible to capture expert cartographic knowledge in an expert system. The knowledge engineer for this research is familiar with cartographic concepts and will be able to understand the technical terminology used by the expert cartographers, texts, and maps.

The sixth and final question posed to determine whether the knowledge domain is appropriate for expert systems is “Is the problem-solving knowledge mainly heuristic and uncertain?” Cartographers utilize their experiential knowledge during map composition. The knowledge of a cartographer is largely heuristic and dependent upon many variables that may not be easily definable. Expert systems are designed to work with heuristic knowledge and provide facilities to work with many complex situations.

The answers to the six questions posed above illustrate that expert systems are well suited to work with the domain of cartographic knowledge. The ill-structured and heuristic nature of cartographic knowledge is an excellent domain for modeling in an expert system. However, an important caveat should be noted. A practical limitation of

expert systems is a lack of casual knowledge (Giarratano and Riley 2005). It is easier to program expert systems with shallow knowledge based on experience and empirical knowledge than the complex underlying causes. A heuristic is an experience-based technique for problem solving when an exhaustive search for a solution is impractical (Russell and Norvig 2003). Heuristic methods allow the expert system to remain shallow in knowledge as a heuristic does not require perfect data or perfect solutions; it aids in the solution with a guarantee of success. Even if an exact solution is known, a heuristic may still be the preferred method of solving the problem because of time or cost constraints. As heuristics provide a “rule-of-thumb” method, and can be based on generally correct assumptions and do not require expensive overhead in knowledge modeling, a heuristic approach has been adopted for the facts and rules in the expert system created for this research.

Obtaining Expert Knowledge

As the cartographic expert system is being designed to emulate the expert knowledge of human cartographers, a translation from human knowledge to computer structures must be undertaken. For this research, three sources of human knowledge of cartography were targeted: cartographers, cartography textbooks, and published general reference maps. The reason for targeting multiple sources of cartographic knowledge was to reduce the possibility for an individual bias in the knowledgebase. To remove the potential for bias, each source was evaluated against the other sources of knowledge to identify commonalities so that the “safest” cartographic knowledge was obtained. The obtained expert knowledge does not purport to contain completely unbiased cartographic

knowledge, as cartography does require a level of human subjectivity and the bias of the author compiling the knowledge may also introduce their own biases.

Obtaining the cartographic expert knowledge was an iterative process of researching cartographic information expressed in textbooks, critically evaluating the symbology and map design of published maps, and discussing cartographic conventions with expert cartographers. The ontology discussed in Chapter 3 was used to focus on the relevant concepts required to properly symbolize a general reference map. Specifically, four top-level classes, and one subclass were focused upon for knowledge recording. The classes focused on were `VisualVariable`, `Graphic`, and `SpatialPhenomenon`, `MapScale`, and `MapBody`. These five classes were identified as the minimum concepts required to choose symbology for a general reference map and concepts obtained from expert sources were made to fit within the concepts of the selected ontology classes.

With the sources of cartographic knowledge identified, the process of obtaining the knowledge began. The cartographic expertise was initially recorded informally in the form of IF-THEN statements and lists of commonly used visual variables. For each general reference map theme, all three sources of information (textbook, maps, and cartographer) were consulted and multiple IF-THEN statements and lists were created. Lastly, these multiple recordings of knowledge were reviewed to find intersecting ideas that would create a safe representation of the general reference map theme.

The combined IF-THEN statements and lists were next converted into facts and rules structures useable by the expert system. Typically, for each cartographic concept obtained from expert sources, both a fact, and a rule would be created; the fact representing the cartographer's knowledge, and the rules representing actions the

cartographer would take based on their knowledge (facts and rules are discussed in more detail in the following section of this Chapter).

To illustrate how this process of translating knowledge to expert system data structures, an example will be presented. This example discusses how the problem of dataset draw ordering was represented and solved in both the informal and expert system structure.

In this example, published general reference maps were evaluated to determine which datasets were placed in what draw order. For each map, the draw order was recorded in a list format similar to the list shown in Figure 5.1. Examples of published maps used for determining draw order are the Blue Ridge Parkway Map (National Park Service 2006), the Texas Official Travel Map (Texas Department of Transportation 2007), The Official Map of Conyers and Rockdale County Georgia (Hodler 2007), and Georgia Highways & Interstates (Rand McNally 2004). After the maps were analyzed, cartography textbooks were referenced in regard to layer draw orders. Three cartography textbooks were referenced for this research: “Cartography: Thematic Map Design” (Dent, Torguson, Hodler 2009), “Elements of Cartography” (Robinson *et al.* 1995), and “Thematic Cartography and Geovisualization” (Slocum *et al.* 2008). In these textbooks, figures of maps were analyzed, and chapters on design of reference maps, layout design, and figure ground relationship were referenced for relevant information. Lastly, the compiled list of information was evaluated by cartographers to determine, in general, which theme would be placed at what position in the drawing order of the map. Two cartographers were used for this research, Dr. Joseph Loon, Blucher Endowed Chair of Surveying and Professor of Geographic Information Science at Texas A&M University –

Corpus Christi, and the author of this research. The information obtained from the published maps and textbooks were reviewed by the cartographers. The result from the cartographers was a single IF-THEN statement and an ordered list representing the determined draw order of each dataset theme as displayed in Figure 5.1.

<u>General Drawing Order of Dataset Themes</u> 1. Hospitals 2. Emergency 3. Airports 4. Roads 5. Railroads 6. Trails 7. Contours 8. Water 9. Parks 10. StateCounty
<u>Ordering Logic</u> IF Target Dataset Theme has a higher draw order (lower number) than a Dataset Theme above THEN place Target Dataset Theme one place above.

Figure 5.1: Obtained Cartographic Information in Initial IF-THEN and List Format

With the cartographic knowledge paired down to the final version informal version, it was converted into the facts and rules structures used by the expert system as displayed in Figure 5.2. The expert system structure required three parts for this translation: a template for the knowledge (deftemplate), the knowledge (deffacts), and the rule (defrule). The deftemplate and deffacts were converted from the list of the drawing orders displayed in Figure 5.1, and the deffacts were converted from the IF-THEN statement.

```

(deftemplate MAIN::theme-ordering
  (slot theme (default ?NONE))
  (multislot above))

(deffacts MAIN::initial-theme-ordering
  (theme-ordering (theme Parks) (above StateCounty))
  (theme-ordering (theme Roads) (above Water Parks StateCounty
    Trails))
  (theme-ordering (theme Water) (above Boundaries Parks))
  (theme-ordering (theme StateCounty) (above))
  (theme-ordering (theme Contours) (above Parks StateCounty))
  (theme-ordering (theme Airports) (above Parks Roads Water
    StateCounty Contours Trails Railroads))
  (theme-ordering (theme Trails) (above Contours Parks
    StateCounty))
  (theme-ordering (theme emergency) (above Parks Roads Water
    StateCounty Contours Trails Railroads))
  (theme-ordering (theme Hospitals) (above Parks Roads Water
    StateCounty Contours Trails Railroads))
  (theme-ordering (theme Railroads) (above Parks Roads Water
    StateCounty Contours Trails)))

(defrule MAIN::order-mapLayers
  ?mapLayer1 <- (mapLayer (theme ?theme) (drawOrder ?loc1))
  (theme-ordering (theme ?theme) (above $? ?abovetheme $?))
  ?mapLayer2 <- (mapLayer (theme ?abovetheme) (drawOrder ?loc2))
  (test (> ?loc1 ?loc2))
  =>
  (modify ?mapLayer1 (drawOrder ?loc2))
  (modify ?mapLayer2 (drawOrder ?loc1)))

```

Figure 5.2: Obtained Cartographic Information in Expert System Structure

This process of obtaining cartographic knowledge was repeated for representing the concept of a map layer, describing colors, choosing colors for each theme, assigning symbology for each theme, and defining and setting the map scale. In all cases, these concepts were recorded in an informal fashion to maintain a high level of human readability. Only after all sources of knowledge has been referenced would the process of translating the knowledge into an expert system structure take place. A more detailed explanation of the expert system structures and construction will take place in the next

section and will cover the base concepts of an expert system followed by details of each structure created for the cartographic expert system.

Methodology

Expert systems are “an intelligent computer program that uses knowledge and inference procedures to solve problems that are difficult enough to require significant human expertise for their solution” (Edward Feigenbaum in Giarratano and Riley 2005, 5). That is, the goal of an expert system is to emulate the thought process of a human to arrive to the same solution of a human expert. At a high level, expert systems are composed of two parts: a knowledge base, and an inference engine. The expert system uses the knowledge stored in the knowledge base, and facts entered by the user to infer an expert answer using the inference engine. Expert systems have been used for many different applications in many different knowledge areas and have successfully emulated the decisions made by experts. The history, further explanation, and applications of expert systems were discussed in more detail in Chapter 2.

This methodology section is split in two sub-sections. The first sub-section, CLIPS Expert System, will describe relevant aspects of the expert system software used to build the cartographic expert system. The second sub-section, Program Design of Cartographic Expert System, will detail the expert cartographic knowledge base and rules programmed into the expert system.

CLIPS Expert System

Many expert systems exist on the market and are composed of different elements and store knowledge in different ways. Commercially available and freely available

expert systems were reviewed for suitability to the goals of this research. The expert system C Language Integrated Production System (CLIPS) was chosen for this research. “CLIPS is a multi-paradigm programming language that provides support for rule-based, object-oriented, and procedural programming.” (Giarratano and Riley 2005) CLIPS was originally designed as a rules-based production system, but later included procedural and object-oriented language support. In its current version (6.24), CLIPS provides all of the functions required to build a cartographic expert system.

CLIPS was chosen as the expert system for this research for four reasons. First, CLIPS is a mature expert system that has been publically available since 1986. Second, CLIPS is in the public domain and is freely available. Third, a CLIPS extension module, named PyCLIPS (Garosi 2008) is available and allows for CLIPS to interface with the Python programming language, thereby allowing CLIPS to work with the programs written for automatic geospatial data set theme identification in Chapter 4. The fourth reason CLIPS was chosen was the way that the knowledge was stored in CLIPS was intuitive and resembled the structure of cartographic knowledge.

CLIPS is composed of many components, however, seven components are primarily used in this research as they provide all the required functionality. These seven components are: facts, fact templates, fact list, default facts, knowledge base, agenda, and inference engine. Each of these seven components will be discussed in the following sections.

Facts

A fact is an item of knowledge and may be entered by a user, or stored as a default item of knowledge used to initialize the expert system. Facts serve two important

purposes. First, facts record a portion of the expert knowledge embedded in the expert system. Second, facts provide as records of input from the user and record outputs from the inference engine; facts transfer knowledge between the user and the expert system.

In CLIPS, facts are represented by a relation name, followed by zero or more slots. A relation name is a symbol (contiguous string of printable characters) that relates to a fact template (fact templates discussed below). Slots are sets of key/value pairs of information and each slot may contain multiple values related to a single key. Figure 5.3 displays an example of a fact stored in CLIPS. In the example fact, “value-range” is the relation name, and the three slots are named “name”, “value-low”, and “value-high”. Each of the three slots has related values of “medium-high”, “70”, and “80” respectively. The entire fact and each slot within the fact are surrounded by parenthesis that delineate where each fact and slot begin and end.

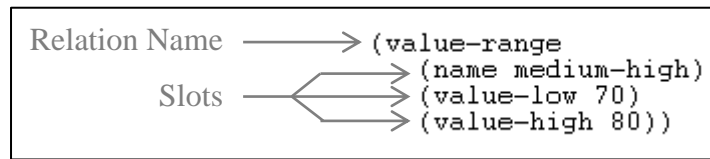


Figure 5.3: Example Fact Stored in CLIPS

Each relation name, slot name, and slot value are known as fields. A field is also known as a primitive data type and CLIPS supports the following primitive data types: float, integer, symbol, string, external address, fact address, instance name, and instance address. Float and integer store numeric information. A symbol is a contiguous string of printable characters. A string is any number of characters within double-quotes.

External addresses represent a pointer to an external data structure returned by an external function; external addresses are not used in this research. Fact addresses, instance names, and instance addresses refer to specific facts and object data structures (instances) within the expert system.

Fact Template

A fact template is a defined data structure that specifies a list of valid slots for a given relation name. Facts must have a fact template defined before they are allowed to be asserted (inserted) into the fact list. In CLIPS, a fact template is known as a “deftemplate” which is short for “default template”. Deftemplates are static data structures that are not typically created or deleted during or between runs of an expert system program.

Each deftemplate must have a relation name specified. Any fact that wishes to use the data structure defined by a deftemplate must start with the same relation name as the deftemplate. Each deftemplate may have many facts related to it using the relation name. Also, a deftemplate cannot be deleted from the expert system until all related facts are first removed from the fact list.

There are two types of slots in CLIPS: slot and multislot. A slot only holds a single value. A multislot holds any number of ordered values separated by spaces. In addition to the two types of slots, each slot can optionally be specified with slot attributes. A slot attribute provides typing and constraint checking on the information being stored in the slot. There are multiple slot attributes available in CLIPS, but only four are used in this research and will now be discussed: default, type, range, allowed-symbols.

The default slot attribute sets the default value of the slot if the fact is asserted without a value provided for the slot. The default slot attribute can contain three possible values: ?NONE, ?DERIVED, or a field value. A default slot value of ?NONE requires that the fact be asserted with a value as it will not accept an empty slot. If ?DERIVED is set as the default, then CLIPS will assign a random value that satisfies all of the slot attributes. If a field value is entered as the default slot attribute, then the entered value will be the value of the slot if no value is entered for the slot when the fact is asserted.

The type slot attribute defines the type of field that the slot will hold. Valid range slot attribute values mirror the primitive field data types.

The range slot attribute constrains the value stored in the slot to a value between and inclusive of the two defined numeric values. The range slot attribute can contain two possible values: ?VARIABLE, or a numeric values. ?VARIABLE represents either no maximum or minimum value.

The allowed values slot attribute specifies an exhaustive list of all values that are allowed to be stored in the slot. The allowed values slot attribute can have two possible values: ?VARIABLE, or a single, or list of possible values. If ?VARIABLE is defined, then any value of the defined field type is allowed, otherwise, only values that match values listed as an allowed value are allowed to be stored in the slot.

The basic syntax for a deftemplate is shown in Figure 5.4. This example deftemplate has the relation name “value-range”. The “value-range” deftemplate has three slots defined each having associated slot attributes. For example, the slot named “value-high” can contain integer values between 0 and 100 inclusive and also has the

default value of ?NONE which means that any fact using this deftemplate must have a value specified for the value-high field.

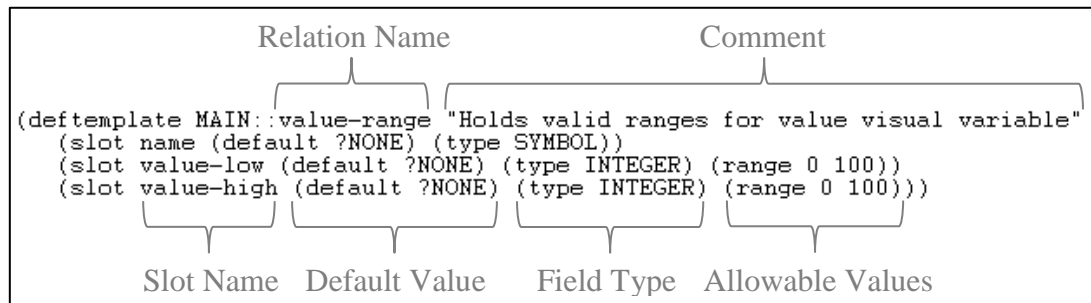


Figure 5.4: Basic Syntax of deftemplate

Fact List

A fact list contains a record of all information with which the expert system can reason. In a fact list, each fact is provided a fact address which provides a way for the expert system to uniquely identify each fact. The fact list acts as the central repository for all knowledge that the expert system will reason with.

Figure 5.5 displays an example of facts stored in a fact list in the CLIPS expert system. The first column on the left is the fact address assigned to each fact by CLIPS. Fact addresses are identified by the lowercase letter 'f' followed by a dash and the fact's unique identifier number. Fact address numbers start at zero and increase sequentially by one. If a fact is deleted, the deleted fact's address is not reused by CLIPS.

f-49	(value-range (name high) (value-low 90) (value-high 100))
f-50	(value-range (name medium) (value-low 50) (value-high 60))
f-51	(value-range (name medium-high) (value-low 70) (value-high 80))
f-52	(value-range (name medium-low) (value-low 30) (value-high 40))
f-53	(value-range (name low) (value-low 10) (value-high 20))
f-54	(value-range (name none) (value-low 0) (value-high 0))

Figure 5.5: Example Portion of Fact-List in CLIPS

Facts in the fact list can be inserted, deleted, or modified by users or expert system rules. Facts are inserted using the “assert” command. When a fact is asserted, the fact is provided with a fact address and added to the fact list. Facts are deleted from the fact list using the “retract” command followed by one or more fact addresses. Facts are modified using the “modify” command followed by the fact address and new slot values. It is important to note that when a fact is modified, it is actually first retracted from the fact list, and then asserted as a new fact with original and modified values and a new fact address. This is important to note because of the property of refraction. Refraction means that a rule (discussed below) will not fire more than one time for each fact. When a rule is modified, it creates a new fact which can then fire a rule that it just fired, thereby creating an infinite loop. Because of this issue, it is important that modified rules do not needlessly fire rules infinitely.

Default Facts

Default facts are facts that exist upon initialization of the expert system. Default facts seed the expert system with knowledge about the world state. This default world state could be the initial state of a problem set which the expert system is meant to solve, and/or facts relating to rules or experience contributed by knowledge domain experts.

In CLIPS, default facts are specified using the deffacts construct. A CLIPS program may have zero or more deffacts constructs. Each deffacts construct contains a

list of facts to add to the fact list upon initialization of the expert system. The facts contained in a deffacts construct must have a related deftemplate otherwise the fact will not be allowed to be inserted into the fact list. Figure 5.6 displays a portion of a deffacts construct that contains facts related to the theme-hsv-color-preferences deftemplate.

```
(deffacts MAIN::initial-color-preferences
  (theme-hsv-color-preferences (theme water) (geometry polygon) (hue blue) (saturation 100))
  (theme-hsv-color-preferences (theme trails) (geometry line) (hue brown) (saturation 100))
  (theme-hsv-color-preferences (theme parks) (geometry polygon) (hue green) (saturation 100))
  (theme-hsv-color-preferences (theme airports) (geometry point) (hue black) (saturation 100))
)
```

Figure 5.6: Example deffacts Construct

Knowledge Base (Rules)

The knowledge base of an expert system contains rules that execute when facts are entered into the fact list that meet the rule's execution requirements. Each rule is provided a name to differentiate the rules to CLIPS. Additionally, rules may be assigned a salience value that indicates the priority of the rule. Rules with higher salience values will fire before rules with lower salience values.

Rules are entered into CLIPS as a “defrules” construct and is composed of two parts separated by an arrow symbol (\Rightarrow): the left-hand side (LHS) which specifies conditional elements, and the right-hand side (RHS) which lists actions. The conditional elements listed in the LHS are a list of pattern matching constraints meant to match values in fact slots. The LHS can have one or more conditional elements. Only when *all* conditional elements on the LHS match facts in the fact list will the rule be activated and placed on the agenda (the agenda is discussed below). Once the rule fires from the agenda, the RHS of the rule performs the actions listed on the RHS of the rule.

Rules are the most complex part of the expert system designed for this research as they specify often complex conditional elements that must match before the expert system takes any action. It is important to understand how conditional elements are constructed on the LHS of a rule as they define when the expert system takes an action. Conditional element pattern matching will now be discussed in two sub-sections so the reader will be able to better understand how expert systems work and to interpret the rules defined later in this chapter. The first sub-section will discuss the basics of conditional elements matching against a single fact. The second sub-section will discuss more advanced conditional elements matching against multiple facts simultaneously.

Basics of Conditional Elements

Conditional elements specify which slots and values of a fact it wishes to match. For instance, let's consider the case of a family tree. For this family tree, we wish to find certain people in the tree based on information about those persons, such as first name. In the expert system, each person would be represented as a fact in the fact list, and each person would have the data structure of the "Person" deftemplate. For our basic family tree, the "Person" deftemplate and three deffacts are listed in Figure 5.7. This basic family tree has three persons expressed as three facts based on the "Person" deftemplate. In this family tree, Kenneth has the son Rick, Rick has the son Billy, and Billy has no son (son is not specified and assumed nil). For simplicity of the following examples, we will also make the assumption that no two people in this family tree will have the same names.

```
(deffacts MAIN::fathers-and-sons
  (Person (first-name Kenneth) (son Joe))
  (Person (first-name Joe) (son Billy))
  (Person (first-name Billy)))

(deftemplate MAIN::Person
  (slot first-name (default ?NONE))
  (slot son (type SYMBOL)))
```

Figure 5.7: Example deffacts and deftemplate for Basic Family Tree

With the facts created, we can now provide the expert system with conditional elements to match facts with and subsequently fire rules. Two basic rules will be displayed and explained. The first rule will provide a very basic example of matching facts to conditional elements. The second rule will demonstrate a slightly more complicated rule that includes variables and a field constraint.

The first rule we will define will simply seek to find the person named “Joe” in the fact list and print that Joe was found. The “find-Joe” rule is listed in Figure 5.8 and has one conditional element in the LHS and one action in the RHS. The conditional element in the LHS matches facts related to the “Person” deftemplate. Specifically, the conditional element matches against one slot in the “Person” facts: the first-name slot. In this case, the LHS will match a “Person” fact that contains the symbol “Joe” in the first-name slot. In the case of our simple family tree, this rule matches a single rule, the “Person” fact with the slot first-name of “Joe”. With the LHS finding a matching fact, the rule activates and is placed on the agenda. When the agenda is run, one statement is printed stating: “The person ‘Joe’ was found.”

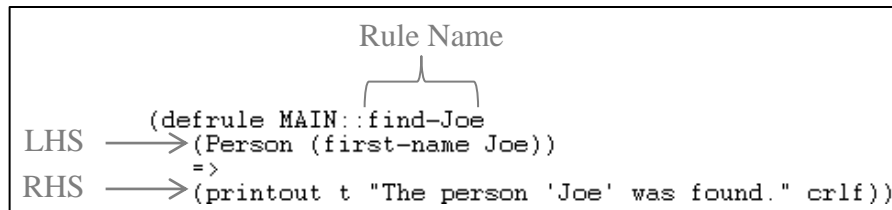


Figure 5.8: find-Joe Rule

The second rule we will define is a little more complex than the previous rule and will print all persons in the fact list that have sons. The “has-son” rule is listed in Figure 5.9 and has one conditional element in the LHS and one action in the RHS. The conditional element in the LHS matches facts related to the “Person” deftemplate. Specifically, the conditional element matches against two slots: the first-name slot, and the son slot. In the LHS of a rule, variables are designated by printable characters proceeded with a question mark (*e.x.* ?name). When matching the first-name slot on the LHS side of the “has-son” rule, the value found in the slot is stored in the variable ?name. Since a variable (?name) was specified as the value to match against the slot, any fact that has a value in the first-name slot will match (and be stored in the ?name variable). In this case, (first-name ?name) matches all three people in our facts list as all three people have values in the first-name slot. The second slot the conditional element matches against, son, is more restrictive than the match against the first-name slot. The conditional element (son ?son&~nil) matches all facts that *do not* have the value “nil” in the son slot. The tilde symbol (~) is a negation operator that, in this case, means “not the value ‘nil’”. The ampersand is an “and field constraint” operator and only binds the value stored in the slot to the variable to its left if the condition following the ampersand is satisfied. In this case, (son ?son&~nil) reads “store the value in the ‘son’ slot into the ?son variable if the

value stored in the slot is not ‘nil’”. In this case, (son ?son&~nil) matches only two people in our fact list: Kenneth and Joe.

