ABSTRACT

Title of Dissertation: AUTOMATED STRUCTURAL AND SPATIAL COMPREHENSION OF DATA TABLES

Marco David Adelfio, Doctor of Philosophy, 2015

Dissertation directed by: Professor Hanan Samet
Department of Computer Science

Data tables on the Web hold large quantities of information, but are difficult to search, browse, and merge using existing systems. This dissertation presents a collection of techniques for extracting, processing, and querying tables that contain geographic data, by harnessing the coherence of table structures for retrieval tasks. Data tables, including spreadsheets, HTML tables, and those found in rich document formats, are the standard way of communicating structured data for typical computer users. Notably, geographic tables (i.e., those containing names of locations) constitute a large fraction of publicly-available data tables and are ripe for exposure to Internet users who are increasingly comfortable interacting with geographic data using web-based maps. Of particular interest is the creation of a large repository of geographic data tables that would enable novel queries such as "find vacation itineraries geographically similar to mine" for use in trip planning or "find demographic datasets that cover regions X, Y, and Z" for sociological research.

In support of these goals, this dissertation identifies several methods for using the structure and context of data tables to improve the interpretation of the contents, even in the pres-

ence of ambiguity. First, a method for identifying functional components of data tables is presented, capitalizing on techniques for sequence labeling that are used in natural language processing. Next, a novel automated method for converting place references to physical latitude/longitude values, a process known as geotagging, is applied to tables with high accuracy. A classification procedure for identifying a specific class of geographic table, the travel itinerary, is also described, which borrows inspiration from optimization techniques for the traveling salesman problem (TSP). Finally, methods for querying spatially similar tables are introduced and several mechanisms for visualizing and interacting with the extracted geographic data are explored.

AUTOMATED STRUCTURAL AND SPATIAL COMPREHENSION OF DATA TABLES

by

Marco David Adelfio

Dissertation submitted to the Faculty of the Graduate School of the
University of Maryland, College Park in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
2015

Advisory Committee:

Professor Hanan Samet, Chair

Professor Larry S. Davis

Professor Richard Marciano

Professor David M. Mount

Professor William Rand

Professor Paul M. Torrens

Preface

The material in this dissertation is based in part on the following publications. I certify that I have made substantial contributions to the jointly authored work included in this document.

Chapter 2:

- M. D. Adelfio and H. Samet. Schema Extraction for Tabular Data on the Web. *Proceedings of the VLDB Endowment*, 6(6) Apr. 2013, pp. 421–432.

Chapter 3:

- M. D. Adelfio and H. Samet. Structured Toponym Resolution Using Combined Hierarchical Place Categories. In *Proceedings of the 7th ACM SIGSPATIAL Workshop on Geographic Information Retrieval (GIR'13)*. Orlando, FL, Nov. 2013.
- M. D. Adelfio and H. Samet. GeoWhiz: Toponym Resolution Using Common Categories. In *Proceedings of the 21th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems (SIGSPATIAL'13)*. Orlando, FL, Nov. 2013, pp. 542–545.

Chapter 4:

- M. D. Adelfio and H. Samet. Itinerary Recognition: Travelers, like Traveling Salesmen, Prefer Efficient Routes. In *Proceedings of the 8th ACM SIGSPATIAL Workshop on Geographic Information Retrieval (GIR'14)*. Dallas, TX, Nov. 2014.

Chapter 5:

- M. D. Adelfio, S. Nutanong, and H. Samet. Similarity Search on a Large Collection of Point Sets. In *Proceedings of the 19th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems (GIS'11)*. Chicago, IL, Nov. 2011, pp. 132–141.

Chapter 6:

- M. D. Adelfio and H. Samet. Automated Tabular Itinerary Visualization. In *Proceedings of the 22nd ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems (SIGSPATIAL'14)*. Dallas, TX, Nov. 2014, pp. 593–596.
- M. D. Adelfio, S. Nutanong, and H. Samet. Searching Web Documents as Location Sets. In *Proceedings of the 19th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems (GIS'11)*. Chicago, IL, Nov. 2011, pp. 489–492.