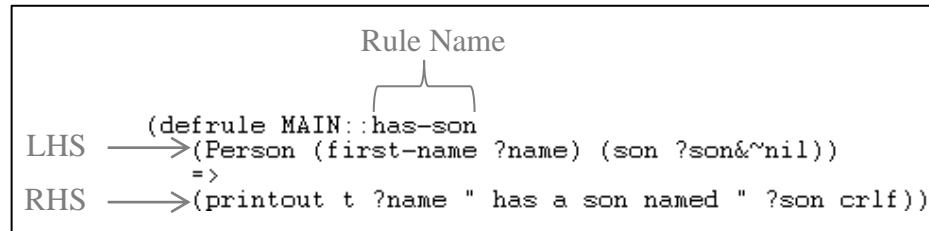


Figure 5.9: has-son Rule

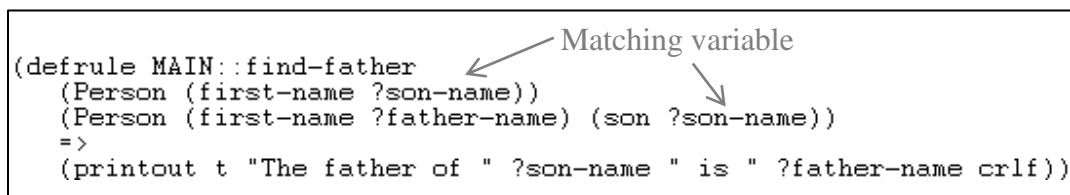
With the “Person” facts Kenneth and Joe matching the LHS of the “has-son” rule, the rule will activate and be placed on the agenda. When the agenda is run, two statements will be printed stating that both Kenneth and Joe have sons followed by their respective son’s names.

Multiple Conditional Elements in the LHS

Matching conditional elements against single facts in a rule is useful for simple situations, however, the real power of conditional elements are shown when multiple facts need to be considered to activate a rule. This sub-section will discuss two examples of conditional elements in rules that match against multiple rules at once. The basic family tree shown in Figure 5.7 will be used as the deftemplate and facts that the conditional elements will match against.

The first rule defined seeks to find the father of a person if the father exists in the basic family tree. The “find-father” rule is listed in Figure 5.10 and has one two conditional elements in the LHS and one action in the RHS. To find the father of a person, the LHS must identify a person whose name is the same as the name of another

person's son's name. This is accomplished through the use of variables in the LHS. The first slot that a variable matches against sets the value of the variable. The first time a value is set in a variable on the LHS of a rule, it will not change and the value in the variable will be inserted into subsequent conditional elements. In our example shown in Figure 5.10, the first line in the LHS sets the value of the variable ?son-name to the value stored in the "son" slot of the fact that matches that first conditional element. Having a ?son-name set in the first conditional element allows the rule to then match a second Person fact whose "son" slot contains the value stored in the ?son-name variable just set by the previous conditional element. The "Person" fact that matches the second conditional element is determined to be the father of the "Person" fact matched by the first conditional element. In our example, if we match the "Person" fact with the value of "Joe" in the "first-name" slot with the first conditional element and storing the value "Joe" in the ?son-name variable, then the "Person" fact with the value of "Kenneth" in the "first-name" slot will match the second conditional element because the "Kenneth" fact has the value "Joe" (which is stored in the variable ?son-name) in the "son" slot.



```
(defrule MAIN::find-father
  (Person (first-name ?son-name))
  (Person (first-name ?father-name) (son ?son-name))
  =>
  (printout t "The father of " ?son-name " is " ?father-name crlf))
```

The diagram shows the rule code with two arrows pointing to the variable ?son-name. One arrow points to the first occurrence in the first conditional element, and the other points to the second occurrence in the second conditional element. The text "Matching variable" is placed above the arrows.

Figure 5.10: find-father Rule

The second rule we will define will seek to find the grandson of a person if a person's son's son exists in the basic family tree. The "find-grandson" rule is listed in

Figure 5.11 and has one three conditional elements in the LHS and one action in the RHS. To find the grandson of a person, the LHS must identify a person whose son has a son. This is accomplished through the use of variables in the LHS. In this rule, the first conditional element binds the ?son variable to the value in the “son” slot of a “Person” fact. On the second line, the ?son variable matches to the value stored in the “first-name” slot of a second “Person” fact and a new variable, ?grandson, matches to the value in the second “Person” fact’s “son” slot. At this point, the ?grandson variable contains the name of the first “Person” fact’s grandson. The third conditional element matches the value stored in the ?grandson variable with the “first-name” slot of the third “Person” fact which is the grandson of the first “Person” fact. In our basic family tree example, Billy is identified as the grandson of Kenneth. With all three conditional elements satisfied by three facts in the fact list, the “find-grandson” rule activates and is place on the agenda. Once the agenda is run, CLIPS prints that “Billy is the grandson of Kenneth”.

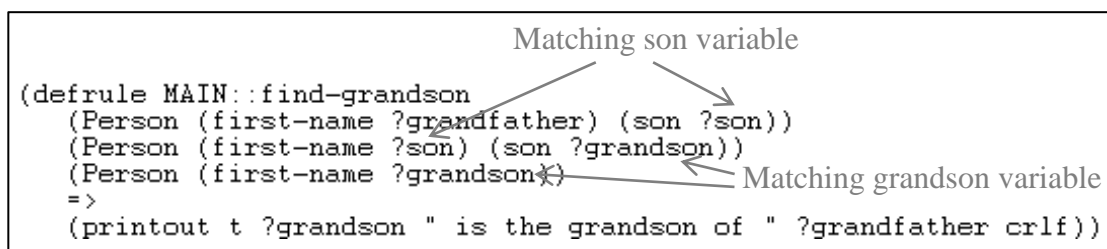


Figure 5.11: find-grandson Rule

Agenda

In CLIPS, the agenda is the collection of all activated rules. A rule is considered activated when all conditional elements match patterns found in facts in the fact list. The agenda acts as an execution queue for when the expert system is run. When the expert

system is run, the activated rules on the agenda are fired and the RHS of each activated rule is executed.

When multiple rules exist on the agenda (which is often the case), the inference engine (discussed below) determines which rule to fire. In order to determine which rules to fire, the inference engine orders the activated rules on the agenda based on salience. Rules that need to fire sooner are assigned a higher salience and, therefore, are placed at the top of the agenda. Rules that may be dependent on the outcome of other fired rules are assigned lower salience values and are placed lower on the agenda. In addition to the inference engine-assigned salience values, the expert system programmer may assign salience values to defrules. The programmer-assigned salience values are respected by the inference engine and orders the activated rules accordingly on the agenda.

Figure 5.12 displays a portion of an agenda displayed by CLIPS. The agenda displays three pieces of information: the salience, activated rule, and matching facts. The first column lists the salience value of the rule. In Figure 5.12, the first and second lines displays rules with salience values of 100, and as 100 is the highest value on the agenda, the two rules are placed at the top of the agenda and will fire first. The third and fourth row show two activates rules with salience values of 20. As a salience of 20 is lower than 100, the two activates rules are placed below the higher salience rules and will fire after the top two rules fire. Should additional rules with higher salience values activate before any of the rules on the agenda can fire, the newly activated rules with higher salience values will be placed ahead of any activates rules with lower salience values on the agenda. The second column in Figure 5.12 displays the activate rule's name. The

last column (after the colon) lists all facts in the fact list that matched the conditional elements in the LHS of the rule thereby causing the rule to activate.

100	choose-random-color: f-26
100	choose-random-color: f-25
20	assign-color-preferences: f-13, f-34, f-48, f-54, f-35
20	assign-color-preferences: f-10, f-28, f-48, f-54, f-35

Figure 5.12: Portion of an Agenda in CLIPS

Inference Engine

The inference engine determines when a rule on the agenda fires. The inference engine matches the facts in the fact list against the conditional elements on the LHS of each rule and arranges the activated rules on the agenda based on its salience value.

There are two commonly used methods of inferencing used by expert systems: forward chaining and backward chaining. Forward chaining reasons conclusions from stated facts, and backwards chaining reasons from a hypothesis to supporting facts. The CLIPS expert system supports forward chaining.

The inference engine in CLIPS operates on a recognize-act cycle which has four steps: conflict resolution, act, match, and check for halt (Giarratano and Riley 2005). The first step, conflict resolution, selects the activated rule on the agenda with the highest priority for firing. The second step, act, fires the selected rule and executes the commands on the RHS of the rule and then removes the rule from the agenda. The third step matches facts in the fact list against the LHS of rules to identify and new rule activations that need to be added to the agenda. The fourth step, check for halt, stops the

execution of the expert system if a halt command is given, otherwise the inference engine starts back at step one.

Program Design of Cartographic Expert System

This section details the deffacts, deftemplates, and defrules programmed into the CLIPS expert system for the purpose selecting symbology for geographic data sets. The input to the expert system is basic information about a data set such as: theme, geometry, data set name, and map scale. To make the input as simple as possible to the user, the expert system receives three of these inputs from the program designed in Chapter 4 leaving scale the only user input. From this basic input information, the expert system will determine draw order, and symbol design for the data sets and return the results of its design decisions.

To build the cartographic expert system, three sets of information were programmed into CLIPS: deffacts, deftemplates, and defrules. These three sets of information together compose the expert cartographic knowledge and allow the inference engine to solve the problem of partially automating the cartographic design process. The following sub-sections will discuss the deffacts, deftemplates, defrules, and how these components interact to design a basic general reference map.

Cartographic deftemplates

The role of a deftemplate in CLIPS is to provide a template for facts. Eleven deftemplates were created for the cartographic expert system, each providing a template for holding important cartographic knowledge from expert sources and information about the input geospatial data sets. Each deftemplate will be displayed and discussed in the following sub-sections. For a complete list of deftemplates, refer to Appendix E.

mapLayer

The “mapLayer” deftemplate holds information about the input data set that will be rendered as a layer in the map. The “mapLayer” deftemplate is displayed in Figure 5.13. There are fourteen slots in the “mapLayer” deftemplate. The first three slots, name, theme, and geometry are the only required slots that must be populated by the user and serve as the only input into the expert system. The remaining eleven slots hold the draw order and symbolization choices.

The name slot holds the name of the input map layer and is a required input. The name of the input map layer serves as a unique identifier for when the results of the expert system are returned to the user. The theme slot stores the designated general reference map theme of the map layer and is a required input. Ten themes are supported by the expert system (not all are shown in Figure 5.13) and they are: airports, contours, roads, emergency services, trails, railroads, state and county boundaries, water features, parks, and hospitals. The designated theme is what the expert system uses to determine appropriate symbol choices and draw order. The geometry slot holds the type of geometry the data set is stored in and is a required input. The expert system only accepts geometry types of point, line, and polygon. The fourth slot, drawOrder, holds the draw order information for the map layer. The draw order is expressed by an integer and ranges from 0 (draw on top) to any positive integer (draw on bottom). All map layers have a default draw order of 0.

Slots five through fourteen hold the symbology information for the map layer as determined by the expert system. The slots hue-value, saturation-value, and value-value hold the symbol’s fill color as expressed using the HSV color model. The slot symbol-

size contains the size of the symbol expressed in points. The symbol slot contains the name of the symbol that should be selected for map layers of geometry type point. The line-width slot contains the width of an outline if the geometry is a point or polygon, and the width of the line symbol if the geometry is line. The line-type slot holds the line pattern, such as dotted or solid. The last three slots, outline-hue-value, outline-saturation-value, and outline-value-value hold the line color expressed using the HSV color model.

```
(deftemplate MAIN::mapLayer
  (slot name (default ?NONE))
  (slot theme (default ?NONE) (allowed-symbols Airports
    emergency Contours Roads Trails Railroads StateCounty
    Water Parks Hospitals))
  (slot geometry (default ?NONE)
    (allowed-symbols point line polygon))
  (slot drawOrder (default 0))
  (slot hue-value (default none))
  (slot saturation-value (default none))
  (slot value-value (default none))
  (slot symbol-size)
  (slot symbol (default none))
  (slot line-width)
  (slot line-type)
  (slot outline-hue-value)
  (slot outline-saturation-value)
  (slot outline-value-value))
```

Figure 5.13: mapLayer deftemplate

theme-ordering

The “theme-ordering” deftemplate holds information about the draw order preferences based on data set themes. The “theme-ordering” deftemplate is displayed in Figure 5.14 and has two slots: theme and above. The theme slot holds the identifier of the theme that the ordering information will be applicable to. The above slot contains a list of themes of which its theme should be drawn on top of.


```
(deftemplate MAIN::theme-ordering
  (slot theme (default ?NONE))
  (multislot above))
```

Figure 5.14: theme-ordering deftemplate

theme-hsv-color-preferences

The “theme-hsv-color-preferences” deftemplate holds the preferred color information for each map theme and is shown in Figure 5.15. The purpose of this deftemplate is to hold the preferred color information for each theme’s symbols. There are five slots in this deftemplate: geometry, theme, hue, saturation, and value. The geometry slot sets the geometry type that the color should be set to. Geometry is designated in the case that a cartographer would have a different color preference based on geometry type. For example, if the location of an airport was represented by a point symbol, the cartographer would be safe in assigning the color black. However, if the airport was represented using a polygon, the cartographer would choose a weaker color such as a grey. The second slot, theme, stores the theme that the color preferences are applied to. The hue slot stores the name of a color and will reference values stored in facts that reference the “hue-range” deftemplate. The saturation slot stores the name of a defined saturation range stored in facts that reference the “saturation-range” deftemplate. The value slot stores the name of a defined value range stored in facts that reference the “value-range” deftemplate.

```
(deftemplate MAIN::theme-hsv-color-preferences
  (slot theme (default ?NONE) (allowed-symbols Airports
    emergency Contours Roads Trails Railroads StateCounty
    Water Parks Hospitals))
  (slot geometry (default ?NONE)
    (allowed-symbols point line polygon))
  (multislot hue (default ?NONE))
  (multislot saturation (default ?NONE))
  (multislot value (default ?NONE)))
```

Figure 5.15: theme-hsv-color-preferences deftemplate

hue-range

The “hue-range” deftemplate stores an integer range of values that represent a named hue. Figure 5.16 displays the “hue-range” deftemplate. A range of values is given for a hue to provide some variability in hues between runs of the expert system. The “hue-range” deftemplate has three slots, name, which holds the name of the hue, hue-low, which holds the low range value for the hue, and hue-high, which holds the high range value for the hue. The hue range values are numbers between 0 and 359 and reference the hue value in the HSV color model.

```
(deftemplate MAIN::hue-range
  (slot name (default ?NONE))
  (slot hue-low (default ?NONE) (type INTEGER)
    (range 0 360))
  (slot hue-high (default ?NONE) (type INTEGER)
    (range 0 360)))
```

Figure 5.16: hue-range deftemplate

saturation-range

The “saturation-range” deftemplate stores an integer range of values that represent a named saturation range. Figure 5.17 displays the “saturation-range” deftemplate. A

range of values is given for a saturation to provide some variability in saturations between runs of the expert system. The “saturation-range” deftemplate has three slots, name, which holds the name of the saturation, saturation-low, which holds the low range value for the saturation, and saturation-high, which holds the high range value for the saturation. The saturation range values are numbers between 0 and 100 and reference the saturation value in the HSV color model.

```
(deftemplate MAIN::saturation-range
  (slot name (default ?NONE))
  (slot saturation-low (default ?NONE) (type INTEGER)
    (range 0 100))
  (slot saturation-high (default ?NONE) (type INTEGER)
    (range 0 100)))
```

Figure 5.17: saturation-range deftemplate

value-range

The “value-range” deftemplate stores an integer range of values that represent a named value range. Figure 5.18 displays the “value-range” deftemplate. A range of values is given for a value to provide some variability in values between runs of the expert system. The “value-range” deftemplate has three slots, name, which holds the name of the value, value-low, which holds the low range value for the value, and value-high, which holds the high range value for the value. The value range values are numbers between 0 and 100 and reference the value in the HSV color model.

```
(deftemplate MAIN::value-range
  (slot name (default ?NONE))
  (slot value-low (default ?NONE) (type INTEGER) (range 0 100))
  (slot value-high (default ?NONE) (type INTEGER) (range 0 100)))
```

Figure 5.18: value-range deftemplate

theme-symbol-preferences

The “theme-symbol-preferences” deftemplate specifies the symbol and associate attributes for each theme and geometry type at a set map scale. As shown in Figure 5.19, the “theme-symbol-preferences” deftemplate has eight slots. The theme slot holds the theme for which the symbol will be applied. The geometry slot specifies which type of geometry the symbol will be applied to. The geometry slot is necessary as a theme may be represented with different geometry based on the scale of the data set. The scale slot holds a named slot that references the fact set in scale-level to determine whether the symbol should be applied at the current map scale. The symbol slot holds the name of the symbol that should be applied to the theme. The line-width slot specifies the width of the line symbol, or width of the outline on a point or polygon symbol. The line-type slot specifies the type of the line, such as dotted or solid. Lastly, the relative-outline-color slot sets the color of the symbol outline (if applicable). The outline color is set as a relative value to the symbol’s fill and the value stored in the relative-outline-color references a fact related to the relative-outline-color deftemplate.

```
(deftemplate MAIN::theme-symbol-preferences
  (slot theme (default ?NONE) (allowed-symbols Airports
    emergency Contours Roads Trails Railroads StateCounty
    Water Parks Hospitals))
  (slot geometry (default ?NONE)
    (allowed-symbols point line polygon))
  (slot scale (default ?NONE))
  (slot symbol)
  (slot symbol-size)
  (slot line-width)
  (slot line-type)
  (slot relative-outline-color))
```

Figure 5.19: theme-symbol-preferences deftemplate

relative-outline-color

The “relative-outline-color” deftemplate stores the information required to set a relative outline color based on a symbol’s fill color. Figure 5.20 displays the “relative-outline-color” deftemplate. This deftemplate has two slots: relative-color-choice and percent-multiplier. The relative-color-choice stores a unique name for the relative value. The percent-multiplier stores a float that is used to multiply a value of a symbol’s fill value to determine the outline’s value.

```
(deftemplate MAIN::relative-outline-color
  (slot relative-color-choice (default ?NONE))
  (slot percent-multiplier (type FLOAT)))
```

Figure 5.20: relative-outline-color deftemplate

scale-level

The “scale-level” deftemplate displayed in Figure 5.21 holds maximum and minimum map scale levels related to a named scale. This deftemplate allows for scale values to be assigned a named scale, such as “large” or “medium” as well as associated

limits on the scale. Symbol preferences will relate to the scales stored in facts that relate to this deftemplate for purposes of determining which symbols to assign. This deftemplate has three slots: name, min, and max. The name slot stores the assigned name to the scale range. The min and max slots store the minimum map scale and maximum map scale respectively.

```
(deftemplate MAIN::scale-level
  (slot name)
  (slot min (type INTEGER))
  (slot max (type INTEGER)))
```

Figure 5.21 scale-level deftemplate

current-scale

The “current-scale” deftemplate holds an integer value that represents the current map scale in the scale slot. A fact must exist for this deftemplate otherwise the expert system will not be able to determine which symbols should be assigned to map layers. This deftemplate is displayed in Figure 5.22.

```
(deftemplate MAIN::current-scale
  (slot scale (type INTEGER)))
```

Figure 5.22: current-scale deftemplate

input-scale

The “input-scale” deftemplate displayed in Figure 5.23 holds an integer value that represents a scale input by the user in the scale slot. The fact related to this deftemplate is used to change the current map scale value to this new input scale.

```
(deftemplate MAIN::input-scale
  (slot scale (type INTEGER) (default ?NONE)))
```

Figure 5.23: input-scale deftemplate

Cartographic deffacts

The role of a deffact in CLIPS is to seed the expert system with knowledge. For the cartographic expert system, the deffacts represent expert cartographic knowledge that will be applied to map layers input by the user. The deffacts in the cartographic expert system comprise the largest portion of the expert system. The deftemplates were designed to allow for easy growth in cartographic knowledge by minimizing the amount of hard-coded information in the deftemplates; this allows the expert system to easily include additional deffacts without having to restructure or redesign deftemplates or the expert system.

This section will discuss a representative sample of the deffacts included in the expert system because it is impractical to show all deffacts in this chapter. For a complete list of deffacts, refer to Appendix E.

initial-theme-ordering

This collection of deffacts stores the map layer ordering information based on its theme. These deffacts relate to the “theme-ordering” deftemplate. Every theme listed in

the above slot in a fact represents every theme that should be placed underneath the current map layer. If multiple themes should be placed underneath the current map layer, then each layer is listed in the above slot separated by spaces. Figure 5.24 displays deffacts for four themes.

```
(deffacts MAIN::initial-theme-ordering
  (theme-ordering (theme Parks) (above StateCounty))
  (theme-ordering (theme Roads)
    (above Water Parks StateCounty Trails))
  (theme-ordering (theme Water) (above Boundaries Parks))
  (theme-ordering (theme StateCounty) (above)))
```

Figure 5.24: Sample of initial-theme-ordering deffacts

Referencing Figure 5.24, the third line lists the theme-ordering fact for the Roads theme. In the above slot for this theme, Water, Parks, StateCounty, and Trails are listed as the themes of map layers that should be placed underneath the roads with respect to draw order.

initial-color-preferences

This collection of deffacts stores the color preferences for a given theme of data. These facts relate to the “theme-hsv-color-preference” deftemplate. These facts represent preferred colors for each theme based on basic cartographic color theory. For instance, themes that cover large areas, such as the StateCounty theme are given a cooler, low saturation color to create a nice ground for the map while figures, such as airports, and water are provided with warmer, medium to high saturation colors to create nice figures and a good contrast from the ground. A sample of the theme color preferences is listed in Figure 5.25.


```
(deffacts MAIN::initial-color-preferences
  (theme-hsv-color-preferences (theme Water)
    (geometry polygon) (hue blue) (saturation high)
    (value medium-high))
  (theme-hsv-color-preferences (theme Trails)
    (geometry line) (hue brown) (saturation high)
    (value high))
  (theme-hsv-color-preferences (theme Parks)
    (geometry polygon) (hue green) (saturation high)
    (value medium-high))
```

Figure 5.25: Sample of initial-color-preferences deffacts

hsv-information

The “hsv-information” deffacts define acceptable hue, saturation, and value ranges named instances. The named hues in this list of facts were selected to represent as pure a hue as possible without much bleeding into adjacent hues on the color wheel. Additionally, if two hues are adjacent on the color wheel, a reasonable gap of hue values was maintained so that hues will remain distinct on the map. The named saturations in this list of facts were selected to provide a small range of saturations for a slightly different looks of the map between runs, and enough of a gap between named saturations to provide sufficient contrast between named saturations. The named values, much like the saturations, provided a small range for value selection and large enough gaps for users to differentiate between values. A sample of facts asserted in this deffacts structure is listed in Figure 5.26.

```
(deffacts MAIN::hsv-information
  (hue-range (name black) (hue-low 0) (hue-high 0))
  (hue-range (name red) (hue-low 0) (hue-high 15))
  (saturation-range (name medium-low) (saturation-low 30)
    (saturation-high 40))
  (saturation-range (name low) (saturation-low 10)
    (saturation-high 20))
  (value-range (name medium-high) (value-low 70)
    (value-high 80))
  (value-range (name medium) (value-low 50)
    (value-high 60))
```

Figure 5.26: Sample of hsv-information deffacts

scales

The “scales” deffacts lists all of the map scales that the expert system recognizes for cartographic purposes. Facts in this deffacts construct relate to the “scale-level” deftemplate. Each scale represents a scale in which a symbol or color choice will change. Each scale-level should have a related symbology fact that can be applied for the scale level for a theme. A sample of the “scales” deffacts is listed in Figure 5.27.

```
(deffacts MAIN::scales
  (scale-level (name large) (min 1000) (max 30000))
  (scale-level (name medium) (min 30001) (max 300000))
  (scale-level (name small) (min 300001) (max 30000000)))
```