Acknowledgements

I would like to express my gratitude to the many people who have provided support and encouragement during my time at Maryland. Arriving at the graduate program with an appetite for classwork, I leave now with a thorough appreciation for the rigors of research, which I expect to shape my life and experiences no matter how far I travel from here.

I am principally thankful for the support of Professor Hanan Samet, who inspired me with the depth and breadth of his work and his persistence in getting things right. His patience as I learned how to conduct research and his encouragement as my advisor throughout my graduate career have been invaluable. I would also like to thank Professors Larry Davis, Richard Marciano, David Mount, William Rand, Paul Torrens, and Amitabh Varshney for serving on my advisory committees. Most of my time on campus has been spent in the lab, where a cadre of students and post-docs collaborated on shared projects, especially NewsStand, and proved to be wonderful colleagues. Brendan Fruin, Shen-Shyang Ho, Hao Li, Mike Lieberman, Yi Nutanong, Shangfu Peng, Gianluca Quercini, and Jagan Sankaranarayanan, I am much obliged for your friendship and support. Many others also deserve recognition for engaging with me in thought-provoking discussions.

My life at school has been immeasurably brightened by my life at home, for which I have many family and friends to whom I will be ever grateful. In particular, my parents, Lydia and Marco, were my first teachers and constant sources of guidance and enthusiasm throughout my academic career. Last, but not least, I would like to thank my daughter Delia for helping me through the final months of school and my wife Kate, for her unwavering love, friendship, and encouragement.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

There are billions of data tables on the Web, but most of them do not exist in publicly-searchable relational databases. Only a small fraction of Internet users who post information to the Web possess technical skills and consequently many structured datasets are posted in self-published tabular presentation formats, such as spreadsheets or HTML tables. In comparison to prose, graphs, or other ways of presenting data, tables have the advantage of being *data dense*, in the sense that more information can be presented in the same amount of space than prose, while at the same time preserving details about the data (such as precise values) which can be difficult to discern when a graph visualization is chosen to show trends. As a result, data tables are able to communicate clear and direct answers to a wide variety of fact-based questions. Despite this ubiquity and utility, it is difficult to search for and discover usable data tables or make use of their structure once they are found. The aim of this dissertation is to address a number of issues in the utilization of data tables, including decomposing tables into functional elements, using context to accurately interpret table contents, identifying specific classes of tables, and interacting with the collective information stored within them in useful ways.

Our work in processing data tables focuses on a particular domain that is common to a large portion of published tables: geographic data. Based on a survey of web-accessible tables, we estimate that one in four spreadsheets and one in six HTML data tables contains

names of places, making place references among the most common entity domains found in tables. Examples of geographic tables include vacation itineraries, conference venue listings, city and county demographics, voting participation statistics, and school enrollment records. The process of associating documents containing place names with the corresponding physical geographic locations has received prior attention in information retrieval settings, where it is known as *geotagging* [11, 58, 83, 84]. Several challenges arise when geotagging documents, due to ambiguities inherent to natural language usage and strings that serve multiple purposes, in some cases referring to both a place and another type of entity and in other cases referring to multiple places. Techniques that achieve high accuracy at geotagging tasks on text documents employ multiple forms of evidence when choosing interpretations for place name candidates. By comparison, the geotagging of tables relies less on techniques from natural language processing (NLP) and more on coherence between values within a table. We note that the results of geotagging data tables have the potential to be more fruitful than those of geotagging plain text, because each location is associated with an additional set of attributes that is uniform throughout the table.

One motivation for this dissertation is to address the mismatch between the utility and accessibility of data tables, particularly geographic data tables, on the Web. While these tables' contents can be thought of as a vast, heterogeneous, distributed database, our tools for exploring this database are relatively primitive. Some search sites, such as Wolfram Alpha and Google's experimental Table Search interface, provide a level of support for structured search queries and results. However, the search functionality generally allows only keyword searches and the domain of available data in the results are limited. Geographic interpretation of tables is not available except through human-guided processes. A key underlying problem is that these online data tables are not accompanied by metadata that would allow programs to easily process them and to understand their structure. Similarly, direct links between documents are not present, meaning that data found in these tables is isolated from

the rich interconnected structure into which the Web is evolving. The absence of primary and foreign keys, uniqueness constraints, column header specifications, and other standard pieces of relational metadata, collectively prevent useful querying of this "database".

To guide the data table extraction process, it is helpful to understand why explicit structural information is not present in these documents. While tabular data intended for computers (e.g., XML, RDF) is published in formats that communicate the structure explicitly, in well-specified ways that allow the data's schema to be easily accessed by algorithms [57], data intended for humans (e.g., spreadsheets, HTML tables, PDFs, DOC files) communicates structural information using implicit visual cues, such as the positioning, styling, and content of titles, column headers, and data. Consequently, there is no straightforward method for directly retrieving the structured data that human-oriented documents store. Although many data publishers recognize this and provide their data in multiple formats to allow for computer or human consumption, a large number of datasets still exist only in human-oriented formats, and thus lack the necessary metadata for querying.

Similarly, the content within each data table exhibits ambiguity due to expectations that table authors have of readers which are not fulfilled by automated systems. Specifically, references to place names can be ambiguous in multiple ways. The two primary forms of confusion are called geo/non-geo ambiguity and geo/geo ambiguity. Geo/non-geo ambiguity encompasses situations where toponyms also have common non-geographic interpretations, such as the values ["Caroline", "Marion"], which could be interpreted as a pair of given names or as neighboring small towns in Wisconsin. Geo/geo ambiguity arises in the common case where two places share the same name, as evidenced by the numerous cities named "Springfield," "Madison," or "Clinton" in the United States alone, or named "London" or "San Antonio" around the world. These ambiguities must be dealt with by any automated geotagging system. However, existing geotagging systems tend to use either hand-crafted heuristics with arbitrary thresholds or machine learning methods that require

substantial training sets of hand annotated data in order to properly geotag documents.

One of the notable transformative effects of the Internet has been its use as a personal publishing platform, where individuals can disseminate information regardless of the size of the audience to whom it is relevant. Like many other types of Web content, data tables are used to communicate content for a variety of reasons and with a variety of audiences in mind. At on one end of the spectrum are *personal tables*, such as listings of an individual's favorite bands or a weekly appointment schedules. On the opposite end are *common tables*, which are relevant to a broader audience. Common tables include data such as census data, fact tables about historical figures, or professional sports team schedules. Common tables may interest a wide audience and be published by several sources or may have an authoritative source that provides the data in a machine-readable format that could obviate the need for table extraction. Yet existing automated table extraction research has primarily been directed at public tables. In order to show the utility of personal tables, one type of table we focus on is the travel itinerary, which many Internet users post online. Without a service to identify and expose these itineraries, they remain difficult to search and browse. Our work looks at the patterns that are used in itinerary creation and which are in turn useful for itinerary detection.

The rest of this chapter is organized as follows. First, we explore characteristics of data tables and how they inform our strategies for processing them in Section 1.1. With this context, in Section 1.2, we survey previous work in the area of table processing and outline an approach for handling tables with complex structure. Section 1.3 provides background on document geotagging and introduces an approach for geotagging tables that incorporates their rich structure. Next, in Section 1.4, we discuss a method for identifying a particular class of tables: travel itineraries. We then introduce a framework for performing spatial similarity search on geographic tables in Section 1.5. Finally, we discuss visualization techniques for interacting with geographic tables in Section 1.6 and outline the remainder of the dissertation in Section 1.7.