Figure 5.27: Sample of scales deffacts

initial-relative-outline-colors

The facts listed in the “initial-relative-outline-colors” deffacts construct define how dark or light an outline of a symbol should be in relation to the symbol’s fill color. The multipliers chosen allow for a significant enough contrast between a fill and outline color to be seen by the map reader. The deffacts are listed in Figure 5.28.

```
(deffacts MAIN::initial-relative-outline-colors
  (relative-outline-color (relative-color-choice darker)
    (percent-multiplier 0.6))
  (relative-outline-color (relative-color-choice same)
    (percent-multiplier 1.0))
  (relative-outline-color (relative-color-choice lighter)
    (percent-multiplier 1.4)))
```

Figure 5.28: initial-relative-outline-colors deffacts

initial-theme-symbol-preferences

The “initial-theme-symbol-preferences” deffacts list the symbology choices for the general reference map themes targeted in this research. The symbols outlined in these facts were chosen to provide safe, commonly used symbols found on general reference maps and would have the widest audient appeal. For each named map scale, a symbol preference should exist so the expert system can determine which symbol to choose for each map scale level. A sample of facts from this defftemplate is displayed in Figure 5.29.

```
(deffacts MAIN::initial-theme-symbol-preferences
  (theme-symbol-preferences (theme Hospitals) (geometry point)
    (scale large) (symbol hospital) (symbol-size 16)
    (line-width 0) (line-type none)
    (relative-outline-color same))
  (theme-symbol-preferences (theme Airports) (geometry point)
    scale large) (symbol airport) (symbol-size 16)
    (line-width 0) (line-type none)
    (relative-outline-color same))
```

Figure 5.29: Sample of initial-theme-symbol-preferences deffacts

Cartographic defrules

The role of a defrule in CLIPS is to execute some action when facts in the fact list exist that meet the conditional elements of the rule. For the cartographic expert system, the seven defrules represent the actions that are to be taken by the expert system in creating the basic general reference map. This section will discuss the seven defrules and how they work. For each defrule code listing, line numbers have been added to assist in referencing specific lines in the discussion. For a complete list of defrules, refer to Appendix E.

choose-random-color

The “choose-random-color” defrule serves to initialize the expert system by choosing a color from a random list of acceptable colors for facts that relate to the “theme-hsv-color-preferences” deftemplate. Figure 5.30 displays the defrule. This rule has the highest salience value (100) of any rule in the cartographic expert system as the colors used for themes must be set first. The salience value is set on line 2 of the defrule. Line 3 is a conditional element that identifies facts related to the “theme-hsv-color-preferences” deftemplate that has at least one hue specified. With facts matched on line 3, the RHS (lines 5-9) randomly chooses one of the defined hues. Line 5 finds the number of hues specified. Line 6 checks to see if more than 1 hue exists. If more than one hue exists, then line 8 randomly chooses one of the hues and stores it in the ?chosenhue variable. Line 9 modifies the fact identified in line 3 and saves the fact with the randomly chosen hue only. The end effect is that the color preference for each theme now only has one hue that will be used for the remainder of the expert system’s execution.

```

1 (defrule MAIN::choose-random-color
2   (declare (salience 100))
3   ?col-pref <- (theme-hsv-color-preferences (hue $?hues))
4   =>
5   (bind ?len (length$ ?hues))
6   (if (neq ?len 1)
7       then
8       (bind ?chosenhue (nth$ (random 1 ?len) ?hues))
9       (modify ?col-pref (hue ?chosenhue))))

```

Figure 5.30: choose-random-color defrule

fix-duplicate-drawOrders

The “fix-duplicate-drawOrders” defrule provides a unique draw order number for each map layer if two or more layers have the same draw order. As each layer in a map must be placed above or below another map layer, no two map layers can share the same drawing order. This defrule prevents duplicate draw orders. Figure 5.31 lists this defrule. This defrule has the second highest salience value (30) in the cartographic expert system as duplicate draw orders should be fixed as soon as they are identified so other defrules can operate properly. To find map layers with duplicate draw orders, a two-part conditional element is used. The first part of the conditional element is on line 3 and matches against a map layer with any name and a set draw order. The name of the map layer is stored in the ?name variable, and the draw order is stored in the ?loc variable. Line 4 is the second part of the conditional element and matches against a map layer that does not have the same name that is stored in the ?name variable; this prevents the defrule from matching a fact against itself. The draw order stored in ?loc is matched against the draw order in the second fact. So, in essence, if the second fact does not have the same name as the first fact but does have the same draw order, then the conditional element is satisfied and the defrule is activated. When the rule fires, line 6 modifies the

first map layer (identified on line 3) to increment its draw order by 1. The two map layers no longer have the same draw order.

```
1 (defrule MAIN::fix-duplicate-drawOrders
2   (declare (salience 30))
3   ?mapLayer1 <- (mapLayer (name ?name) (drawOrder ?loc))
4   (mapLayer (name ~?name) (drawOrder ?loc))
5   =>
6   (modify ?mapLayer1 (drawOrder (+ ?loc 1))))
```

Figure 5.31: fix-duplicate-drawOrders defrule

order-mapLayers

The “order-mapLayers” defrule arranges the map layers’ draw order based on the theme ordering preferences stored in facts that relate to the “theme-ordering” deftemplate. Figure 5.32 lists the defrule.

```
1 (defrule MAIN::order-mapLayers
2   ?mapLayer1 <- (mapLayer (theme ?theme) (drawOrder ?loc1))
3   (theme-ordering (theme ?theme) (above $? ?abovetheme $?))
4   ?mapLayer2 <- (mapLayer (theme ?abovetheme) (drawOrder ?loc2))
5   (test (> ?loc1 ?loc2))
6   =>
7   (modify ?mapLayer1 (drawOrder ?loc2))
8   (modify ?mapLayer2 (drawOrder ?loc1)))
```

Figure 5.32: order-mapLayers defrule

The “order-mapLayers” defrule has a four-part conditional element shown in lines 2 thru 5 in Figure 5.32. Line 2 matches a map layer storing its theme and drawOrder in the ?theme and ?loc1 variables respectively. Line 3 looks for a fact that specifies which themes should be draw below the current theme. It performs this match by matching the

value stored in ?theme against facts that reference the “theme-ordering” deftemplate.

Once a match has been found for theme ordering, one of the themes listed as being below the current theme is stored in the ?abovetheme variable. Line 4 uses the value stored in ?abovetheme to identify any map layer that has the theme that should be draw below the map layer identified in line 2. If a map layer is found, its location is stored in ?loc2. Line 5 tests to see if the draw order of the first map layer is greater than the draw order of the second map layer. If the second map layer’s draw order is lower, then the rule activates and lines 7 and 8 switch the draw order numbers between the two map layers.

set-scale

The “set-scale” defrule sets the current map scale equal to a newly inputted scale. In order for the inputted map scale to be successfully set, it must fall within the range of a named map scale. Figure 5.33 lists the “set-scale” defrule. Lines 2 thru 5 list the four-part conditional element for this defrule to activate. Line 2 matches the current scale fact and line 3 matches the newly inputted scale fact and stores the scale number in the ?input-scale-value variable. Line 4 matches a named scale fact that references the “scale-level” deftemplate. Line 5 tests to see if the value stored in ?input-scale-value is between the minimum and maximum scales for the named scale. If the test passes, then the rule is activated and the RHS will execute. When the RHS executes, line 7 retracts the newly inputted scale fact, and line 8 updates the current scale fact with the name of the scale that matched on the LHS of the defrule.

```

1 (defrule MAIN::set-scale
2   ?cur-scale <- (current-scale)
3   ?in-scale <- (input-scale (scale ?input-scale-value))
4   (scale-level (name ?name) (min ?min) (max ?max))
5   (test (and (>= ?input-scale-value ?min)
6              (<= ?input-scale-value ?max)))
7   =>
8   (retract ?in-scale)
9   (modify ?cur-scale (scale ?name)))

```

Figure 5.33: set-scale defrule

assign-color-preferences

The “assign-color-preferences” defrule applies colors to map layers based on the preferred colors set in facts that relate to the “theme-hsv-color-preferences” deftemplate. This defrule is listed in Figure 5.34. This defrule has a five-part conditional element. The first part shown on line 2 matches a map layer that does not have a hue, saturation, or value set. The second part shown on line 3 matches a fact that references the “theme-hsv-color-preferences” deftemplate and has the same theme and geometry as the map layer identified on line 2. The named hue, saturation, and value that are applicable to the theme and geometry are saved in variables. The third, fourth, and fifth parts of the conditional element shown on lines 4, 5, and 6 match the named hue, value, and saturation against facts that relate to the “saturation-range”, “value-range”, and “hue-range” deftemplates. The maximum and minimum values for each part of the HSV color model are stored in variables. With the five conditional elements satisfied, the defrule will activate and the RHS will execute. On the RHS of the defrule, line 8 handles the special case of the color white. A white color must have a saturation of 0 and a value of 100. Line 9 modifies the map layer identified in line 2 and sets the hue, saturation, and

value of the map layer with randomly chosen values for each within the ranges set for the named hue, saturation, and values.

```
1 (defrule MAIN::assign-color-preferences
2   ?layer <- (mapLayer (hue-value none) (saturation-value none)
3     (value-value none) (theme ?theme) (geometry ?geometry))
4   (theme-hsv-color-preferences (theme ?theme)
5     (geometry ?geometry) (hue ?hue) (saturation ?saturation)
6     (value ?value $?))
7   (saturation-range (name ?saturation) (saturation-low ?sat-low)
8     (saturation-high ?sat-high))
9   (value-range (name ?value) (value-low ?val-low)
10    (value-high ?val-high))
11  (hue-range (name ?hue) (hue-low ?hue-low)
12    (hue-high ?hue-high))
13  =>
14  (if (eq ?hue white) then (bind ?sat-low 0)
15    (bind ?sat-high 0) (bind ?val-low 100)
16    (bind ?val-high 100))
17  (modify ?layer (hue-value (random ?hue-low ?hue-high))
18    (saturation-value (random ?sat-low ?sat-high))
19    (value-value (random ?val-low ?val-high))))
```

Figure 5.34: assign-color-preferences defrule

match-theme-colors

The “match-theme-colors” defrule shown in Figure 5.35 sets all map layers of the same theme to the same hue, saturation, and value so the theme has a unified color on the map. This defrule has a two-part conditional element. The first part of the conditional element on line 2 identifies a map layer that has a hue, value, and saturation assigned. The second part of the conditional element shown on line 3 finds a map layer that has the same theme and geometry as the first map layer and does not already have the same hue, saturation, and value as the first map layer. With the conditional elements matched, the rule will activate and set the hue, saturation, and value of the second map layer equal to the hue, saturation, and value of the first map layer as shown on line 5.

```

1 (defrule MAIN::match-theme-colors
2   (mapLayer (theme ?theme) (geometry ?geometry)
3     (hue-value ?hue-value&~none)
4     (saturation-value ?sat-value&~none)
5     (value-value ?val-value&~none))
6   ?layer2 <- (mapLayer (theme ?theme) (geometry ?geometry)
7     (hue-value ~?hue-value)
8     (saturation-value ~?sat-value)
9     (value-value ~?val-value))
10  =>
11  (modify ?layer2 (hue-value ?hue-value)
12    (saturation-value ?sat-value)
13    (value-value ?val-value)))

```

Figure 5.35: match-theme-colors defrule

assign-symbolology

The “assign-symbolology” defrule assigns symbology appropriate to the map scale to map layers. Figure 5.36 displays this defrule. This defrule has a four-part conditional element for the rule to activate. The first part of the conditional element on line 2 checks to make sure that a map scale has been set. As the symbol choices are dependent on map scale, if no map scale is set, no symbology will be assigned. Line 3 is the second part of the conditional element matches a map layer that does not have a symbol assigned. The third part of the conditional element is on line 4. This line matches a fact that relates to the “theme-symbol-preferences” deftemplate with the same geometry and theme as the map layer matched on line 3. Additionally, the symbol attributes are stored in variables so they can be assigned to the map layer in the RHS of the defrule. The fourth part of the conditional element shown on line 5 matches the named relative outline color stored from line 4 with a fact that relates to the “relative-outline-color” deftemplate. The outline color multiplier is stored in a variable for used on the RHS. Once all four parts of the

conditional element are satisfied, the rule activates. Line 7 assigns the symbol attributes to the map layer identified on line 3.

```
1 (defrule MAIN::assign-symbology
2   (current-scale (scale ?current-scale))
3   ?layer <- (mapLayer (theme ?theme) (geometry ?geometry)
4     (symbol none) (hue-value ?hue-value)
5     (saturation-value ?saturation-value)
6     (value-value ?value-value))
7   (theme-symbol-preferences (scale ?current-scale)
8     (geometry ?geometry) (theme ?theme) (symbol ?symbol)
9     (symbol-size ?symbol-size) (line-width ?line-width)
10    (line-type ?line-type)
11    (relative-outline-color ?relative-outline-color))
12  (relative-outline-color
13    (relative-color-choice ?relative-outline-color)
14    (percent-multiplier ?percent-multiplier))
15  =>
16  (modify ?layer (symbol ?symbol) (symbol-size ?symbol-size)
17    (line-width ?line-width) (line-type ?line-type)
18    (outline-hue-value ?hue-value)
19    (outline-value-value
20      (* ?value-value ?percent-multiplier))
21    (outline-saturation-value ?saturation-value)))
```

Figure 5.36: assign-symbology defrule

Results and Discussion

This section displays and discusses six map outputs from the cartographic expert system; three large scale, and three small scale maps to show that the expert system can symbolize differently at difference scales. The assumption is made that the map user has selected data appropriate for mapping at the chosen scale. For these six maps, at least one map layer of each theme will be included. A few themes will have more than one map layer. Having multiple map layers of the same theme will demonstrate the expert system's capability to homogenize colors across map layers of the same theme. In order to demonstrate the expert system's capacity to homogenize colors across map layers, we

will use multiple map layers containing the same theme. In addition to the six maps generated by the cartographic expert system, two map outputs will be discussed that used the randomly assigned colors provided by the mapping software to provide a before and after view of the maps.

Large Scale Maps

The first set of three maps was created at a map scale of 1:30,000 over a portion of Tarrant County, Texas. Ten map layers were placed on the map to be symbolized by the expert system. Table 5.1 lists the alias name of the data set, the theme of the data set, and the geometry of the data set used in the three large scale maps.

Table 5.1: Data Sets used in Large Scale Map

Alias Name	Theme	Geometry
Hospitals	Hospitals	point
Airports	Airports	point
Railroads	Railroads	line
Water Body	Water	polygon
Rivers	Water	line
Parks	Parks	polygon
Emergency Services	Emergency	point
Major Roads	Roads	line
Roads	Roads	line
Counties	StateCounty	polygon

Before running the expert system, a map was saved in order to visualize how the data was symbolized in the mapping software before the expert system was run. This map is shown in Figure 5.37. This map is an example of what is often presented to the user of the map software when adding data sets. The map layer order is not optimal, as the Counties map layer is obscuring the view of the Major Roads and Roads map layers. Additionally, the Railroads are symbolized as a thin red line as are the rivers. The Parks

and Water Bodies map layers are very close in color and do not differentiate enough from each other to be useful to the map reader. The map maker will need to reorder and symbolize each layer in full to arrive at a pleasing map.

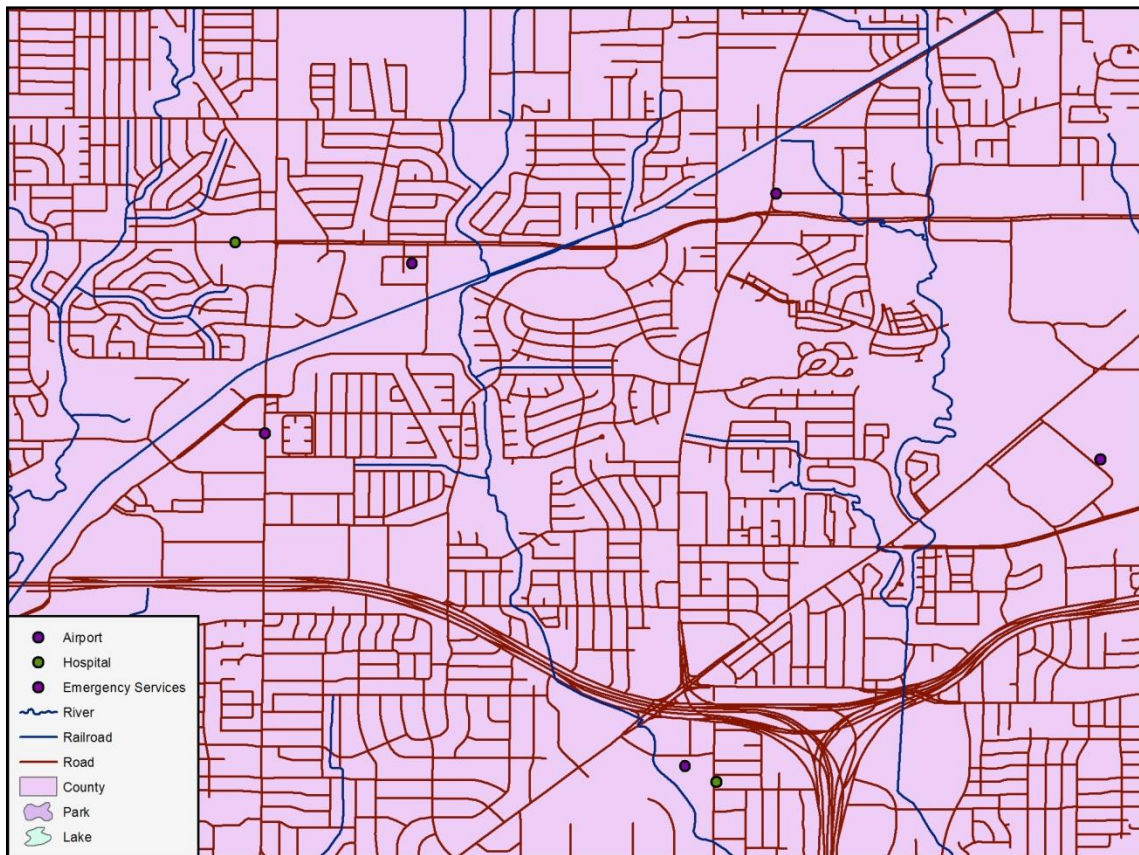


Figure 5.37: Large Scale Map with Default Colors

With the map layers added to the mapping software, the expert system was run three times at the same scale with the data sets shown in Table 5.1 in order to depict how each run of the expert system produces different results. The three map outputs are shown below in Figures 5.38, 5.39, and 5.40 and will be discussed next.

The first map, shown in Figure 5.38, provides a reasonably well-designed general reference map based on the inputs. The expert system successfully ordered the map layers based on the preferred map ordering and the result is pleasing. No layer is obscured by another layer. The expert system provided a white fill color for the Counties map layer and creates excellent ground for the figures on the map. The transportation layers (Roads, Major Roads, and Railroads) use the color black and a small line width as it is assumed that at this large map scale, more detailed roads would be displayed, therefore, a thinner line width would be appropriate so to not cause lines to bleed into each other and visually dominate the map. Additionally, both map layers of theme Roads have been provided the same hue, saturation, and value to provide a unified look across the theme. The Parks layer is close to a pure green and has a high and medium-high saturation and value respectively. This allows the Parks layer to be viewed as a figure, but not overly dominating the map with a full valued color. The Rivers and Water Body map layers were assigned the same pure blue color with a high and medium-high saturation and value respectively. The color choice for the Water theme works well with the other layers and is pleasing. The Airports map layer uses an airplane symbol at a reasonable size and black color. A standard Hospital symbol with a standard blue color at a reasonable size was used in the Hospitals map layer. Lastly, the Emergency Services map layer uses a red star as emergencies can be viewed as an advancing, warm red color. Overall, the first map produced by the cartographic expert system is reasonably well symbolized and provides a great starting point for novice and expert cartographers alike.

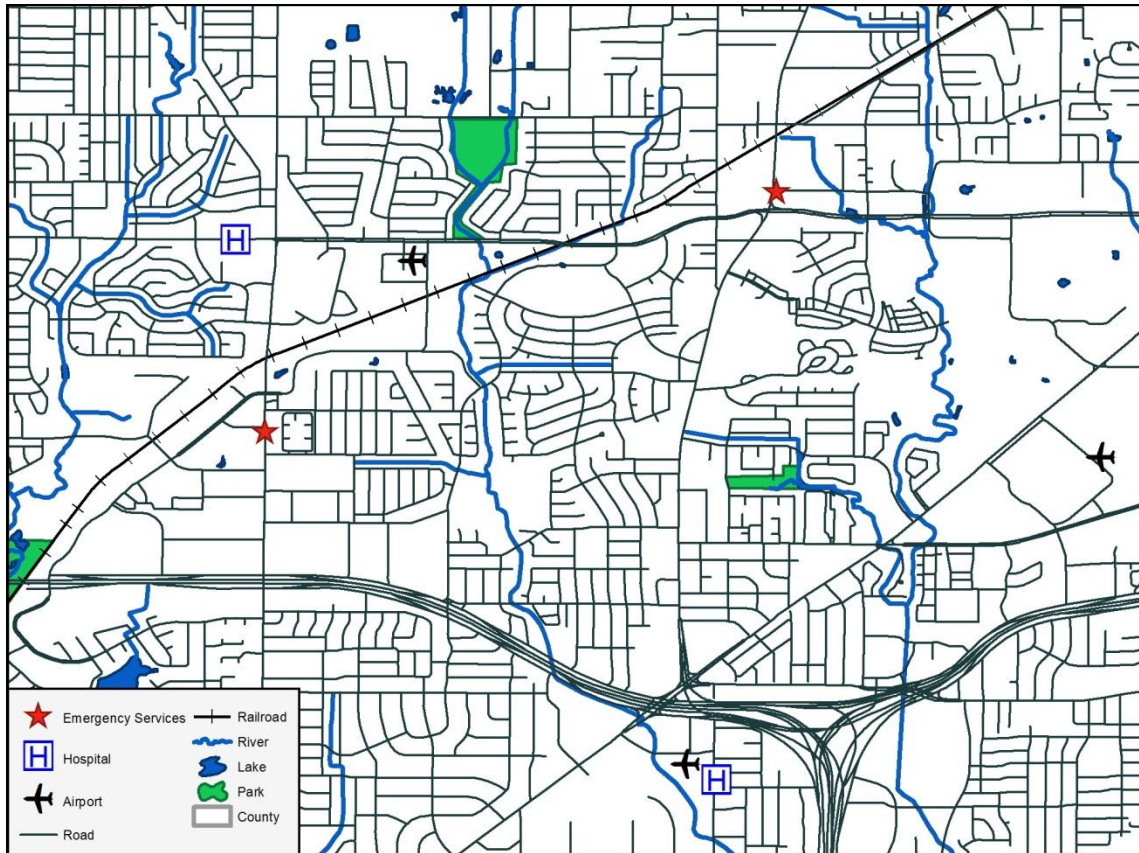


Figure 5.38: First Run of Expert System for Large Scale Data Sets

The second map, shown in Figure 5.39, also provides a reasonably well-designed general reference map based on the inputs. The expert system successfully ordered the map layers based on the preferred map ordering and the result is pleasing. No layer is obscured by another layer. The expert system provided a tan fill color for the Counties map layer and creates reasonable ground for the figures on the map. The transportation layers (Roads, Major Roads, and Railroads) retain similar colors to the first map and colors are copied across different map layers with the same theme. Hospitals, Rivers, Water Bodies, Railroads, and Parks all retain similar colors to the first map and are visually pleasing. The Emergency Services map layer still uses a star for the symbol, but

has a fill color of blue in this map. As blue is a reasonable color to use for emergency and other services, this works well on the map. Overall, the second map produced by the cartographic expert system is reasonably well symbolized.

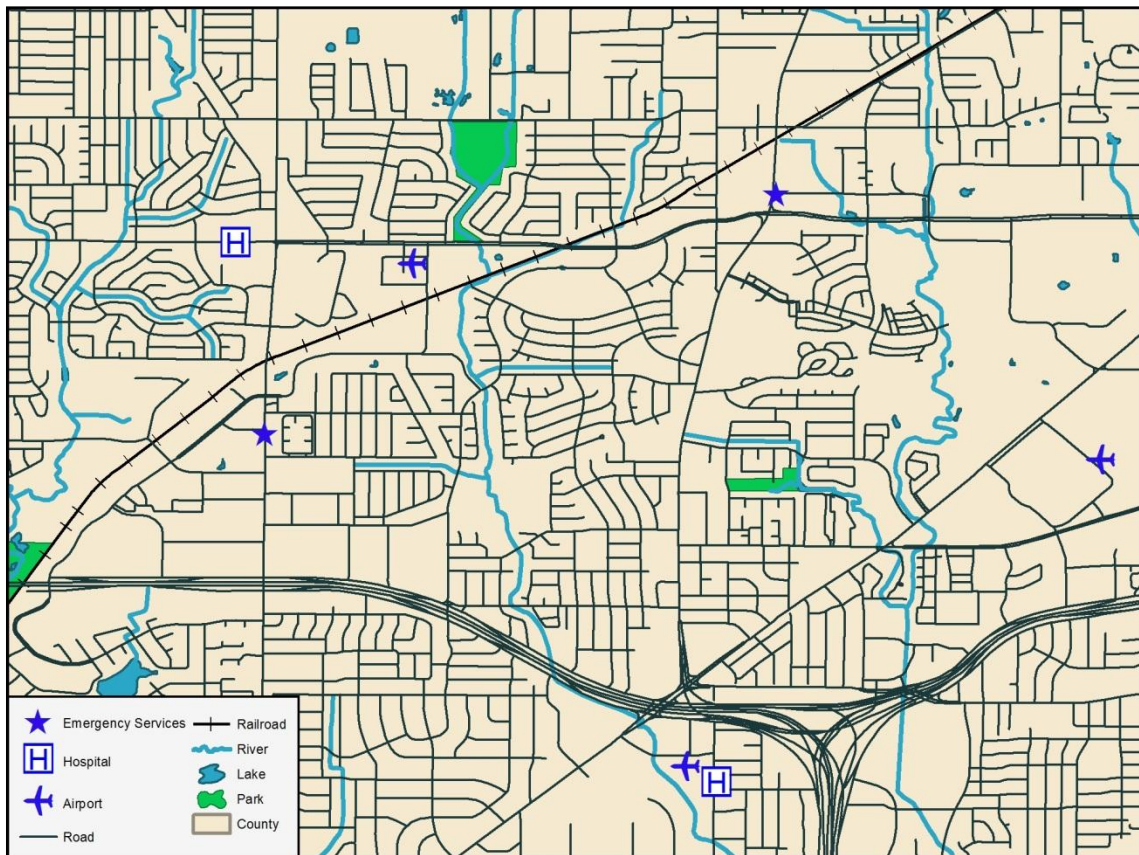


Figure 5.39: Second Run of Expert System for Large Scale Data Sets

The third map, shown in Figure 5.40, again, provides a reasonably well-design general reference map. Once again, the expert system ordered the map layers based on the preferred map ordering and the result is pleasing. The expert system provided a light green fill color for the Counties map layer which provides good contrast with the figure map layers. The Roads and Major Roads map layers use a medium saturation and value

purple color which provides an alternate, slightly off-beat, but still useful color for the roads. Railroads are still using hatched thin lines. The remainder of the map layers, with the exception of the Airports map layer, was symbolized with similar colors to the first map. The Airports map layer still uses the airplane symbol, but with a bright blue color. Like the first two maps, the third map creates a reasonably well-design map and would be an acceptable base map for cartographers to tweak.

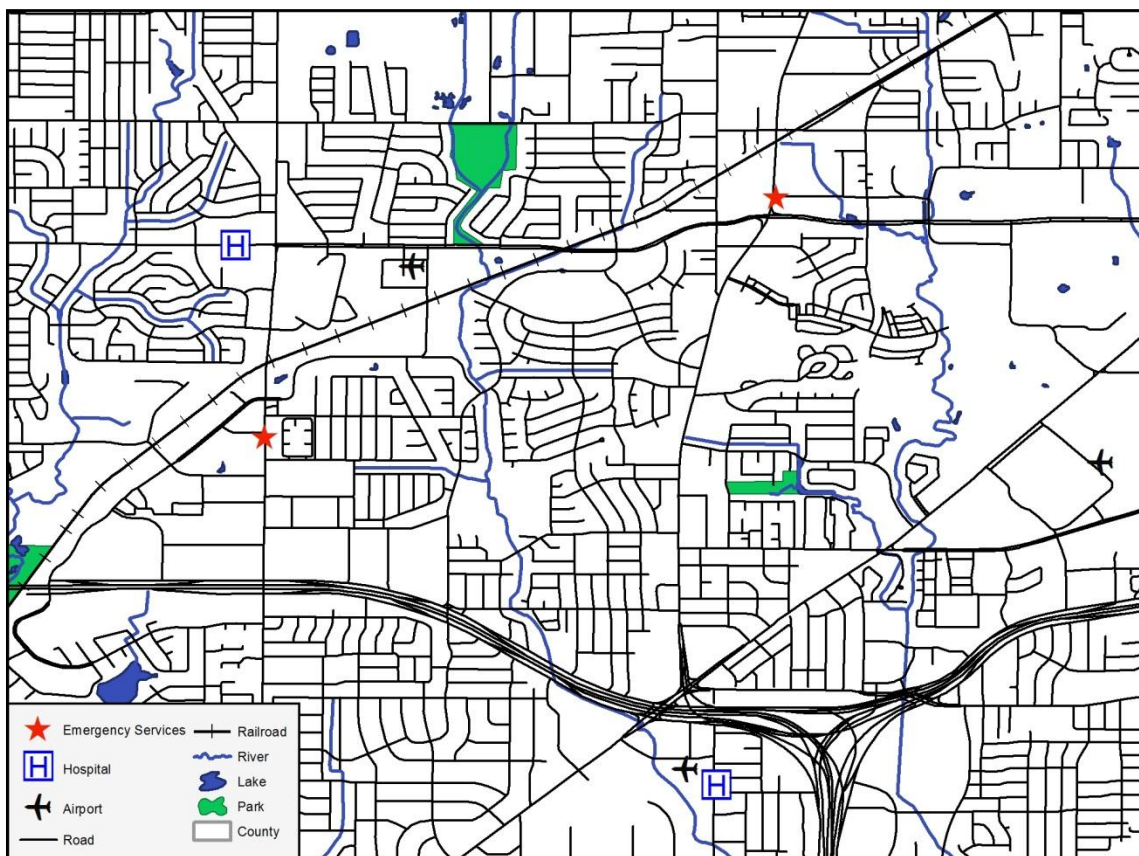


Figure 5.40: Third Run of Expert System for Large Scale Data Sets

These three maps designed by the expert system are more visually pleasing than the default colors chosen by the software when the datasets are added. Figure 5.41

displays the default map and three expert system designed maps for easier reference. Each of the three expert system designed maps follows cartographic expertise by using acceptable colors, commonly used symbols, and proper draw ordering. Each of these designed maps also provides a reasonably well-designed general reference map.

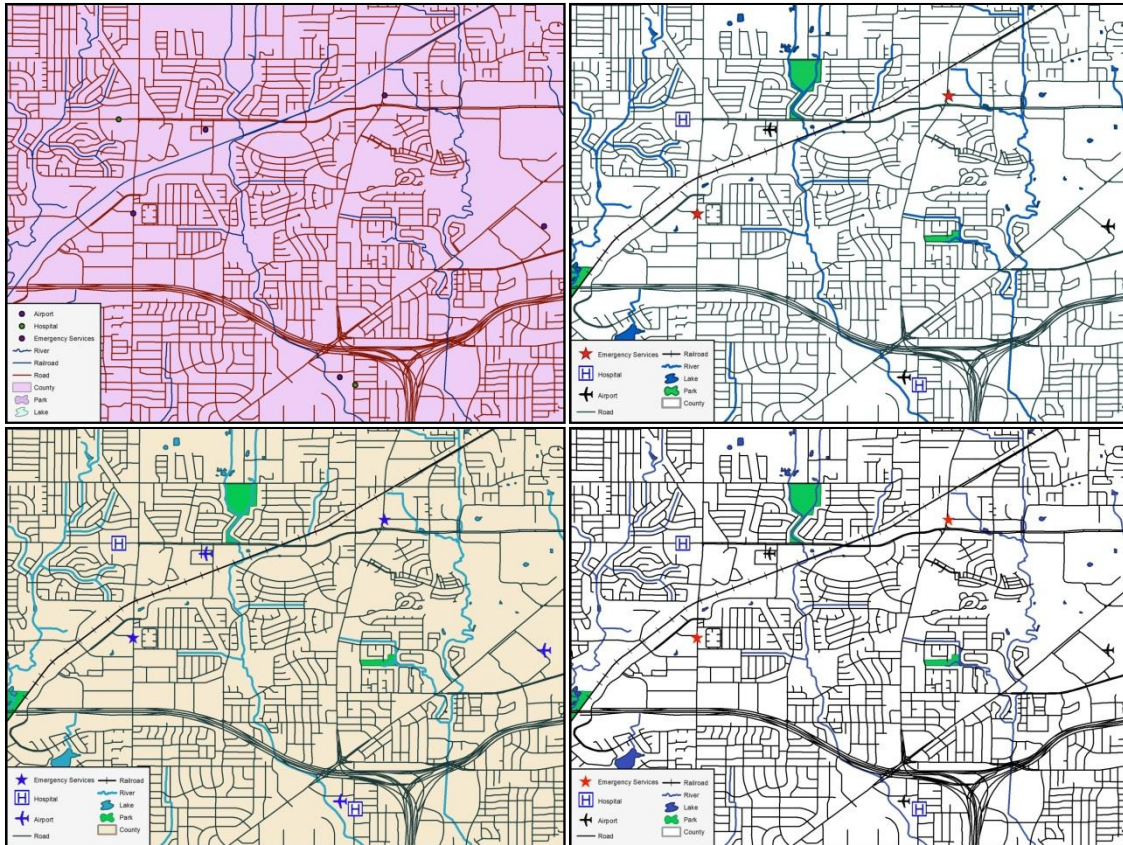


Figure 5.41: Summary of Default and Expert System Designed Maps

Small Scale Maps

The second set of three maps was created at a map scale of 1:7,500,000 showing the entire state of Texas. As this is a large scale map, only five map layers were included to simulate what a reasonably-well populated general reference map would look like.

Table 5.2 lists the alias name of the data set, the theme of the data set, and the geometry of the data set used in the three small scale maps.

Table 5.2: Data Sets used in Small Scale Map

Alias Name	Theme	Geometry
Water Body	Water	polygon
Rivers	Water	line
Parks	Parks	polygon
Major Roads	Roads	line
Counties	StateCounty	polygon

Before the expert system was run on the small scale map, a map was saved to show how the data was symbolized in the mapping software before the expert system was run. This map is shown in Figure 5.42. This map is an example of what is often presented to the user of the map software when adding data sets. The map layer order is not optimal, as the Counties map layer is obscuring the view of the Parks map layer. The Major Roads, Rivers, Water Bodies, and Counties need to be reorganized to provide a better draw order. Additionally, all five map layers are symbolized poorly and with poor color choices. Like the default map created for the large scale map in Figure 5.37, the map maker will need to reorder and symbolize each layer in full to arrive at a pleasing map.

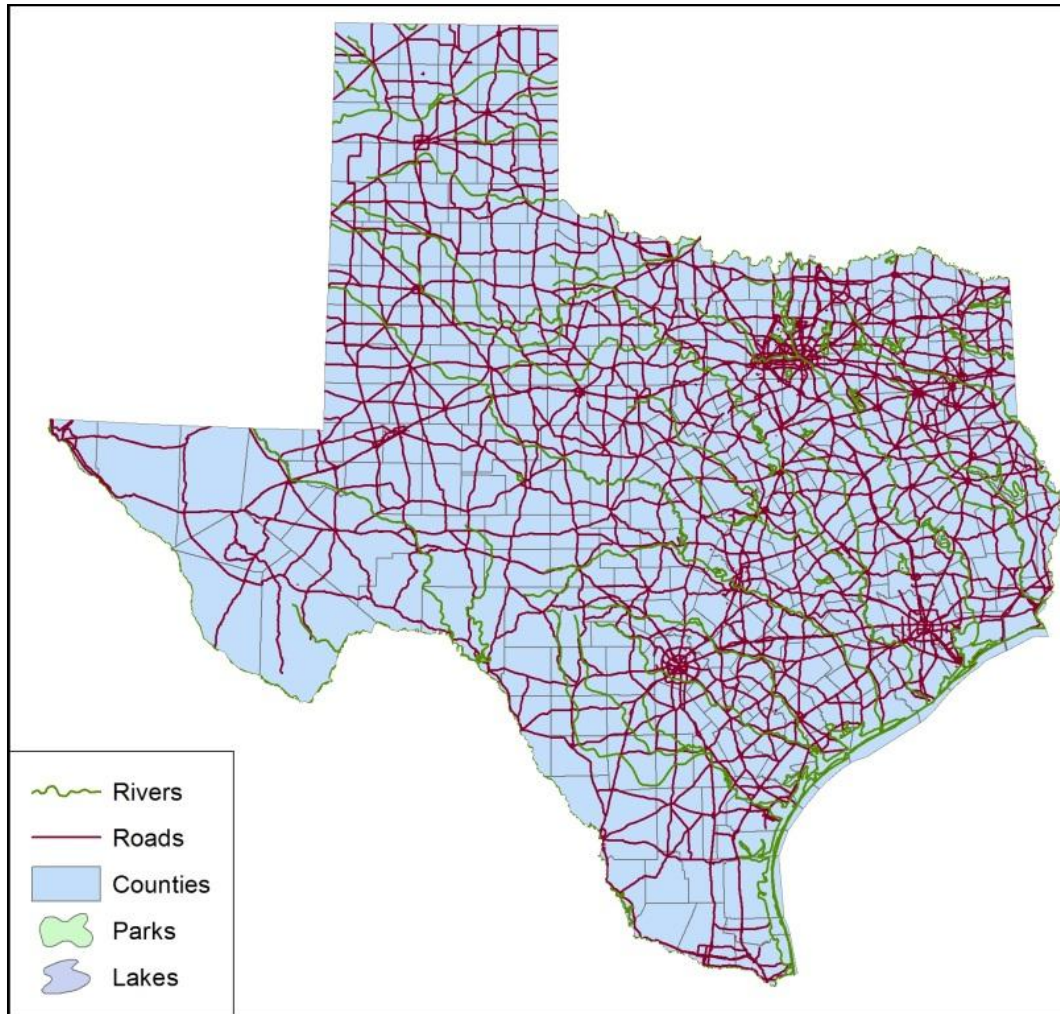


Figure 5.42: Small Scale Map with Default Colors

With the small scale map layers added to the mapping software, the expert system was run three times at the same scale and with the data sets shown in Table 5.2 to show how each run of the expert system produces different results. The three map outputs are shown below in Figures 5.43, 5.44, and 5.45 and will be discussed next.

The first small scale map, shown in Figure 5.43, provides a reasonably well-designed general reference map of Texas. The expert system successfully ordered the map layers based on the preferred map ordering and the result is pleasing. No layer is

obscured by another layer as was the case in the default map. The expert system provided a white fill color for the Counties map layer with light grey outlines with a slightly thick outline; this creates excellent ground for the figures on the map. The Major Roads map layer use the color black and with a 1.25 point line width as it is assume that at this large map scale, less detailed roads would be displayed, therefore, a slightly thick line width would be appropriate so the Roads themed layers would not be lost on the map. The Parks layer is close to a pure green and has a high and medium-high saturation and value respectively. The Rivers and Water Body map layers were assigned the same pure blue color with a high and medium-high saturation and value respectively. The color choice for the Water theme works well with the other layers and is generally pleasing if not a little too dark for this map. Overall, the first map produced by the cartographic expert system is reasonably well symbolized and provides quite an improvement over the default symbols provided by the mapping software.

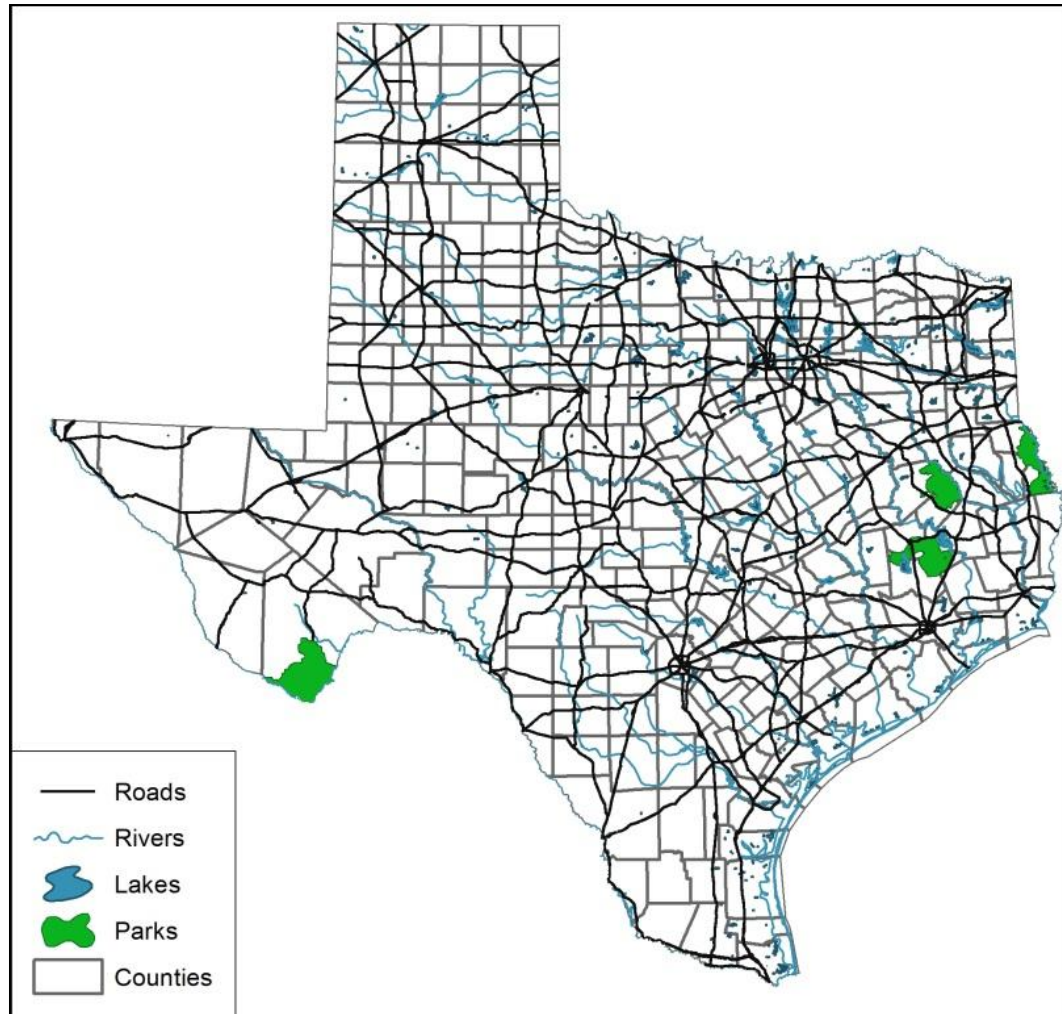


Figure 5.43: First Run of Expert System for Small Scale Data Sets

The second map, shown in Figure 5.44, also provides a reasonably well-designed general reference map based on the inputs. The expert system successfully ordered the map layers based on the preferred map ordering and the result is pleasing. No layer is obscured by another layer. The expert system again provided a white fill color for the Counties map layer. The transportation layer, Major Roads, was assigned a dark purple color and provides a reasonable color choice for the Roads theme, however the purple color does seem to resemble the color of the Water themed features. The Rivers and

Water Bodies map layers are symbolized with a bold blue color and stand out nicely on the map, if not a little too much. Overall, the second map produced by the cartographic expert system provides a good starting point for further tweaks by the map maker.

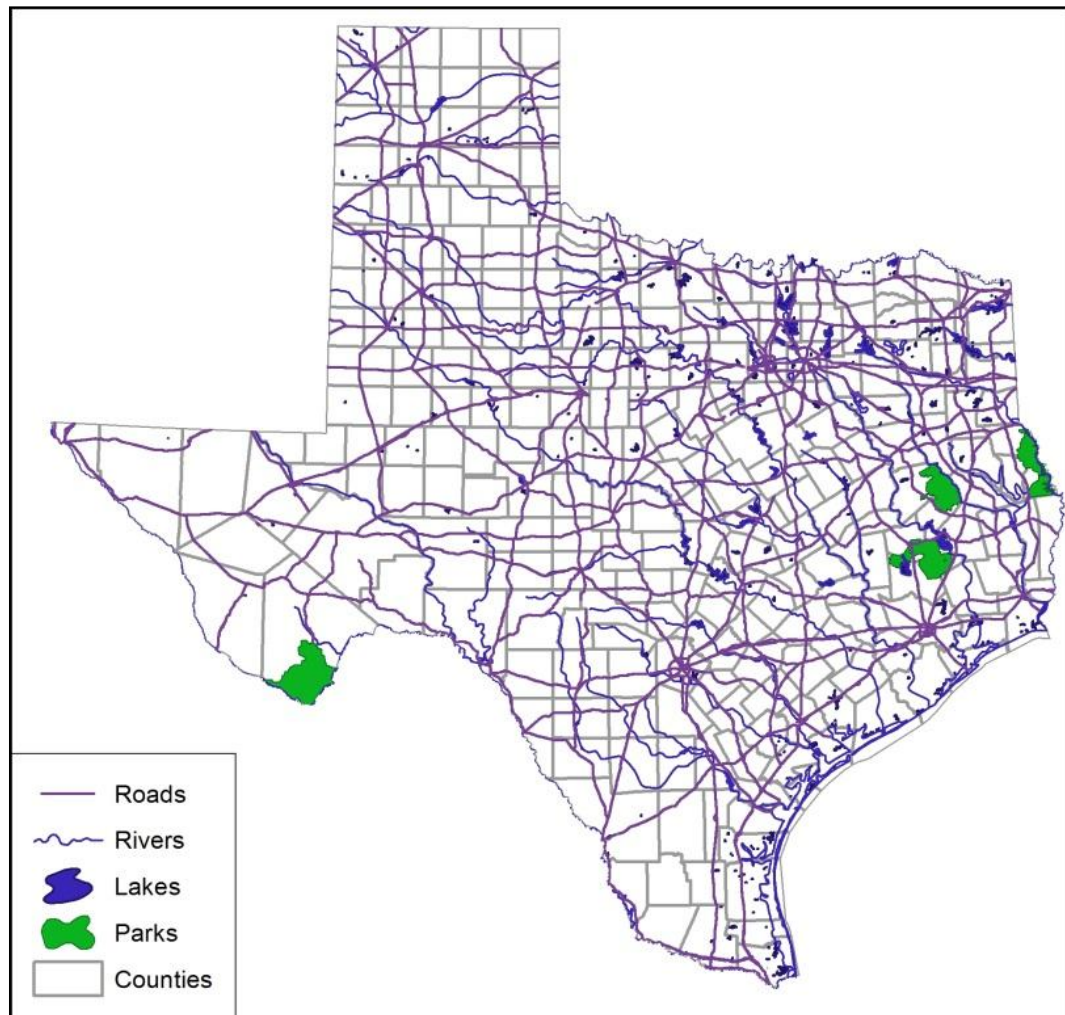


Figure 5.44: Second Run of Expert System for Small Scale Data Sets

The third map, shown in Figure 5.45, once again provides a reasonably well-design general reference map. Once again, the expert system ordered the map layers based on the preferred map ordering and the result is pleasing. The expert system

provided a light tan fill color for the Counties map layer which provides good contrast with the figure map layers. The Major Roads map layer returns to the color black. The remainder of the map layers, have been symbolized with similar colors and line thicknesses as the first map. The third map, like the first two, provide a reasonable starting point for a general reference map.

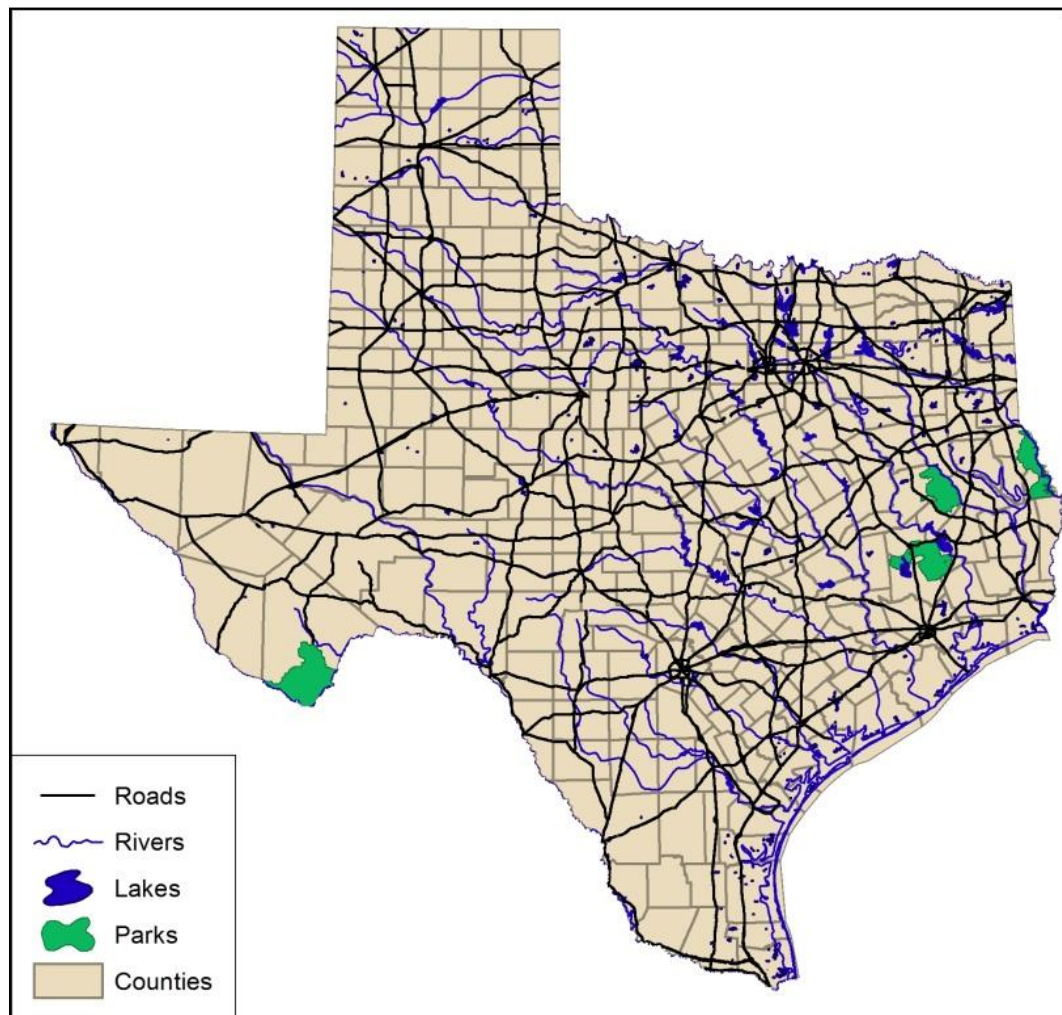


Figure 5.45: Third Run of Expert System for Small Scale Data Sets

These three expert system-designed small scale maps provide reasonably well-designed general reference maps as displayed in Figure 5.46. The three designed maps use acceptable colors, commonly used symbols, and proper draw ordering to create a good representation of the spatial phenomenon being mapped. While these three maps may not be used as the end product for a map product, they provide a good starting point for new map makers, and save time for experienced cartographers as not all symbols will need to be fully modified.

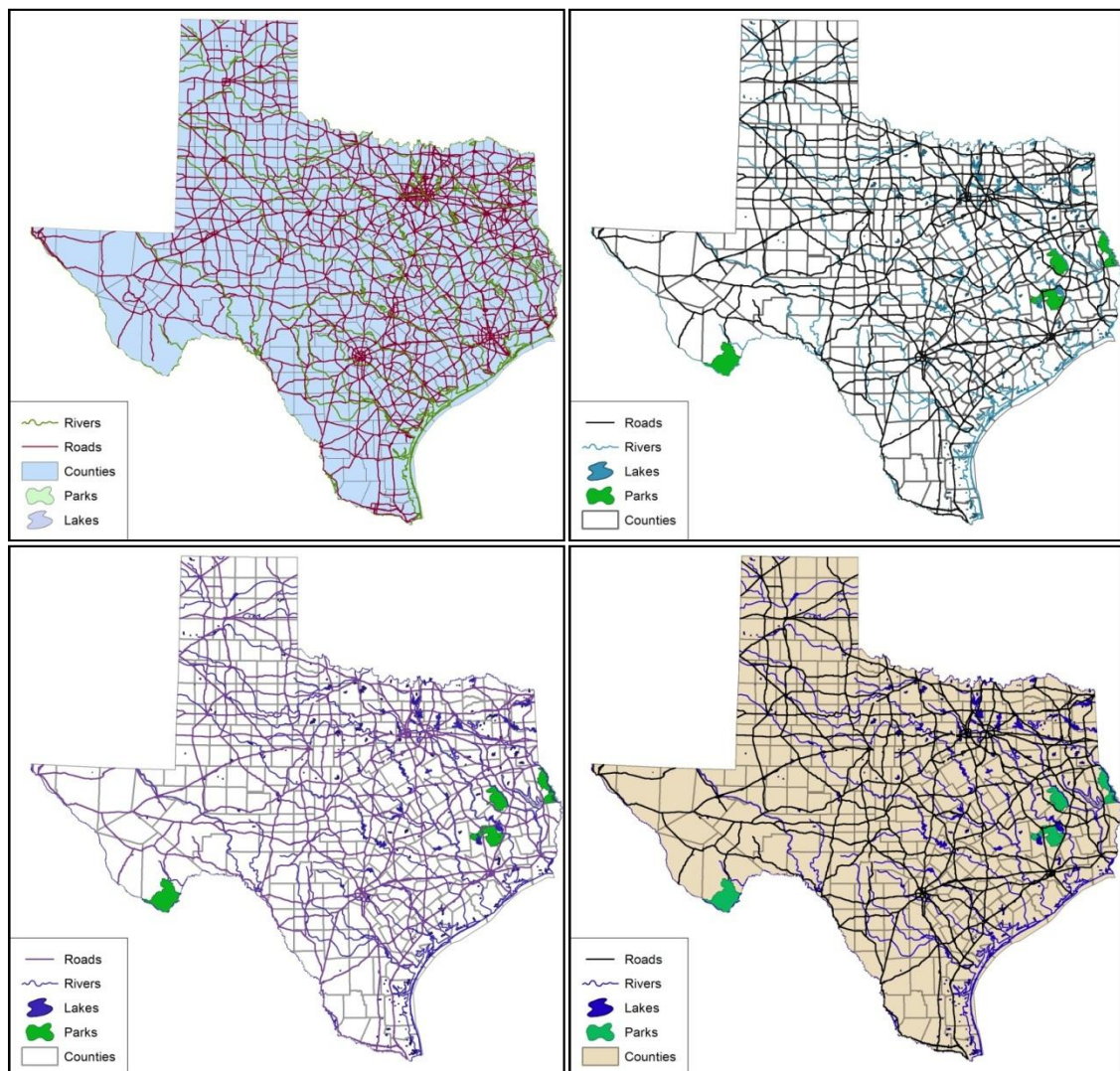


Figure 5.46: Summary of Default and Expert System Designed Maps

Conclusions

The results of this section of the research show that there is promise partially automating the symbolization of a general reference map using a cartographic expert system. The six maps created by the expert system show that it can produce reasonably well-design general reference maps and would be a good starting point for the map maker opposed to the default, random colors chosen when the map layers are added to the map. While not all color combinations seemed to work completely, the resulting maps use safe enough colors and cartographic conventions to be used without any, or, minimal changes depending on the needs of the map maker.

While the results showed only the symbolization of a small and large scale map, the expert system is designed in a way to easily include other named scale ranges which would allow for insertion of additional expert cartographic knowledge. Multiple iterations of the expert system's facts, rules, and templates were undertaken to pare down its complexity to the expert system presented in this chapter. Future expansion to the expert system should be reasonably straight-forward.

Chapter Summary

The purpose of designing and implementing a cartographic expert system was to determine whether a computer program can design an attractive general reference map with minimal input from the user. With only the map layer name entered by the map maker, the expert system, and the program designed in Chapter 4 choose the theme, and extract the geometry from the map layer and return a symbolized map layer in context of the other map layers included on the map. The six maps presented in this chapter

demonstrate that the cartographic expert system designed in this research produces reasonably well designed maps and can save time for the map maker in only requiring them to make minor changes, if any at all before use.

CHAPTER 6

CONCLUSIONS

The overarching goal of this dissertation was to create knowledge and tools that disrupts the positive feedback loop reducing the public's cartographic knowledge. To explore a method of disrupting the feedback loop, this research was guided by the primary research question "How can cartographic knowledge be successfully expressed in an artificial intelligence system to partially automate the cartographic design process?" To answer the question, this research pursued three major goals that all work towards the partial automation of the cartographic design process. The first objective was to create a map ontology to represent the concept of a map and build a framework on which digital data structures could be built to store cartographic knowledge. The second objective was to data mine keywords, field names, and shape types from geospatial datasets and associated metadata to uncover patterns that are useful for automatic theme identification. The third objective was to build a cartographic expert system that uses its expert knowledge to make symbolization choice for general reference map design. Progress made in this research towards achieving these goals is summarized in this chapter.

The map ontology developed in this dissertation provides a starting point for the construction and sharing of cartographic knowledge in the cartography and geographic information science communities. Many ontologies currently exist that describe objects related to mapping, but no ontology framework exists to bring the knowledge together in

the context of a cartographic map. The map ontology presented in Chapter 2 provides the structure and framework needed to link ontologies of objects to a map.

The most important contribution of the map ontology to the field of cartography is the attempt at formalizing the vocabulary, structure, and relationships of a map. This explicit designation of a map can be useful to multiple user groups such as cartographers, computer programmers, and data producers. Cartographers can share their cartographic expertise using a unified language and structure when describing how their knowledge relates to the map. When multiple cartographers subscribe to the same ontology, their knowledge can be shared in an easier fashion since no translation between concepts of what comprises a map needs to be undertaken. For computer scientist who are writing mapping software, for instance, the map ontology provides a logical structure of how a map is structured from the objects in the real world (spatial phenomenon) through the data sets (graphic variables and attributes) and symbolization (visual variables) on to the map body and its arrangement with other supporting elements (layout elements and labels). This structure that the map ontology provides can assist the computer scientist in relating the parts of a map to the constructs programmed in a computer to build a mapping software product. Lastly, data producers can benefit from the map ontology by linking ontologies of spatial phenomenon to the related classes in the map ontology. By linking these ontologies together, other users of the map ontology will be able to more easily understand the construction of the data sets as they relate to producing a map showing the spatial phenomenon.

The use of ontologies is still an emerging trend in geography, however, the literature suggests that as the geography, computer science, and information technology

fields continue to evolve, ontologies will provide an important service for information interoperability. The map ontology created in this research is an important starting point for unifying ontologies for the purpose of mapping.

The second objective of data mining geospatial datasets was to determine whether datasets and metadata could be successfully data mined for use in identifying data sets commonly found on general reference maps. The innovative idea behind this goal is to provide a method for a computer program to autonomously identify a data set's theme. With automatic theme identification, benefits are possible for both map makers and computer programmers. For map makers, having the computer automatically determine the theme of a data set has the potential to streamline many processes and reduce time required to complete tasks. The fewer inputs required by the user for map making, the more automated map making can become. For the computer programmers, automatic detection removes the need for input from the user and will allow the program to operate autonomously where it previously required human input for this step.

The results of the data mining shown in Chapter 4 demonstrates that data mining metadata and geospatial datasets can lead to successful classification of datasets into themes. Keywords and field names found in the metadata documents provided the most relevant predictors, while shape type provided some significant prediction information for a few themes. While the performance of the automatic classification could use some improvement, initial results are promising and, with future refinement, it is believed that automatic theme identification through data mining will continue to improve in its accuracy.

The third objective, building a cartographic expert system, addresses the goal of automatic symbolization of map layers on a general reference map. Providing a computer with the faculties to infer the best symbol attributes for map layers is an important step towards the automation of general reference map production. The maps produced by the expert system created in this research demonstrate that a computer can autonomously symbolize map layers in an attractive fashion.

The facts, templates, and rules created for the cartographic expert system demonstrate how a reasonably straight-forward program design can produce acceptable general reference map symbolization. Attempting to emulate the same behavior using traditional procedural programming methods would require significantly more data structures, loops, if-then statements, and other complicating constructs compared with the construction of an expert system.

Collectively, successful completion of these three objectives answers the primary research question in the affirmative. Additionally, this research produced three innovations in the field of cartography: the production of a map ontology, indication that metadata can be successfully data mined for automatic theme detection, and proof that an expert system can create a well-designed general reference map.

Conclusions

Combining the map ontology, automatic theme identification, and the cartographic expert system is potentially game changing for the democratization of mapping. With future ingestion of additional cartographic knowledge and added complexity of rules, an expert system can be developed that will allow a wide audience of

map makers the tools necessary to develop an attractive map with little-to-no input other than the data sets. Providing this amount of automation for map makers can serve to save an enormous amount of time and lower the entry point for map making.

With a widespread user base linking to the map ontology, the sharing of data sets will become easier for both people and computers. Unifying the vocabulary allows for more efficient searches for data and the structure of the ontology will allow computer programs to go beyond keywords for searching and examine the context of the data and allow for advanced comparisons and transformations.

Over the past thirty years, a revolution, spurred by technological innovation and driven by the importance placed on spatial analysis, has prompted a paradigm shift in the mapping world. Individuals once unable to participate in the map making process, find themselves with the technology, if not the cartographic knowledge, to map whatever phenomena they desire. This revolution, more commonly referred to as the democratization of mapping, has often placed the role of technology (in the map making process) above that of the knowledge of sound map making. With the democratization of mapping comes a larger community of map makers and increased consumption of poorly designed maps and public perception of what a well-designed map is. To fight the trend of poorly designed maps, map makers must be educated in cartography and cartographic design; however, many barriers exist in accomplishing what should be considered the democratization of cartography. The ideas, information, and tools presented in this research provide multiple starting points for taking the democratization of mapping to the next level: the democratization of cartography.

Future Work

This research aimed and succeeded at partially automating the cartographic design process, however, there is still much research to be done. Five areas of work that are immediately apparent are: the wide release of the map ontology, exploring additional facets for data mining, automatic scale determination of datasets, build additional functionality into the expert system, and exploring methods of cartographic knowledge extraction. Each of these five future research areas will be outlined to provide a starting point for forming future research questions.

Construction of the map ontology provides an important starting point for sharing and collaborating. However, the map ontology is of no use if it is not widely available to the cartographic community. Therefore, the map ontology will be placed online and advertised to the cartographic community so that it may be adopted and built upon. The end goal is for the map ontology to be the basis for the construction of a larger, more detailed, cartographic ontology that provides a vehicle for greater sharing and structuring of maps.

Future work in data mining of geospatial metadata has a large amount of potential. This research focused on geometric shape type, keywords, and field names. This is but a small amount of what is stored in geospatial metadata and the datasets themselves. Additional facets of metadata and datasets should be explored for areas of exploitation relevant to automatic theme identification. Facets that seem attractive targets are lineage, abstract, and attribute entries. These three facets of metadata and datasets might hold theme-identifying information and would hopefully increase the accuracy of the results displayed in this research.

A third area of future research is in automatic determination of the scale at which geospatial datasets have been created. The cartographic expert system designed in this research assumed that the user provided scale-appropriate data for symbolization. Allowing for such freedom may not be beneficial to novice mappers that do not know how ill-advised mapping data at a greatly enlarged or reduced scale can be. If an algorithm was produced that could determine the mapping scale of a dataset, then users could be alerted when they are mapping at too large or small of a scale relative to the dataset.

The fourth area of research that should be explored is the addition of functionality to the expert system designed in this research. Specifically, building the ability of the expert system to determine the *type* of features inside of a theme would allow for more powerful symbolization. For instance, this research can currently automatically identify datasets that are of the theme “road”, but cannot further break down the data records into classes, or *types*, of roads. If this functionality was available, then the cartographic knowledge related to symbolizing roads based on type could be included and would provide additional benefit for the user. Initial thoughts lead to exploring patterns available in road datasets and linking them to broad classes of roads.

The last apparent area of future work is in researching and developing a methodology for extracting expert cartographic knowledge and translating that knowledge into computer form. This research used a simple method of knowledge extraction and conversion into computer form, however, a more rigorous, structured method should be investigated.

REFERENCES

- Agarwal, P. 2005. Ontological considerations in GIScience. *International Journal of Geographical Information Science* 19 (5):501–536.
- Arentze, T. A., A. W. J. Borghers, and J. P. Timmermans. 1996. An efficient search strategy for site-selection decisions in an expert system. *Geographical Analysis* 28 (April):126–46.
- Armstrong, M. P., S. De, P. J. Densham, P. Lolonis, G. Rushton, and V. K. Tewari. 1990. A knowledge-based approach for supporting locational decision making. *Environment and Planning B: Planning and Design* 17 (July):341–64.
- Audi, R. 1995. *The Cambridge Dictionary of Philosophy*. Cambridge, UK: Cambridge University Press.
- Berlinski, D. 2001. *The Advent of the Algorithm: The 300-Year Journey from an Idea to the Computer*. Mariner Books.
- Berners-Lee, T., J. Hendler, and O. Lassila. 2001. The Semantic Web. *Scientific American* (May 2001). <http://www.scientificamerican.com/article.cfm?id=the-semantic-web> (last accessed 5 March 2010).
- Bertin, J. 1983. *Semiology of Graphics*. Madison, Wisconsin: University of Wisconsin Press.
- Bossler, J. D., D. L. Pendleton, G. F. Swetnam, R. L. Vitalo, C. R. Schwarz, S. Apler, and H. P. Danley. 1988. Knowledge-Based Cartography: The NOS Experience. *The American Cartographer* 15 (2):169–161.
- Breiman, L., J. H. Friedman, R. A. Olshen, and C. J. Stone. 1984. *Classification and regression trees*. Belmont, CA: Wadsworth.
- Buttenfield, B., M. Gahegan, H. Miller, and M. Yuan. 2000. Geospatial Data Mining and Knowledge Discovery. http://www.ucgis.org/priorities/research/research_white/2000%20Papers/emerging/gkd.pdf.
- Carlson, W. 2003. A Critical History of Computer Graphics and Animation. <http://design.osu.edu/carlson/history/lessons.html> (last accessed 3 August 2011).
- Chandrasekaran, B., J.R. Josephson, and V. R. Benjamins. 1998. The Ontology of Tasks and Methods. In *Proceedings of Knowledge Acquisition, Modeling and Management*

1998. Banff, Alberta, Canada <http://ksi.cpsc.ucalgary.ca/KAW/KAW98/chandra/> (last accessed 12 April 2010).

Chandrasekaran, B., John R. Josephson, and V. R. Benjamins. 1999. What are Ontologies, and Why Do We Need Them? *IEEE Intelligent Systems* 14 (1):20–26.

Chang, K. 2008. *Introduction to Geographic Information Systems*. New York: McGraw-Hill.

Christensen, J., J. Marks, and S. Shieber. 1995. An Empirical Study of Algorithms for Point-Feature Label Placement. *ACM Transactions on Graphics* 14 (3):203–232.

Curry, B., and L. Moutinho. 1992. Computer Models for Site Location Decisions. *International Journal of Retail and Distribution Management* 20:12–23.

CyCorp. 2010. OpenCyc. *Overview of OpenCyc*. <http://www.cyc.com/cyc/opencyc> (last accessed 9 August 2010).

Dent, B. D., J. S. Torguson, and T. W. Hodler. 2009. *Cartography: Thematic Map Design* Sixth ed. New York: McGraw-Hill.

Dobson, J. E. 1979. Automated Geography. *The Professional Geographer* 35 (2):135–145.

Doddi, S., M. V. Marathe, A. Mirzaian, B. M. E. Moret, and B. Zhu. 1997. Map labeling and its generalizations. In *Proceedings of the eighth annual ACM-SIAM symposium on Discrete algorithms*, 148–157. New Orleans, Louisiana, United States: Society for Industrial and Applied Mathematics <http://portal.acm.org/citation.cfm?id=314250> (last accessed 15 March 2010).

Esri. 2011. What is ArcPy? *ArcGIS Desktop 10 Help*. <http://help.arcgis.com/en/arcgisdesktop/10.0/help/index.html#//000v000000v7000000> (last accessed 7 January 2012).

Fayyad, U. M., G. Piatetsky-Shapiro, and P. Smyth. 1996. From data mining to knowledge discovery: an overview. In *Advances in knowledge discovery and data mining*, 1–34. American Association for Artificial Intelligence.

Federal Geographic Data Committee. 1998. *FGDC-STD-001-1998. Content standard for digital geospatial metadata (revised June 1998)*. Washington, DC: Federal Geographic Data Committee. http://www.fgdc.gov/standards/projects/FGDC-standards-projects/metadata/base-metadata/v2_0698.pdf.

Feigenbaum, E. A., and P. McCorduck. 1983. *The Fifth Generation: Artificial Intelligence and Japan's Computer Challenge to the World* 1st Printing. Addison Wesley Publishing Company.

- Fernandez, M., A. Gomez-Perez, and N. Juristo. 1997. *METHONTOLOGY: From Ontological Art Towards Ontological Engineering*. AAAI. www.aaai.org.
- Fischer, M. M. 1994. From conventional to knowledge-based geographic information systems. *Computers*, 18 (July-August):233–42.
- Forrest, D. 1993. Expert Systems and Cartographic Design. *The Cartographic Journal* 29 (2):3.
- . 1999a. Developing Rules for Map Design: A Functional Specification for a Cartographic-Design Expert System. *Cartographica* 36 (3):31.
- . 1999b. Geographic Information: Its Nature, Classification, and Cartographic Representation. *Cartographica* 36 (2).
- Garosi, F. 2008. *PyCLIPS*. <http://pyclips.sourceforge.net/web/>.
- Federal Geographic Data Committee. 2000. Content Standard for Digital Geospatial Metadata Workbook.
- Giarratano, J. C., and G. D. Riley. 2005. *Expert Systems, Principles and Programming*. Course Technology.
- Goodchild, M. 2000. Cartographic Futures on a Digital Earth. *Cartographic Perspectives* 36:3–11.
- Gruber, T. R. 1993. Toward Principles for the Design of Ontologies Used for Knowledge Sharing. *International Journal Human-Computer Studies* 43:907–928.
- Guarino, N. 1997. Semantic Matching: Formal Ontological Distinctions for Information Organization, Extraction, and Integration. In *Information Extraction: A Multidisciplinary Approach to an Emerging Information Technology*, International Summer School, SCIE-9, ed. M. Pazienza, 139–170. Franscati, Italy.
- Guarino, N., and P. Giaretta. 1995. Ontologies and Knowledge Bases: Towards a Terminological Clarification. In *Towards Very Large Knowledge Bases*, ed. N. J. I. Mars. IOS Press.
- Hand, D. J. 1998. Data Mining - Reaching Beyond Statistics. *Research in Official Statistics* 2:5–17.
- Hodler, Thomas (Cartographer). 2007. The Official Map of Conyers and Rockdale County Georgia [general reference map]. Conyers-Rockdale Chamber of Commerce, Conyers, GA.
- Howard, H. H. 2004. Expert Systems Teaches Cartographic Design. *Geospatial Solutions* 14 (8):22.

- Initiative, D. C. M. 2008. *Dublin Core Metadata Element Set, Version 1.1*.
- Iosifescu-Enescu, I., and L. Hurni. 2007. Towards Cartographic Ontologies or “How Computers Learn Cartography.”
- Jackson, P. 1990. *Introduction to expert systems* 2nd ed. Reading, MA: Addison-Wesley.
- Jenks, G. F. 1976. Contemporary Statistical Maps - Evidence of Spatial and Geographic Ignorance. *American Cartographer* 3:11–18.
- Krygier, J. 2005. *Is Cartography Dead?*
- Kuhn, W. 2001. Ontologies in support of activities in geographical space. *International Journal of Geographical Information Science* 15 (7):613–631.
- Kumar, N. 2000. Automation and Democratization of Cartography: An Example of a Mapping System at CEM, University of Durham. *The Cartographic Journal* 37 (1).
- Leung, L., and K. S. Leung. 1993. An intelligent expert system shell for knowledge-based geographic information systems. *International Journal of Geographic Information Systems* 7 (May-June):189–214.
- Lundh, F. 2005. *ElementTree*. <http://effbot.org/zone/element-index.htm>.
- Lutz, M., and E. Klien. 2006. Ontology-based retrieval of geographic information. *International Journal of Geographical Information Science* 20 (3):233–260.
- Lutz, Michael. 2007. Ontology-Based Descriptions for Semantic Discovery and Composition of Geoprocessing Services. *Geoinformatica* 11 (1):1–36.
- Mackaness, W. A. 1986. Towards a Cartographic Expert System. In *Auto Carto London*, ed. M. Blakemore, 578 – 587. London, England.
- Mark, D., M. Egenhofer, S. Hirtle, and B. Smith. 2000. UCGIS Emerging Research Theme: Ontological Foundations for Geographic Information Science. http://www.ucgis.org/priorities/research/research_white/2000%20papers/emerging/ontology_new.pdf.
- Mattmiller, B. 2006. Technology helps foster “democratization of cartography” (Sept. 20, 2006). *University of Wisconsin-Madison News*. <http://www.news.wisc.edu/12916> (last accessed 12 March 2010).
- Mennis, J., and J. W. Liu. 2005. Mining Association Rules in Spatio-Temporal Data: An Analysis of Urban-Socioeconomic and Land Cover Change. *Transact* 9 (1):5–17.
- Mierswa, I., M. Wurst, R. Klinkenberg, M. Scholz, et al. 2006. YALE: Rapid Prototyping for Complex Data Mining Tasks. In *Proceedings of the 12th ACM SIGKDD international*

conference on Knowledge discovery and data mining - KDD '06, 935. Philadelphia, PA, USA <http://rapid-i.com/content/view/30/214/lang,en/> (last accessed 14 February 2011).

Mitchell, T. M. 1997. *Machine Learning* 1st ed. McGraw-Hill Science/Engineering/Math.

Monmonier, M. S. 1982. Computer-Assisted Cartography: Principles and Prospects. Englewood Cliffs, NJ: Prentice Hall College Div.

Monmonier, M. S. 1984. Geographic Information and Cartography. *Progress in Human Geography* 8:381–391.

Mozolin, M. V. 1997. Inductive Machine Learning Methods in Geographic Research.

Muehrcke, P. C., and J. O. Muehrcke. 1998. *Map Use: Reading, Analysis, and Interpretation* Fourth. Madison, Wisconsin: JP Publications.

Muller, J. C. 1983. Ignorance graphique ou cartographie de l'ignorance. *Cartographica* 20 (17-30).

Muller, J. C., R. D. Johnson, and L. R. Vanzella. 1986. A Knowledgebased Approach for Developing Cartographic Expertise. In *Proceedings of the Second International Symposium on Data Handling*. Seattle, Washington.

Muller, J. C., and W. Zeshen. 1990. A Knowledge Based System for Cartographic Symbol Design. *Cartographic Journal* 27 (1):24–30.

Murakami, H. 1990. Automatic feature extraction for map revision.

Myatt, G. 2007. *Making Sense of Data*. Hoboken, New Jersey: John Wiley & Sons, Inc.

National Park Service. 2006. Blue Ridge Parkway [general reference map].

Novick, D., and K. Ward. 2006. Why Don't People Read the Manual? In *Proceedings of SIGDOC 2006*. Myrtle Beach, SC http://works.bepress.com/david_novick/14.

Nyerges, T., and P. Jankowski. 1989. A knowledge base for map projection selection. *The American Cartographer* 16 (January):29–38.

Olson, J. M. 2004. Cartography 2003. *Cartographic Perspectives* 47.

Openshaw, S., and C. Openshaw. 1997. *Artificial Intelligence in Geography*. New York: John Wiley & Sons, Inc.

Ortolana, L., and C. D. Perman. 1989. Applications to Urban Planning: An Overview. In *Expert Systems: Applications to Urban Planning*, 133–43. New York: Springer-Verlag.

- Rand McNally. 2004. Georgia Highways & Interstates [general reference map]. Skokie, IL.
- Raubal, M. 2001. Ontology and epistemology for agent-based wayfinding simulation. *International Journal of Geographical Information Science* 15 (7):653–665.
- Rhind, D. 1977. Computer-Aided Cartography. *Transactions of the Institute of British Geographers* 2 (1):71–97. (last accessed 3 August 2011).
- Robinson, Arthur H., Joel L. Morrison, Phillip C. Muehrcke, A. Jon Kimerling, and Stephen C. Guptill. 1995. *Elements of Cartography* 6th ed. New York: Wiley.
- Robinson, G., and M. Jackson. 1985. *Expert Systems in Map Design*.
- Robinson, V., A. U. Frank, and M. A. Blaze. 1986. Expert systems applied to problems in geographic information systems: introduction, review and prospects. *Computers, Environment, and Urban Systems* 11 (4):161–73.
- Rød, J. K., and F. Ormeling. 2001. An agenda for democratising cartographic visualization. *Norwegian Journal of Geography* 55 (1):38–41.
- Russell, S., and P. Norvig. 2003. *Artificial Intelligence: A Modern Approach* 2nd ed. Upper Saddle River, New Jersey: Prentice-Hall Inc.
- Schlimmer, J. C., and P. Langley. 1991. *Paradigms for Machine Learning*. NASA Ames Research Center: NASA.
http://ntrs.nasa.gov/archive/nasa/casi.ntrs.nasa.gov/19920016857_1992016857.pdf.
- Skidmore, A. K. 1996. An operational GIS expert system for mapping forest soils. *Photogrammetric Engineering and Remote Sensing* 62 (May):501–12.
- Slocum, Terry A., Robert B. McMaster, Fritz C. Kessler, and Hugh H. Howard. 2008. *Thematic Cartography and Geovisualization* 3rd ed. Upper Saddle River, New Jersey: Prentice Hall.
- Smith, T. R. 1984. Artificial Intelligence and its Applicability to Geographical Problem Solving. *The Professional Geographer* 36 (2):147–158.
- Snyder, J. P. 1987. Map Projections - A Working Manual.
http://onlinepubs.er.usgs.gov/djvu/PP/PP_1395.pdf.
- Sowa, J. F. 1998. *Knowledge representation: logical, physical, and computational foundations*. Boston: PWS Publishing Co.
- Texas Department of Transportation. 2007. Texas Department of Transportation Official Travel Map [general reference map]. Austin, TX.

- Timpf, S. 2002. Ontologies of wayfinding: a traveler's perspective. *Networks and Spatial Economics* 2:9–33.
- Tobler, W. R. 1959. Automation and Cartography. *Geographical Review* 49 (4):526–534.
- Uschold, M. 1996. Building Ontologies: Towards a Unified Methodology. In *Proceedings of Expert Systems '96*. Cambridge, UK.
- Uschold, M., and M. Gruninger. 1996. Ontologies: Principles, Methods and Applications. *Knowledge Engineering Review* 11 (2).
- W3C. 1999. *HTML 4.01 Specification*. W3C. <http://www.w3.org/TR/1999/REC-html401-19991224/> (last accessed 7 January 2012).
- . 2004. OWL Web Ontology Language Reference. <http://www.w3.org/TR/owl-ref/> (last accessed 9 September 2010).
- . 2008. Extensible Markup Language (XML) 1.0 (Fifth Edition). W3C. <http://www.w3.org/TR/REC-xml/> (last accessed 7 January 2012).
- Wood, D. 2003. Cartography is Dead (Thank God!). *Cartographic Perspectives* 45.
- Wright, J. R. 1989. ISIS: Toward an Integrated Spatial Information Systems. In *Expert Systems*, 43–66. New York: Springer-Verlag.
- Ying, S., and L. Li. 2005. Knowledge Representation of Cartographic Representation. In *Proceedings of International Symposium on Spatio-Temporal Modeling*. Peking, China: ISPRS.

APPENDIX A

ONTOLOGY VOCABULARY

This vocabulary was created as part of the specification process of the ontology development. Each term was deemed important in determining the basic concept of a map and would need to be included, or at least considered, when constructing the map ontology. The terms are logically grouped based on how they interact and relate with each other.

- Map
 - General Reference Map
 - Thematic Map
 - Qualitative Map
 - Quantitative Map
- Map Projection
 - Projection Properties
 - Azimuthal
 - Compromise
 - Conformal
 - Equal Area
 - Equidistant
- Production Medium
 - Paper

- Digital
- Attribute
 - Interval
 - Nominal
 - Ordinal
 - Ratio
- Graphic
 - Raster
 - Vector
 - Point
 - Line
 - Polygon
- Layout Element
 - Ancillary
 - Text
 - Object
 - Directional Indicator
 - Graticule
 - Label
 - Legend
 - Map Body
 - Metadata
 - Neat line

- Title
 - Scale
 - Graphical
 - Representative
 - Verbal
- Spatial Phenomenon
 - Continuous
 - Discrete
- Visual Variable
 - Arrangement
 - Focus
 - Hue
 - Orientation
 - Saturation
 - Shape
 - Size
 - Texture
 - Value

APPENDIX B

METADATA CLEANING PROGRAM

The purpose of this program is to crawl through each metadata document and 1) search for documents that had formatting problems and remove them from the population; 2) compile descriptive statistics about the metadata documents to uncover commonalities and anomalies within the data; 3) extract keywords found in each metadata document and write the total of each keyword, compiled by theme, to an output file; 4) remove keywords that do not lend themselves to theme identification such as “at”, “the”, “none”, or “and”; and 5) combine plural and singular forms of keywords into one entry.

This program was written in the Python programming language and uses ESRI’s ArcPy application programming interface (ESRI 2011) and the ElementTree Python module (Lundh 2005).

```
from elementtree.ElementTree import parse
import os, re, arcpy

def countKeywords(theDir):
    #Holds list and count of keywords found
    myDict = {}

    #Holds list and count of field names
    fieldNamesDict = {}

    #Holds total count of field names
    totalFieldNames = 0

    #holds directories parsed so to not duplicate reading .xml and .txt
    files
    parsedFiles = {}

    #holds running total of keywords matched for average later
    totalKeywords = 0
```

```

#Top-level of directory we wish to extract keywords from
#print results to files in folder specified by user
#outputFolder = raw_input("Input/Output folder please: ").strip()
#top = outputFolder #copy for the loops
outputFolder = theDir
top = theDir

#counters to keep track of how much has been parsed
numSHPParsed = 0
numXMLParsed = 0
numXMLMatched = 0
numHTMLParsed = 0
numHTMLMatched = 0
numTXTParsed = 0
numTXTMatched = 0
pluralsRemoved = 0

#function that removes HTML tags and returns stripped string
def remove_html_tags(data):
    p = re.compile(r'<.*?>')
    return p.sub('', data)

#This section will walk through the directories under 'top' to find
.xml files
# then extract keywords
for root, dirs, files in os.walk(top, topdown=True):
    for name in files:
        #get extension of file to see if it is .xml
        ext=os.path.splitext(name)[1].lower()
        if (ext == '.xml'):
            #Record that we have looked at this directory
            parsedFiles[root] = 1
            numXMLParsed += 1
            #Read xml file into elementtree
            #walk through xml to find keywords
            tree = parse(os.path.join(root, name))
            elem = tree.getroot()
            idinfo = elem.find("idinfo")
            if idinfo is not None:
                keywords = idinfo.find("keywords")
                if keywords is not None:
                    themeKeywords = keywords.find("theme")
                    if themeKeywords is not None:
                        keysMatched = 0
                        #Open keyword (.kwd) file for writing
                        #if not os.path.exists(fileName[:-
4]+'.kwd'):
                            keywordFile = open(fileName[:-4] + '.kwd',
'w')
                            for node in
themeKeywords.getiterator('themekey'):
                                if (node is not None) and node.text is
not None:
                                    if ((node.text.lower() == "none")
or (re.match("required\.*",node.text.lower()))):
                                        continue
                                    else:

```

```
#####
commas and space

fndKeywords.split(",")

kwd.split(" ")

re.split('\s+',kwd)

kwdsSplitOnSpaces:
kwdSplitOnSpace.strip()
kwdSplitOnSpace.rstrip(".")

#####

keyword file

keywordFile.write(kwdSplitOnSpace + "\n")

exists in dictionary, increment

myDict):

myDict[kwdSplitOnSpace] +=1

node.text.lower() + " found in XML and incremented\n"
entry and set to 1

myDict[kwdSplitOnSpace] = 1

node.text.lower() + " found in XML and created\n"
    #if keysMatched > 0 and noneFlag == 0:
    if keysMatched > 0:
        numXMLMatched += 1

    #close keyword file
    if keywordFile:
        keywordFile.close()
    del keywordFile

    #get extension to see if it is .shp
    # if found, extract fields into .fld file
    #####elif (ext == '.shp'):
    if (ext == '.shp'):
        numSHPParsed +=1
        fileName = os.path.join(root, name)

#####
# Trying to split based on

fndKeywords = node.text.lower()
keywordList =

for kwd in keywordList:
    kwd = kwd.strip()
    #kwdsSplitOnSpaces =

    kwdsSplitOnSpaces =

    for kwdSplitOnSpace in

        kwdSplitOnSpace =

        kwdSplitOnSpace =

        totalKeywords+=1
        keysMatched += 1

#Write keyword to

#if keyword already

if (kwdSplitOnSpace in

#print
#otherwise create new

else:

#print

#####
```

```

        #If .fld file does not exist, create file and ready for
writing
        fieldNameFile = open(fileName[:-4] + '.fld', 'w')
        #
        fieldNameFile.write(arcpy.Describe(fileName).ShapeType)
        #get list of fields
        fieldList = arcpy.ListFields(fileName)
        for field in fieldList:
            theField = field.name.lower()
            #ignore FID and Shape
            if theField != "fid" and theField != "shape":
                totalFieldNames += 1
                if fieldNameFile:
                    fieldNameFile.write(field.name + "\n")
                if theField in fieldNamesDict:
                    fieldNamesDict[theField] += 1
                else:
                    fieldNamesDict[theField] = 1
        if fieldNameFile:
            fieldNameFile.close()

#delete arcpy
#del arcpy

#This section will walk through the directories under 'top' to find
.htm/.html files
#The reason this is not done in the loop before is because I want
to give priority
# to .xml files since they are more well structured.
for root, dirs, files in os.walk(top, topdown=True):
    for name in files:
        #get extension of file to see if it is .htm/.html
        ext=os.path.splitext(name)[1].lower()
        if ((ext == '.htm' or ext == '.html') and root not in
parsedFiles):
            #Record that we have looked at this directory
            parsedFiles[root] = 1
            numHTMLParsed +=1
            #open file for reading
            f = open(os.path.join(root, name), 'r')
            numHTMLMatched1 = 0
            #open/create keyword file
            keywordFile = open(fileName[:-4] + '.kwd', 'w')
            #go line-by-line through file
            for line in f:
                #strip HTML tags
                line = remove_html_tags(line)
                line.strip()
                #look for "Theme_Keyword:"
                m = re.search("\.*Theme_Keyword\s*:", line)
                if m:
                    #get everything after matched re
                    matchedKeyword = line[m.end():].strip()
                    #####
                    # Trying to split based on commas and space

```



```

        fndKeywords = matchedKeyword.lower()
        keywordList = fndKeywords.split(",")
        for kwd in keywordList:
            kwd = kwd.strip()
            #kwdSplitOnSpaces = kwd.split(" ")
            kwdSplitOnSpaces = re.split('\s+',kwd)
            for kwdSplitOnSpace in kwdSplitOnSpaces:
                totalKeywords+=1
                numHTMLMatched1 +=1
                kwdSplitOnSpace =

kwdSplitOnSpace.strip()

kwdSplitOnSpace.rstrip(".")

#####
                                #Write keyword to keyword file
                                keywordFile.write(kwdSplitOnSpace +

"\n")

                                #Increment in dictionary or create new
entry

                                if kwdSplitOnSpace in myDict:
                                    myDict[kwdSplitOnSpace] += 1
                                else:
                                    myDict[kwdSplitOnSpace] = 1

#look for "Theme Keyword:"
m = re.search("\.*Theme\s*Keyword\s*:", line)
if m:
    #get everything after matched re
    matchedKeyword = line[m.end():].strip()
    #####
    # Trying to split based on commas and space
    fndKeywords = matchedKeyword.lower()
    keywordList = fndKeywords.split(",")
    for kwd in keywordList:
        kwd = kwd.strip()
        #kwdSplitOnSpaces = kwd.split(" ")
        kwdSplitOnSpaces = re.split('\s+',kwd)
        for kwdSplitOnSpace in kwdSplitOnSpaces:
            totalKeywords+=1
            numHTMLMatched1 +=1
            kwdSplitOnSpace =

kwdSplitOnSpace.strip()

kwdSplitOnSpace.rstrip(".")

#####
                                #Write keyword to keyword file
                                keywordFile.write(kwdSplitOnSpace +

"\n")

                                #Increment in dictionary or create new
entry

                                if kwdSplitOnSpace in myDict:
                                    myDict[kwdSplitOnSpace] += 1
                                else:
                                    myDict[kwdSplitOnSpace] = 1

```

```

        #keep track of whether the file had at least one
matched keyword
        if numHTMLMatched1 > 0:
            numHTMLMatched +=1
        #else:
            #print "Did not match: " + os.path.join(root, name)
        #close file
        f.close()

        #close keyword file
        if keywordFile:
            keywordFile.close()
        del keywordFile

    #This section will walk through the directories under 'top' to find
.txt/.met files
    #The reason this is not done in the loop before is because I want
to give priority
    # to .xml files since they are more well structured.
    for root, dirs, files in os.walk(top, topdown=True):
        for name in files:
            #get extension of file to see if it is .txt
            ext=os.path.splitext(name)[1].lower()
            #check for .txt/.met and a .xml file was not parsed in this
directory
            if ((ext == '.txt' or ext == '.met') and root not in
parsedFiles):
                numTXTParsed += 1
                #open file for reading
                f = open(os.path.join(root, name), 'r')
                numTXTMatched1 = 0
                #open/create keyword file
                keywordFile = open(fileName[:-4] + '.kwd', 'w')
                #go line-by-line through file
                for line in f:
                    #look for "Theme_Keyword:"
                    m = re.search("\s*Theme_Keyword\s*:", line)
                    if m:
                        #get everything after matched re
                        matchedKeyword = line[m.end():].strip()
                        #####
                        # Trying to split based on commas and space
                        fndKeywords = matchedKeyword.lower()
                        keywordList = fndKeywords.split(",")
                        for kwd in keywordList:
                            kwd = kwd.strip()
                            #kwsSplitOnSpaces = kwd.split(" ")
                            kwsSplitOnSpaces = re.split('\s+',kwd)
                            for kwdSplitOnSpace in kwsSplitOnSpaces:
                                totalKeywords+=1
                                numTXTMatched1 +=1
                                kwdSplitOnSpace =
kwdSplitOnSpace.strip()
                                kwdSplitOnSpace =
kwdSplitOnSpace.rstrip(".")

```

```
#####
#Write keyword to keyword file
keywordFile.write(kwdSplitOnSpace +
"\n")

#Increment in dictionary or create new
entry

if kwdSplitOnSpace in myDict:
    myDict[kwdSplitOnSpace] += 1
else:
    myDict[kwdSplitOnSpace] = 1

#look for "Theme Keyword:"
m = re.search("\s*Theme\s*Keyword\s*:", line)
if m:
    #get everything after matched re
    matchedKeyword = line[m.end():].strip()
    #####
    # Trying to split based on commas and space
    fndKeywords = matchedKeyword.lower()
    keywordList = fndKeywords.split(",")
    for kwd in keywordList:
        kwd = kwd.strip()
        #kwdSplitOnSpaces = kwd.split(" ")
        kwdSplitOnSpaces = re.split('\s+', kwd)
        for kwdSplitOnSpace in kwdSplitOnSpaces:
            totalKeywords+=1
            numTXTMatched1 +=1
            kwdSplitOnSpace =

kwdSplitOnSpace.strip()

kwdSplitOnSpace.rstrip(".")

#####
#Write keyword to keyword file
keywordFile.write(kwdSplitOnSpace +
"\n")

#Increment in dictionary or create new
entry

if kwdSplitOnSpace in myDict:
    myDict[kwdSplitOnSpace] += 1
else:
    myDict[kwdSplitOnSpace] = 1

#keep track of whether the file had at least one
matched keyword
if numTXTMatched1 > 0:
    numTXTMatched +=1
#close file
f.close()

#close keyword file
if keywordFile:
    keywordFile.close()
del keywordFile

#Combine plural and non-plural keywords
dictKeys = myDict.keys()
```

```

    #Remove "ies" from end of word and add "y" to see if base word
exists
    for key in dictKeys:
        if key[-3:] == "ies":
            if (key[0:-3] + "y") in myDict:
                try:
                    #print "removing duplicate: " + key[0:-3] + "y with
" + key

                    myDict[(key[0:-3]+"y")] += myDict.pop(key)
                    pluralsRemoved += 1
                except KeyError:
                    continue

    #Remove "es" from end of word to see if base word exists
    for key in dictKeys:
        if key[-2:] == "es":
            if (key[0:-2]) in myDict:
                try:
                    #print "removing duplicate: " + key[0:-2] + " with
" + key

                    myDict[key[0:-2]] += myDict.pop(key)
                    pluralsRemoved += 1
                except KeyError:
                    continue

    #Remove "s" from end of word to see if base word exists
    for key in dictKeys:
        if key[-1:] == "s":
            if key[0:-1] in myDict:
                try:
                    #print "removing duplicate: " + key[0:-1] + " with
" + key

                    myDict[key[0:-1]] += myDict.pop(key)
                    pluralsRemoved += 1
                except KeyError:
                    continue

    #Remove meaningless terms. e.x. and, or, not, a, this, " "
    meaninglessDict =
{'and':1,'or':1,'not':1,'the':1,'a':1,'this':1,'that':1,'in':1,
'of':1,'is':1,'has':1,'on':1, 'at':1, '':1,'by':1,'for':1,
'please':1,'was':1,'sake':1,'also':1,'pdigirolamo@atlantaregional.com':
1,'wwang@atlantaregional.com':1}
    for key in myDict.keys():
        if key in meaninglessDict:
            totalKeywords -= myDict[key]
            myDict.pop(key)
            #print "Removed '" + key + "' from keyword list"

    #Write keywords to file
    f = open(outputFolder + "\\keywords.txt",'w')

    for k, v in myDict.iteritems():
        f.write(k + ";;" + str((float(v)/float(totalKeywords))*100.0) +
"\n")
    f.close()

```

```

    #Write field names to file
    f = open(outputFolder + "\\fieldNames.txt", 'w')
    for a,b in fieldNamesDict.iteritems():
        f.write(a + ";;" + str((float(b)/float(totalFieldNames))*100.0)
+ "\n")
    f.close()

    averageFieldNames = totalFieldNames / numSHPParsed

    averageKeywords = totalKeywords /
(numXMLMatched+numHTMLMatched+numTXTMatched)

    f = open(outputFolder + "\\extractLog.txt", 'w')
    f.write("Number of SHP files parsed: " + str(numSHPParsed) + "\n")
    f.write("Number of XML files parsed: " + str(numXMLParsed) + "\n")
    f.write("Number of XML files with keywords matched: " +
str(numXMLMatched) + "\n")
    f.write("Number of HTML files parsed: " + str(numHTMLParsed) +
"\n")
    f.write("Number of HTML files with keywords matched: " +
str(numHTMLMatched) + "\n")
    f.write("Number of TXT/MET files parsed: " + str(numTXTParsed) +
"\n")
    f.write("Number of TXT/MET files with keywords matched: " +
str(numTXTMatched) + "\n")
    f.write("Plurals removed: " + str(pluralsRemoved) + "\n")
    f.write("Average number of keywords per metadata file: " +
str(averageKeywords) + "\n")
    f.write("Total number of keywords found: " + str(totalKeywords) +
"\n")
    f.write("Number of unique keywords:" + str(len(myDict)) + "\n")
    f.write("Total number of field names found: " +
str(totalFieldNames) + "\n")
    f.write("Number of unique field names:" + str(len(fieldNamesDict))
+ "\n")
    f.write("Average number of field names per shapefile: " +
str(averageFieldNames) + "\n")

    f.close()

    print "Number of SHP files parsed: " + str(numSHPParsed)
    print "Number of XML files parsed: " + str(numXMLParsed)
    print "Number of XML files with keywords matched: " +
str(numXMLMatched)
    print "Number of HTML files parsed: " + str(numHTMLParsed)
    print "Number of HTML files with keywords matched: " +
str(numHTMLMatched)
    print "Number of TXT/MET files parsed: " + str(numTXTParsed)
    print "Number of TXT/MET files with keywords matched: " +
str(numTXTMatched)
    print "Plurals removed: " + str(pluralsRemoved)
    print "Average number of keywords per metadata file: " +
str(averageKeywords)
    print "Total number of keywords found: " + str(totalKeywords)
    print "Number of unique keywords:" + str(len(myDict))
    print "Total number of field names found: " + str(totalFieldNames)

```

```

    print "Number of unique field names:" + str(len(fieldNamesDict))
    print "Average number of field names per shapefile: " +
str(averageFieldNames)

```

```

def main():
    directoryList = [r"L:\",
                     r"L:\Boundaries",
                     r"L:\Boundaries\StateCounty",
                     r"L:\Boundaries\ZipCode",
                     r"L:\Physical",
                     r"L:\Physical\Contours",
                     r"L:\Physical\Water",
                     r"L:\PointsOfInterest",
                     r"L:\PointsOfInterest\Airports",
                     r"L:\PointsOfInterest\Hospital",
                     r"L:\PointsOfInterest\Parks",
                     r"L:\PointsOfInterest\PoliceFire",
                     r"L:\Transportation",
                     r"L:\Transportation\Railroads",
                     r"L:\Transportation\Roads",
                     r"L:\Transportation\Trails",]
    for theDir in directoryList:
        countKeywords(theDir)

if __name__ == "__main__":
    main()

```

APPENDIX C

DATA MINING ALGORITHM-FRIENDLY DATA FORMATTING PROGRAM

The purpose of this program is to take the output of the Metadata Cleaning Program listed in Appendix B and format it into a data mining algorithm-friendly format. This is accomplished writing the following to an output file: 1) comparing keywords and field names for each theme against all shapefiles, 2) the shapefile name, 3) the geometry type of each shapefile, and 4) the training variable.

This program was written in the Python programming language and uses ESRI's ArcPy application programming interface (ESRI 2011) and the ElementTree Python module (Lundh 2005).

```
import os, arcpy, string
from elementtree.ElementTree import parse

#This function will compare the keywords and field names in 'theDir'
with the other category directories.
# it will then print out the results of the match.
#It will also print out whether the results of the match correctly
identifies the category or not
def compareStuff(theDir, outDir):

    #Holds list of directories to iterate over. Represents categories
    minorCategoryDirectoryList = [r"L:\Boundaries\StateCounty",
                                   r"L:\Boundaries\ZipCode",
                                   r"L:\Physical\Contours",
                                   r"L:\Physical\Water",
                                   r"L:\PointsOfInterest\Airports",
                                   r"L:\PointsOfInterest\Hospital",
                                   r"L:\PointsOfInterest\Parks",
                                   r"L:\PointsOfInterest\PoliceFire",
                                   r"L:\Transportation\Railroads",
                                   r"L:\Transportation\Roads",
                                   r"L:\Transportation\Trails"]

    #holds field names in base category
    fieldNamesDict = {}
    #holds keywords in base category
    keywordsDict = {}
```

```

        #read the field names for the base category and fill the field
names dictionary with them
        fieldNamesFile = theDir + r"\fieldNames.txt"
        f = open(fieldNamesFile, 'r')
        for line in f:
            line.strip()
            before, sep, after = line.partition(";;")
            #fieldNamesDict[before.lower()] = int(after)
            fieldNamesDict[before.lower()] = float(after)
        f.close()

        #read the keywords for the base category and fill the keywords
dictionary with them
        keywordFile = theDir + r"\keywords.txt"
        f = open(keywordFile, 'r')
        for line in f:
            line.strip()
            before, sep, after = line.partition(";;")
            #keywordsDict[before.lower()] = int(after)
            keywordsDict[before.lower()] = float(after)
        f.close()

        #Iterate through each category...
        for top in minorCategoryDirectoryList:
            #holds the numeric result of the matched keywords, keyed by
file name
            keywordsResult = {}
            #holds the numeric result of the matched field names, keyed by
file name
            fieldNamesResult = {}
            #holds the shape type, keyed by file name
            shapeType = {}

            #for each file in the top directory
            for root, dirs, files in os.walk(top, topdown=True):
                for name in files:
                    ext=os.path.splitext(name)[1].lower()
                    #once we reach a .fld file, which contains a list of
field names for a shapefile
                    if(ext == '.fld'):
                        #get the shape type for use later
                        shapeType[os.path.join(root, name)] =
arcpy.Describe(os.path.join(root, name[:-3]+'shp')).shapeType
                        #open file holding field names
                        f = open(os.path.join(root, name), 'r')
                        #reset match counter
                        result = 0
                        #for each field name...
                        for line in f:
                            line = line.strip()
                            #if this field name matches a field name in
the base class
                            if line.lower() in fieldNamesDict:
                                #increment the result by the score of
matching that field name

```



```

                                #result +=
int(fieldNamesDict[line.lower()])
                                result +=
float(fieldNamesDict[line.lower()])
                                #close field (.fld) file
                                f.close()
                                #add the field name file name to the result
dictionary so we can recover results later
                                fieldNamesResult[os.path.join(root, name)] = result

                                #once we reach a .kwd file, which contains a list of
keywords for a shapefile
                                if(ext == '.kwd'):
                                    #open file holding keywords
                                    f = open(os.path.join(root, name), 'r')
                                    #reset match counter
                                    result = 0
                                    #for each keyword....
                                    for line in f:
                                        line = line.strip()
                                        #If this keyword matches a keyword in the base
class...
                                        if line.lower() in keywordsDict:
                                            #increment the result by the score of
matching that keyword
                                            #result += int(keywordsDict[line.lower()])
                                            result += float(keywordsDict[line.lower()])
                                            #close keyword (.kwd) file
                                            f.close()
                                            #add the keyword file name to the result dictionary
so we can recover the result later
                                            keywordsResult[os.path.join(root, name)] = result

                                #Output file name = outDir plus leaf directory of base class +
.csv
                                outFile = outDir + theDir[string.rfind(theDir, "\\"): ] + ".csv"
                                #If the file already exists, append to it
                                if os.path.isfile(outFile):
                                    f = open(outFile, 'a')
                                #If the file does not exist, create a new one and write a
header
                                else:
                                    f = open(outFile, 'w')
                                    f.write("ShapeType, Correct, MatchField, MatchKey,
FileName\n")

                                #for each key, value pair in the field names result
dictionary...
                                for k, v in fieldNamesResult.iteritems():
                                    ##if we are comparing the base class vs the base class...
                                    if top == theDir:
                                        #write out result but indicate "yes" to show a positive
result
                                        outputString = shapeType[k] + ",Yes," + str(v) + ","
                                    ##if we are comparing the base class against a different
class...
                                    else:

```

```

        #write out result and indicate "no" to show a negative
result
        outputString = shapeType[k] + ",No," + str(v) + ","

        ##if the key is also in the keywordsResults dictionary...
        if (k[:-3]+'kwd') in keywordsResult:
            #write out result
            outputString += str(keywordsResult[k[:-3]+'kwd'])
        else:
            outputString += "0"

        #Add the shapefile name and then add a newline
        outputString += "," + k[:-3]+'shp' + "\n"

        #write the built output string to the output file
        f.write(outputString)

    #close the output file
    f.close()

def main():
    #Holds list of directories to iterate over. Represents categories
    minorCategoryDirectoryList = [r"L:\USDA\Boundaries\StateCounty",
                                   r"L:\Boundaries\ZipCode",
                                   r"L:\Physical\Contours",
                                   r"L:\Physical\Water",
                                   r"L:\PointsOfInterest\Airports",
                                   r"L:\PointsOfInterest\Hospital",
                                   r"L:\PointsOfInterest\Parks",
                                   r"L:\PointsOfInterest\PoliceFire",
                                   r"L:\Transportation\Railroads",
                                   r"L:\Transportation\Roads",
                                   r"L:\Transportation\Trails"]

    #Get output directory for compare files
    outDir = raw_input("Output DIRECTORY (WITHOUT TRAILING SLASH)
location")

    #For each category...
    for theDir in minorCategoryDirectoryList:
        #do the comparison
        compareStuff(theDir, outDir)

if __name__ == "__main__":
    main()

```

APPENDIX D

DATASET THEME PREDICTION PROGRAM

The purpose of this program is to predict the theme of an input dataset based on the included geospatial metadata. This is accomplished through scoring how much the information in the geospatial metadata match the information extracted from the training datasets. The score is then evaluated against the minimum scores required to be considered a member of a theme. The program then proffers to the user the highest ranked themes until the user selects the correct theme or the program does not have any candidate themes with a sufficient score to proffer to the user.

This program was written in the Python programming language and uses ESRI's ArcPy application programming interface (ESRI 2011) and the ElementTree Python module (Lundh 2005).

```
import os, sys, array, arcpy, re
from elementtree.ElementTree import parse

#This program will make a decision on what the theme of the data is
# based on matched field names, keywords and shapetype

#Set to 1 to print score numbers
#Set to 0 to hide score numbers
printScores = 0

#Root directory to theme list files
#It will look inside the same directory where the python script lives.
If this does not work, you can manually set the path
themeDirRoot = sys.path[0] + r"\ThemeLists"

#This function will call individual functions that make decisions on
how much a theme matches with a theme
#Input: theme to match against
#       fieldNames of target file
#       keywordsList of target file
#Output: boolean (1/0) stating whether it was a match or not
def makeDecision(theme, fieldNames, keywordsList, shapeType):
    if theme=='Airports':
```

```

        return decideOnAirports(theme, fieldNames, keywordsList,
shapeType)
    if theme=='Contours':
        return decideOnContours(theme, fieldNames, keywordsList,
shapeType)
    if theme=='Hospitals':
        return decideOnHospitals(theme, fieldNames, keywordsList,
shapeType)
    if theme=='Parks':
        return decideOnParks(theme, fieldNames, keywordsList,
shapeType)
    if theme=='PoliceFire':
        return decideOnPoliceFire(theme, fieldNames, keywordsList,
shapeType)
    if theme=='Railroads':
        return decideOnRailroads(theme, fieldNames, keywordsList,
shapeType)
    if theme=='Roads':
        return decideOnRoads(theme, fieldNames, keywordsList,
shapeType)
    if theme=='StateCounty':
        return decideOnStateCounty(theme, fieldNames, keywordsList,
shapeType)
    if theme=='Trails':
        return decideOnTrails(theme, fieldNames, keywordsList,
shapeType)
    if theme=='Water':
        return decideOnWater(theme, fieldNames, keywordsList,
shapeType)
    if theme=='ZipCode':
        return decideOnZipCode(theme, fieldNames, keywordsList,
shapeType)

#Decides on statecounty theme
#True if:
#   Match based on fields > 50.157
def decideOnStateCounty(theme, fieldNames, keywordsList, shapeType):
    #holds final result of matches
    result = 0

    #Get dictionary of field names and score
    fieldsDict = getFieldNames(theme)
    for k,v in fieldsDict.iteritems():
        for field in fieldNames:
            if field == k:
                result += float(v)
    if printScores == 1:
        print "Results for statecounty: " + str(result)

    #Requirements are met, return 1
    if result > 50.157:
        return result / 50.157
    else:    #Requirements not met, return 0
        #return 0
        return result / 50.157

#Decides on zipcode theme

```

```

#True if:
# Match based on keywords > 25.893 or keywords > 14.583 and shape =
polygon or fields > 32.031
def decideOnZipCode(theme, fieldNames, keywordsList, shapeType):
    result = 0
    resultList = []

    #Get dictionary of keywords and scores
    keywordsDict = getKeywordsAndScores(theme)
    for k,v in keywordsDict.iteritems():
        for keyword in keywordsList:
            if keyword == k:
                result += float(v)
    if printScores == 1:
        print "Results for zipcode KEYWORDS: " + str(result)

    #If matches against keywords...
    if result > 25.893:
        return result / 25.893
    else:
        resultList.append(result/25.893)

    #failed test 1, going to test 2
    #see if keywords meets this test minimum
    if result > 14.583 and shapeType.lower() == "polygon":
        if printScores == 1:
            print "Results for water SHAPE: " + shapeType
            #####Maybe add multiplier here
        return result / 14.583
    else:
        if shapeType.lower() == "polygon":
            multiplier = 1.5
        else:
            multiplier = .5
        resultList.append((result / 14.583) * multiplier)

    #failed test 1 and test 2, going to test 3
    result = 0
    #Get dictionary of field names and score
    fieldsDict = getFieldNames(theme)
    for k,v in fieldsDict.iteritems():
        for field in fieldNames:
            if field == k:
                result += float(v)
    if printScores == 1:
        print "Results for zipcode FIELD: " + str(result)

    #Requirements are met, return 1
    if result > 32.031:
        return result / 32.031
    else:
        resultList.append(result/32.031)

    highestResult = 0
    for res in resultList:
        if res > highestResult:
            highestResult = res

```

```

    return highestResult

#Checks for match between two terms with plurals removed from keyword
#Returns true if match found
#Returns false if match not found
def checkForMatch(keyword, keyword2):
    if keyword == keyword2:
        return "True"
    if keyword[-3:] == "ies":
        #Remove "ies" from end of word and add "y" to see if
base word exists
        if (keyword[0:-3] + "y") == keyword2:
            print "Removing ies for y from " + keyword
            return "True"
    if keyword[-2:] == "es":
        #Remove "es" from end of word to see if base word exists
        if (keyword[0:-2]) == keyword2:
            print "Removing es from " + keyword
            return "True"
    if keyword[-1:] == "s":
        print "Removing s from " + keyword
        #Remove "s" from end of word to see if base word exists
        if (keyword[0:-1]) == keyword2:
            return "True"

    #No match found, return False
    return "False"

#Decides on water theme
#True if:
#   Match based on keywords > 8.841
def decideOnWater(theme, fieldNames, keywordsList, shapeType):
    result = 0

    #Get dictionary of keywords and scores
    keywordsDict = getKeywordsAndScores(theme)
    for k,v in keywordsDict.iteritems():
        for keyword in keywordsList:
            if checkForMatch(keyword,k) == "True":
                result += float(v)

    if printScores == 1:
        print "Results for water: " + str(result)

    #If matches against keywords...
    if result > 8.841:
        return result / 8.841
    else:
        #return 0
        return (result / 8.841)

```

```

#Decides on water theme
#True if:
# Match based on keywords > 8.841
def decideOnWaterOLD(theme, fieldNames, keywordsList, shapeType):
    result = 0

    #Get dictionary of keywords and scores
    keywordsDict = getKeywordsAndScores(theme)
    for k,v in keywordsDict.iteritems():
        for keyword in keywordsList:
            if keyword == k:
                result += float(v)
            elif keyword[-3:] == "ies":
                #Remove "ies" from end of word and add "y" to see if
base word exists
                if (keyword[0:-3] + "y") == k:
                    print "Removing ies for y from " + keyword
                    result += float(v)
                elif keyword[-2:] == "es":
                    #Remove "es" from end of word to see if base word
exists
                if (keyword[0:-2]) == k:
                    print "Removing es from " + keyword
                    result += float(v)
                elif keyword[-1:] == "s":
                    print "Removing s from " + keyword
                    #Remove "s" from end of word to see if base
word exists
                if (keyword[0:-1]) == k:
                    result += float(v)

    if printScores == 1:
        print "Results for water: " + str(result)

    #If matches against keywords...
    if result > 8.841:
        return result / 8.841
    else:
        #return 0
        return (result / 8.841)

#Decides on trails theme
#True if:
# Match based on keywords > 22.222
def decideOnTrails(theme, fieldNames, keywordsList, shapeType):
    result = 0

    #Get dictionary of keywords and scores
    keywordsDict = getKeywordsAndScores(theme)
    for k,v in keywordsDict.iteritems():
        for keyword in keywordsList:
            if keyword == k:
                result += float(v)
    if printScores == 1:
        print "Results for trails: " + str(result)

    #If matches against keywords...

```

```

    if result > 22.222:
        return result / 22.222
    else:
        #return 0
        return result / 22.222

#Decides on railroads theme
#True if:
# Match based on keywords > 25.941
def decideOnRailroads(theme, fieldNames, keywordsList, shapeType):
    result = 0

    #Get dictionary of keywords and scores
    keywordsDict = getKeywordsAndScores(theme)
    for k,v in keywordsDict.iteritems():
        for keyword in keywordsList:
            if keyword == k:
                result += float(v)
    if printScores == 1:
        print "Results for railroads: " + str(result)

    #If matches against keywords...
    if result > 25.941:
        return result / 25.941
    else:
        #return 0
        return result / 25.941

#Decides on policefire theme
#True if:
# Match based on keywords > 9.348 or keywords > 8.587 and point
def decideOnPoliceFire(theme, fieldNames, keywordsList, shapeType):
    result = 0

    #Get dictionary of keywords and scores
    keywordsDict = getKeywordsAndScores(theme)
    for k,v in keywordsDict.iteritems():
        for keyword in keywordsList:
            if keyword == k:
                result += float(v)
    if printScores == 1:
        print "Results for policefire: " + str(result)
        print "Results for policefire SHAPE: " + shapeType

    #If matches against keywords...
    if result > 9.348:
        return result / 9.348

    #If keywords match and is point shapefile
    if result > 8.587 and shapeType.lower() == "point":
        #####Consider adding multiplier
here too
        return result / 8.587
    else:
        if shapeType.lower() == "point":
            multiplier = 1.5
        else:

```



```

        multiplier = .5
    #return 0
    if printScores == 1:
        print "PoliceFire Multiplier: " + str(multiplier)
    return (result / 8.587) * multiplier

#Decides on airport theme
#True if:
# Match based on Field names > 17.328 or Keywords > 32.716
def decideOnAirports(theme, fieldNames, keywordsList, shapeType):
    #holds final result of matches
    result = 0
    resultList = []

    #Get dictionary of field names and score
    fieldsDict = getFieldNames(theme)
    for k,v in fieldsDict.iteritems():
        for field in fieldNames:
            if field == k:
                result += float(v)
    if printScores == 1:
        print "Results for parks FIELD NAMES: " + str(result)

    #First test to see if fields match
    #Requirements are met, return 1
    if result > 17.328:
        return result / 17.328
    else:
        resultList.append(result/17.328)

    #Failed first test, trying second test
    result = 0

    #Get dictionary of keywords and scores
    keywordsDict = getKeywordsAndScores(theme)
    for k,v in keywordsDict.iteritems():
        for keyword in keywordsList:
            if keyword == k:
                result += float(v)
    if printScores == 1:
        print "Results for parks KEYWORDS: " + str(result)

    #If matches against field names (and previously keywords)...
    if result > 32.716:
        return result / 32.716
    else:
        resultList.append(result/32.716)

    highestResult = 0
    for res in resultList:
        if res > highestResult:
            highestResult = res

    return highestResult

#Decides on contours theme
#True if:

```

```

# Match based on Field names > 31.559 or 14.259 < Field names <=
15.209
def decideOnContours(theme, fieldNames, keywordsList, shapeType):
    #holds final result of matches
    result = 0

    #Get dictionary of field names and score
    fieldsDict = getFieldNames(theme)
    for k,v in fieldsDict.iteritems():
        for field in fieldNames:
            if field == k:
                result += float(v)
    if printScores == 1:
        print "Results for contours: " + str(result)

    #Requirements are met, return 1
    #if result > 31.559 or 14.259 < result <= 15.209:
    if result > 31.559:
        return result / 31.559
    else: #Requirements not met, return 0
        return result/31.559

#Decides on parks theme
#True if:
# Match based on Field names > 13.743 or Keywords > 22.407
def decideOnParks(theme, fieldNames, keywordsList, shapeType):
    #holds final result of matches
    result = 0
    resultList = []

    #Get dictionary of field names and score
    fieldsDict = getFieldNames(theme)
    for k,v in fieldsDict.iteritems():
        for field in fieldNames:
            if field == k:
                result += float(v)
    if printScores == 1:
        print "Results for parks FIELD NAMES: " + str(result)

    #First test to see if fields match
    #Requirements are met, return 1
    if result > 13.743:
        return result / 13.743
    else:
        resultList.append(result/13.743)

    #Failed first test, trying second test
    result = 0

    #Get dictionary of keywords and scores
    keywordsDict = getKeywordsAndScores(theme)
    for k,v in keywordsDict.iteritems():
        for keyword in keywordsList:
            if keyword == k:
                result += float(v)
    if printScores == 1:
        print "Results for parks KEYWORDS: " + str(result)

```

```

#If matches against field names (and previously keywords)...
if result > 22.407:
    return result / 22.407
else:
    resultList.append(result/22.407)

highestResult = 0
for res in resultList:
    if res > highestResult:
        highestResult = res

return highestResult

#Decides on roads theme
#True if:
# Match based on Field names > 3.701
def decideOnRoads(theme, fieldNames, keywordsList, shapeType):
    #holds final result of matches
    result = 0

    #Get dictionary of field names and score
    fieldsDict = getFieldNames(theme)
    for k,v in fieldsDict.iteritems():
        for field in fieldNames:
            if field == k:
                result += float(v)
    if printScores == 1:
        print "Results for roads: " + str(result)

    #Requirements are met, return 1
    if result > 3.701:
        return result / 3.701
    else: #Requirements not met, return 0
        #return 0
        return result / 3.701

#Decides on hospitals theme
#True if:
# Match based on Keywords > 11.654
def decideOnHospitals(theme, fieldNames, keywordsList, shapeType):
    #holds final result of matches
    result = 0

    #Get dictionary of keywords and scores
    keywordsDict = getKeywordsAndScores(theme)
    for k,v in keywordsDict.iteritems():
        for keyword in keywordsList:
            if keyword == k:
                result += float(v)
    if printScores == 1:
        print "Results for hospitals: " + str(result)

    #Requirements are met, return 1
    if result > 11.654:
        return result / 11.654
    else: #Requirements not met, return 0

```

```

        #return 0
        return result/ 11.654

#Get field names from results files and split into dictionary of
k=field name and v=score
def getFieldNames(theme):
    returnDict = {}
    f = open(themeDirRoot + "\\\" + theme + "_FieldNames.txt",'r')
    for line in f:
        newline = line.strip()
        before, sep, after = newline.partition(";;")
        returnDict[before]=after
    f.close()
    return returnDict

#Get field names from results files and split into dictionary of
k=field name and v=score
def getKeywordsAndScores(theme):
    returnDict = {}
    f = open(themeDirRoot + "\\\" + theme + "_Keywords.txt",'r')
    for line in f:
        newline = line.strip()
        before, sep, after = newline.partition(";;")
        returnDict[before]=after
    f.close()
    return returnDict

#Given a shapefile, the function will attempt to extract and return
# keywords from .xml/.html/.met/.txt files in that order
def getKeywords(targetFile):
    #Check to see if a .xml file exists...
    if os.path.exists(os.path.splitext(targetFile)[0].lower() +
'.xml'):
        return
    getKeywordsFromXML(os.path.splitext(targetFile)[0].lower() + '.xml')
    if os.path.exists(targetFile + '.xml'):
        return getKeywordsFromXML(targetFile + '.xml')
    if os.path.exists(os.path.splitext(targetFile)[0].lower() +
'.html'):
        return
    getKeywordsFromHTML(os.path.splitext(targetFile)[0].lower() + '.html')
    if os.path.exists(os.path.splitext(targetFile)[0].lower() +
'.htm'):
        return
    getKeywordsFromHTML(os.path.splitext(targetFile)[0].lower() + '.htm')
    if os.path.exists(os.path.splitext(targetFile)[0].lower() +
'.met'):
        return
    getKeywordsFromTXTMET(os.path.splitext(targetFile)[0].lower() + '.met')
    if os.path.exists(os.path.splitext(targetFile)[0].lower() +
'.txt'):
        return
    getKeywordsFromTXTMET(os.path.splitext(targetFile)[0].lower() + '.txt')
    #Couldn't find a metadatafile, so return empty array
    returnKeywords = ()
    return returnKeywords

```

```

def getKeywordsFromTXTMET(txtMetFile):
    #This section will walk through the directories under 'top' to find
    .txt/.met files
    #The reason this is not done in the loop before is because I want
    to give priority
    #    to .xml files since they are more well structured.

    #Holds keywords in list
    returnKeywords = []

    #open file for reading
    f = open(txtMetFile, 'r')
    #go line-by-line through file
    for line in f:
        #look for "Theme_Keyword:"
        m = re.search("\s*Theme_Keyword\s*:", line)
        if m:
            #get everything after matched re
            matchedKeyword = line[m.end():].strip()
            #####
            # Trying to split based on commas and space
            fndKeywords = matchedKeyword.lower()
            keywordList = fndKeywords.split(",")
            for kwd in keywordList:
                kwd = kwd.strip()
                #kwdSplitOnSpaces = kwd.split(" ")
                kwdSplitOnSpaces = re.split('\s+', kwd)
                for kwdSplitOnSpace in kwdSplitOnSpaces:
                    kwdSplitOnSpace = kwdSplitOnSpace.strip()
                    kwdSplitOnSpace = kwdSplitOnSpace.rstrip(".")
                    #####
                    #Add keyword to list to return
                    returnKeywords.append(kwdSplitOnSpace)

        #look for "Theme Keyword:"
        m = re.search("\s*Theme\s*Keyword\s*:", line)
        if m:
            #get everything after matched re
            matchedKeyword = line[m.end():].strip()
            #####
            # Trying to split based on commas and space
            fndKeywords = matchedKeyword.lower()
            keywordList = fndKeywords.split(",")
            for kwd in keywordList:
                kwd = kwd.strip()
                #kwdSplitOnSpaces = kwd.split(" ")
                kwdSplitOnSpaces = re.split('\s+', kwd)
                for kwdSplitOnSpace in kwdSplitOnSpaces:
                    numTXTMatched1 +=1
                    kwdSplitOnSpace = kwdSplitOnSpace.strip()
                    kwdSplitOnSpace = kwdSplitOnSpace.rstrip(".")
                    #####
                    #Add keyword to list to return
                    returnKeywords.append(kwdSplitOnSpace)

    #close input file
    f.close()

```

```

    #Return unique list of keywords
    return createUniqueSet(returnKeywords)

#function that removes HTML tags and returns stripped string
def remove_html_tags(data):
    p = re.compile(r'<.*?>')
    return p.sub('', data)

def getKeywordsFromHTML(htmlFile):

    #Holds keywords in list
    returnKeywords = []

    #open html file for reading
    f = open(htmlFile, 'r')
    #go line-by-line through file
    for line in f:
        #strip HTML tags
        line = remove_html_tags(line)
        line.strip()
        #look for "Theme_Keyword:"
        m = re.search("\.*Theme_Keyword\s*:", line)
        if m:
            #get everything after matched re
            matchedKeyword = line[m.end():].strip()
            #####
            # Trying to split based on commas and space
            fndKeywords = matchedKeyword.lower()
            keywordList = fndKeywords.split(",")
            for kwd in keywordList:
                kwd = kwd.strip()
                kwdsSplitOnSpaces = re.split('\s+', kwd)
                for kwdSplitOnSpace in kwdsSplitOnSpaces:
                    kwdSplitOnSpace = kwdSplitOnSpace.strip()
                    kwdSplitOnSpace = kwdSplitOnSpace.rstrip(".")
                    #####
                    #Add keyword to list to return
                    returnKeywords.append(kwdSplitOnSpace)

        #look for "Theme Keyword:"
        m = re.search("\.*Theme\s*Keyword\s*:", line)
        if m:
            #get everything after matched re
            matchedKeyword = line[m.end():].strip()
            #####
            # Trying to split based on commas and space
            fndKeywords = matchedKeyword.lower()
            keywordList = fndKeywords.split(",")
            for kwd in keywordList:
                kwd = kwd.strip()
                kwdsSplitOnSpaces = re.split('\s+', kwd)
                for kwdSplitOnSpace in kwdsSplitOnSpaces:
                    kwdSplitOnSpace = kwdSplitOnSpace.strip()
                    kwdSplitOnSpace = kwdSplitOnSpace.rstrip(".")
                    #####
                    #Add keyword to list to return

```

```

        returnKeywords.append(kwdSplitOnSpace)

#close input file
f.close()

#Return unique list of keywords
return createUniqueSet(returnKeywords)

def getKeywordsFromXML(xmlFile):
    #Holds keywords in list
    returnKeywords = []

    tree = parse(xmlFile)
    elem = tree.getroot()
    idinfo = elem.find("idinfo")
    if idinfo is not None:
        keywords = idinfo.find("keywords")
        if keywords is not None:
            #themeKeywords = keywords.find("theme")
            for themeKeywords in keywords.getiterator('theme'):
                #unindent this line if rolling back
                if themeKeywords is not None:
                    for node in themeKeywords.getiterator('themekey'):
                        if (node is not None) and node.text is not
None:
                            if ((node.text.lower() == "none") or
(re.match("required\.*",node.text.lower()))):
                                continue
                            else:
                                # Trying to split based on commas and
space

                                fndKeywords = node.text.lower()
                                keywordList = fndKeywords.split(",")
                                for kwd in keywordList:
                                    kwd = kwd.strip()
                                    kwdsSplitOnSpaces =
re.split('\s+',kwd)

                                    for kwdSplitOnSpace in
kwdsSplitOnSpaces:
                                        kwdSplitOnSpace =
kwdSplitOnSpace.strip()
                                        kwdSplitOnSpace =
kwdSplitOnSpace.rstrip(".")

                                        #Add keyword to list to return

                                returnKeywords.append(kwdSplitOnSpace)
                                #Return unique list of keywords
                                return createUniqueSet(returnKeywords)

#Returns array with unique items (removes duplicate entries)
def createUniqueSet(seq):
    # Not order preserving
    keys = {}
    for e in seq:
        keys[e] = 1

```

```

    return keys.keys()

def printResults(mostLikely, highestScore):
    if highestScore >= 1:
        print "I am confident that the input shapefile is type " +
mostLikely + " with a score of " + str(highestScore) + "."
    elif highestScore >= .5:
        print "I am reasonably confident that the input shapefile is
type " + mostLikely + " with a score of " + str(highestScore) + "."
    elif highestScore >= .1:
        print "I think that the input shapefile is type " + mostLikely
+ " with a score of " + str(highestScore) + "."
    else:
        print "I cannot match the input shapefile to any theme with a
reasonable amount of certainty. Sorry. The next closest that matches
is: " + mostLikely + " with a score of " + str(highestScore) + "."

def main():
    #list of themes and name of file without extension
    themeLists = ("Airports", "Contours", "Hospitals", "Parks",
"PoliceFire", "Railroads", "Roads",
                  "StateCounty", "Trails", "Water", "ZipCode")
    #Holds results of scores
    scoreResults = {}

    #Flag set to determine if found by any decision tree
    foundByDecTree = 0

    #The file we are trying to make a decision on
    #targetFile = r"D:\Users\Rick\Documents\My
Dropbox\Disseration\Data Mining\Test Shapefiles\fire.shp"
    targetFile = raw_input("Absolute path to shapefile to evaluate: ")
    targetFile.strip()

    if not arcpy.Exists(targetFile):
        print "Could not find the file!"
        sys.exit("Could not find the file you referenced.")

    #Array holding field names of targetFile
    targetFields = []

    #Gets the field names from the target file
    theFields = arcpy.ListFields(targetFile)
    #Appends each field name to the targetFields array for use later
    for line in theFields:
        targetFields.append(line.name.lower())

    #Holds shape type of target file
    targetShape = arcpy.Describe(targetFile).ShapeType

    #Holds dictionary of keywords of target file
    targetKeywords = getKeywords(targetFile)

    #Remove plurals from keywords dictionary

```



```

    #Iterate through each theme...
    for theme in themeLists:
        #make decision against a theme
        scoreResults[theme] = makeDecision(theme, targetFields,
targetKeywords, targetShape)
    if printScores == 1:
        for k,v in scoreResults.iteritems():
            print k + " = " + str(v)

    mostLikely = ""
    highestScore = 0
    for k,v in scoreResults.iteritems():
        if v > highestScore:
            highestScore = v
            mostLikely = k

    #Print results to user
    printResults(mostLikely, highestScore)

    correct = raw_input("Is it " + mostLikely + "? (y/n)")
    while correct == "n":
        previousHighest = highestScore
        highestScore = 0
        mostLikely = ""
        for k,v in scoreResults.iteritems():
            if (highestScore < v < previousHighest):
                highestScore = v
                mostLikely = k
        if highestScore > .1:
            printResults(mostLikely, highestScore)
            correct = raw_input("Is it " + mostLikely + "? (y/n)")
        else:
            printResults(mostLikely, highestScore)
            print "I give up!"
            break;

if __name__ == "__main__":
    main()

```

APPENDIX E

CARTOGRAPHIC EXPERT SYSTEM PROGRAM

The purpose of this expert system program is to accept input representing map layers and their themes, use included cartographic knowledge for map design, and output design decisions. There are three components of the cartographic expert system: deffacts, deftemplates, and defrules. They are presented in this order in this appendix.

deffacts

```
(deffacts MAIN::initial-theme-ordering
  (theme-ordering (theme Airports) (above parks roads water state contours trails
railroads))
  (theme-ordering (theme Parks) (above boundaries))
  (theme-ordering (theme Roads) (above water parks boundaries trails))
  (theme-ordering (theme Water) (above boundaries parks))
  (theme-ordering (theme StateCounty) (above))
  (theme-ordering (theme Contours) (above parks boundaries))
  (theme-ordering (theme Trails) (above contours parks boundaries))
  (theme-ordering (theme emergency) (above parks roads water state contours trails
railroads))
  (theme-ordering (theme Hospitals) (above parks roads water state contours trails
railroads))
  (theme-ordering (theme Railroads) (above parks roads water state contours trails)))
```

```
(deffacts MAIN::initial-color-preferences
  (theme-hsv-color-preferences (theme Water) (geometry polygon) (hue blue) (saturation
high) (value medium-high))
  (theme-hsv-color-preferences (theme Trails) (geometry line) (hue brown) (saturation
high) (value high))
  (theme-hsv-color-preferences (theme Parks) (geometry polygon) (hue green) (saturation
high) (value medium-high))
  (theme-hsv-color-preferences (theme Airports) (geometry point) (hue black) (saturation
none) (value none))
  (theme-hsv-color-preferences (theme StateCounty) (geometry polygon) (hue brown
green white) (saturation medium-low) (value high)))
```

```

(theme-hsv-color-preferences (theme emergency) (geometry point) (hue red) (saturation
high) (value high))
(theme-hsv-color-preferences (theme Hospitals) (geometry point) (hue blue) (saturation
high) (value high))
(theme-hsv-color-preferences (theme Roads) (geometry line) (hue black) (saturation
none) (value medium-low))
(theme-hsv-color-preferences (theme Contours) (geometry line) (hue brown) (saturation
medium) (value medium))
(theme-hsv-color-preferences (theme Railroads) (geometry line) (hue black) (saturation
none) (value none)))

```

```

(deffacts MAIN::hsv-information
(hue-range (name black) (hue-low 0) (hue-high 0))
(hue-range (name red) (hue-low 0) (hue-high 15))
(hue-range (name orange) (hue-low 25) (hue-high 40))
(hue-range (name yellow) (hue-low 50) (hue-high 55))
(hue-range (name blue) (hue-low 190) (hue-high 260))
(hue-range (name brown) (hue-low 35) (hue-high 45))
(hue-range (name green) (hue-low 100) (hue-high 150))
(hue-range (name white) (hue-low 0)(hue-high 0))
(saturation-range (name high) (saturation-low 90) (saturation-high 100))
(saturation-range (name medium-high) (saturation-low 70) (saturation-high 80))
(saturation-range (name medium) (saturation-low 50) (saturation-high 60))
(saturation-range (name medium-low) (saturation-low 30) (saturation-high 40))
(saturation-range (name low) (saturation-low 10) (saturation-high 20))
(saturation-range (name none) (saturation-low 0) (saturation-high 0))
(value-range (name high) (value-low 90) (value-high 100))
(value-range (name medium) (value-low 50) (value-high 60))
(value-range (name medium-high) (value-low 70) (value-high 80))
(value-range (name medium-low) (value-low 30) (value-high 40))
(value-range (name low) (value-low 10) (value-high 20))
(value-range (name none) (value-low 0) (value-high 0)))

```

```

(deffacts MAIN::scales
(scale-level (name large) (min 1000) (max 30000))
(scale-level (name medium) (min 30001) (max 300000))
(scale-level (name small) (min 300001) (max 30000000))
(current-scale))

```

```

(deffacts MAIN::initial-scale
(input-scale (scale 10000)))

```

```

(deffacts MAIN::initial-relative-outline-colors
(relative-outline-color (relative-color-choice darker) (percent-multiplier 0.6))
(relative-outline-color (relative-color-choice same) (percent-multiplier 1.0))
(relative-outline-color (relative-color-choice lighter) (percent-multiplier 1.4)))

```

```
(deffacts MAIN::initial-theme-symbol-preferences
  (theme-symbol-preferences (theme Hospitals) (geometry point) (scale large) (symbol
hospital) (symbol-size 16) (line-width 0) (line-type none) (relative-outline-color same))
  (theme-symbol-preferences (theme Airports) (geometry point) (scale large) (symbol
airport) (symbol-size 16) (line-width 0) (line-type none) (relative-outline-color same))
  (theme-symbol-preferences (theme emergency) (geometry point) (scale large) (symbol
star) (symbol-size 18) (line-width 0.5) (line-type solid) (relative-outline-color darker))
  (theme-symbol-preferences (theme Contours) (geometry line) (scale large) (symbol
line) (symbol-size 0) (line-width 1) (line-type solid) (relative-outline-color same))
  (theme-symbol-preferences (theme Roads) (geometry line) (scale large) (symbol line)
(symbol-size 0) (line-width 4) (line-type solid) (relative-outline-color same))
  (theme-symbol-preferences (theme Roads) (geometry line) (scale medium) (symbol
line) (symbol-size 0) (line-width 2.5) (line-type solid) (relative-outline-color same))
  (theme-symbol-preferences (theme Roads) (geometry line) (scale small) (symbol line)
(symbol-size 0) (line-width 1.5) (line-type solid) (relative-outline-color same))
  (theme-symbol-preferences (theme Trails) (geometry line) (scale large) (symbol line)
(symbol-size 0) (line-width 1) (line-type dashed) (relative-outline-color same))
  (theme-symbol-preferences (theme Railroads) (geometry line) (scale large) (symbol
railroad) (symbol-size 6) (line-width 6) (line-type solid) (relative-outline-color same))
  (theme-symbol-preferences (theme Water) (geometry polygon) (scale large) (symbol
polygon) (symbol-size 0) (line-width 1) (line-type solid) (relative-outline-color darker))
  (theme-symbol-preferences (theme Parks) (geometry polygon) (scale large) (symbol
polygon) (symbol-size 0) (line-width 1) (line-type solid) (relative-outline-color darker))
  (theme-symbol-preferences (theme StateCounty) (geometry polygon) (scale large)
(symbol polygon) (symbol-size 0) (line-width 3) (line-type solid) (relative-outline-color
darker))))
```

deftemplate

```
(deftemplate MAIN::mapLayer
  (slot name (default ?NONE))
  (slot theme (default ?NONE) (allowed-symbols Airports emergency Contours Roads
Trails Railroads StateCounty Water Parks Hospitals))
  (slot geometry (default ?NONE) (allowed-symbols point line polygon))
  (slot drawOrder (default 0))
  (slot hue-value (default none))
  (slot saturation-value (default none))
  (slot value-value (default none))
  (slot symbol-size)
  (slot symbol (default none))
  (slot line-width)
  (slot line-type)
  (slot outline-hue-value)
  (slot outline-saturation-value)
  (slot outline-value-value))
```

```

(deftemplate MAIN::theme-ordering
  (slot theme (default ?NONE))
  (multislot above))

(deftemplate MAIN::theme-hsv-color-preferences
  (slot theme (default ?NONE) (allowed-symbols Airports emergency Contours Roads
Trails Railroads StateCounty Water Parks Hospitals))
  (slot geometry (default ?NONE) (allowed-symbols point line polygon))
  (multislot hue (default ?NONE))
  (multislot saturation (default ?NONE))
  (multislot value (default ?NONE)))

(deftemplate MAIN::hue-range
  (slot name (default ?NONE))
  (slot hue-low (default ?NONE)(type INTEGER)(range 0 360))
  (slot hue-high (default ?NONE)(type INTEGER)(range 0 360)))

(deftemplate MAIN::saturation-range
  (slot name (default ?NONE))
  (slot saturation-low (default ?NONE)(type INTEGER)(range 0 100))
  (slot saturation-high (default ?NONE)(type INTEGER)(range 0 100)))

(deftemplate MAIN::value-range
  (slot name (default ?NONE))
  (slot value-low (default ?NONE)(type INTEGER)(range 0 100))
  (slot value-high (default ?NONE)(type INTEGER)(range 0 100)))

(deftemplate MAIN::theme-symbol-preferences
  (slot theme (default ?NONE) (allowed-symbols Airports emergency Contours Roads
Trails Railroads StateCounty Water Parks Hospitals))
  (slot geometry (default ?NONE) (allowed-symbols point line polygon))
  (slot scale(default ?NONE))
  (slot symbol)
  (slot symbol-size)
  (slot line-width)
  (slot line-type)
  (slot relative-outline-color))

(deftemplate MAIN::relative-outline-color
  (slot relative-color-choice (default ?NONE))
  (slot percent-multiplier(type FLOAT)))

(deftemplate MAIN::scale-level
  (slot name)
  (slot min (type INTEGER))

```

```

(slot max (type INTEGER)))

(deftemplate MAIN::current-scale
  (slot scale (type INTEGER)))

(deftemplate MAIN::input-scale
  (slot scale (type INTEGER)(default ?NONE)))

                                defrules

(defrule MAIN::order-mapLayers
  ?mapLayer1 <- (mapLayer (theme ?theme) (drawOrder ?loc1))
  (theme-ordering (theme ?theme) (above $? ?abovetheme $?))
  ?mapLayer2 <- (mapLayer (theme ?abovetheme) (drawOrder ?loc2))
  (test (> ?loc1 ?loc2))
  =>
  (modify ?mapLayer1 (drawOrder ?loc2))
  (modify ?mapLayer2 (drawOrder ?loc1)))

(defrule MAIN::fix-duplicate-drawOrders
  (declare (salience 30))
  ?mapLayer1 <- (mapLayer (name ?name) (drawOrder ?loc))
  (mapLayer (name ~?name) (drawOrder ?loc))
  =>
  (modify ?mapLayer1 (drawOrder (+ ?loc 1))))

(defrule MAIN::choose-random-color
  (declare (salience 100))
  ?col-pref <- (theme-hsv-color-preferences (hue $?hues))
  =>
  (bind ?len (length$ ?hues))
  (if (neq ?len 1)
    then
      (bind ?chosenhue (nth$ (random 1 ?len) ?hues))
      (modify ?col-pref (hue ?chosenhue))))

(defrule MAIN::assign-color-preferences
  (declare (salience 20))
  ?layer <- (mapLayer (hue-value none) (saturation-value none) (value-value none)
    (theme ?theme) (geometry ?geometry))
  (theme-hsv-color-preferences (theme ?theme) (geometry ?geometry) (hue ?hue)
    (saturation ?saturation) (value ?value $?))
  (saturation-range (name ?saturation) (saturation-low ?sat-low) (saturation-high ?sat-high))
  (value-range (name ?value) (value-low ?val-low) (value-high ?val-high))
  (hue-range (name ?hue) (hue-low ?hue-low) (hue-high ?hue-high))

```

```

=>
  (if (eq ?hue white) then (bind ?sat-low 0)(bind ?sat-high 0)(bind ?val-low 100)(bind
?val-high 100))
  (modify ?layer (hue-value (random ?hue-low ?hue-high)) (saturation-value (random
?satsat-low ?sat-high)) (value-value (random ?val-low ?val-high))))

(defrule MAIN::match-theme-colors
  (mapLayer (theme ?theme) (geometry ?geometry) (hue-value ?hue-value&~none)
(saturation-value ?sat-value&~none) (value-value ?val-value&~none))
  ?layer2 <- (mapLayer (theme ?theme) (geometry ?geometry) (hue-value ~?hue-value)
(saturation-value ~?sat-value) (value-value ~?val-value))
  =>
  (modify ?layer2 (hue-value ?hue-value) (saturation-value ?sat-value) (value-value ?val-
value)))

(defrule MAIN::set-scale
  ?cur-scale <- (current-scale)
  ?in-scale <- (input-scale (scale ?input-scale-value))
  (scale-level (name ?name) (min ?min) (max ?max))
  (test (and (>= ?input-scale-value ?min) (<= ?input-scale-value ?max)))
  =>
  (retract ?in-scale)
  (modify ?cur-scale (scale ?name))
  (printout t "Scale is: " ?name crlf))

(defrule MAIN::assign-symbology
  (current-scale (scale ?current-scale))
  ?layer <- (mapLayer (theme ?theme) (geometry ?geometry) (symbol none) (hue-value
?hue-value) (saturation-value ?saturation-value) (value-value ?value-value))
  (theme-symbol-preferences (scale ?current-scale) (geometry ?geometry) (theme
?theme) (symbol ?symbol) (symbol-size ?symbol-size) (line-width ?line-width) (line-type
?line-type) (relative-outline-color ?relative-outline-color))
  (relative-outline-color (relative-color-choice ?relative-outline-color) (percent-multiplier
?percent-multiplier))
  =>
  (modify ?layer (symbol ?symbol) (symbol-size ?symbol-size) (line-width ?line-width)
(line-type ?line-type) (outline-hue-value ?hue-value) (outline-value-value (* ?value-value
?percent-multiplier)) (outline-saturation-value ?saturation-value)))

```