1.1 Characteristics of Data Tables

Data tables incorporate various aspects of natural language and structured databases to express their contents. Since they are constructed by humans, creative uses of their format and deviations from any formal set of authorship rules are inevitable, which brings to mind the challenges in parsing natural language. At the same time, data tables would be less useful if we could not apply some expectation of order and coherence to the interpretation of their contents, much as we can with more formal relational database systems. Given their semi-structured nature, it is sometimes helpful to think of data tables as *natural language databases* that combine aspects of data, structure, and language.

- **Data**. Data can be defined broadly as a set of attributes that apply to a set of entities. In theory, any data can be presented in tabular form, by grouping entities that share attributes into separate tables, and linking entities with textual references. In practice, the constraints imposed by the tabular data grid format limits the types of data that can be displayed, insofar as the displayed list of attributes is shared across multiple entities and the type of attributes are limited to those that can be displayed in a grid cell.

- **Structure**. Understanding table structure is crucial for interpreting all but the simplest of tables. A header row is the most common structural table element, but complex structures are also found in many tables, including hierarchical headers, nested row groupings, and aggregation of other values, among others. The structure of a table dictates the functional role of individual table cells.

- **Language**. The contents of each cell are determined by the value of the appropriate attribute, together with the language and formatting constructs used to express it. For example, highly coupled attributes like city and country may be grouped into a single, comma-separated cell value. Specific data domains, such as dates, numbers, lists, and text, come with expectations about formatting and presentation based on the viewer's

experience with processing such values in natural language settings.

Any table processing system should account for the relationship between data, structure, and language, by specifying the domains that can be handled for each. The body of available tables that match any constraints within these areas can vary widely, and when systems only accept simple tables (e.g., with a single header row followed by data rows) with atomic, textual cell values, they greatly reduce the pool of tables available to them.

While humans process tables much differently than computers, they still require cues and context in order to understand the contents. Human table authors assume that human table readers will not look at each cell value in isolation, but instead will be able to infer properties of an individual cell from its surroundings. The cell's context provides the means for *disambiguation*, the process through which cell values are assigned specific interpretations. Geographic place names (or *toponyms*) are commonly ambiguous and can require the use of context for accurate interpretation. The toponym resolution problem, the focus of Chapter 3, involves utilizing surrounding context in order to assign geographic interpretations to place names. Toponyms, or place names, are textual references to geographic entities that are frequently used in place of spatial specifications of the same entities (e.g., latitude and longitude coordinates of point features, or region boundaries for areas), but introduce ambiguity when multiple places share the same name. Additionally, table creators employ a variety of layout and formatting patterns to communicate structural information about a table's data. Some patterns are extremely common (such as positioning column headers in the top row, or positioning the primary entity of each row in the leftmost column) while others are rare (such as repeating column headers in the bottom row) or are primarily found in tables within a specific domain (such as the divided rows found in the periodic table of elements).

To understand table composition, it is helpful to remember that table authors have many options to present their data, but in many cases tables lie at a sweet spot balancing precision and density of data. While using natural language (e.g., English) sentences to describe data

is possible, it is typically much less compact than a presentation of the same data in tabular format. On the opposite end of the spectrum are charts, graphs, and diagrams, which can include thousands of data points in a small space by sacrificing the precision of textually specified values. The desire for both compactness and precision results in table designs that require user inferences rather than use extra space to specify a strict schema specification, while also giving an incentive to table authors to "fill out" the table and include data values that might otherwise not be included.

1.2 Table Processing

Presentational data tables[1] make use of a variety of mechanisms to signal how the table is structured. In Figure 1.1, a sample of tables found in documents on the Web are shown, illustrating a variety of table styles and structures. Research on using data tables has focused on a variety of applications, including ontology generation and expansion [38, 43, 46, 52, 103], question answering [40, 80], correcting spreadsheet errors [36], allowing search over data tables [19, 20, 21, 67, 70], and data integration [16, 30, 59, 71, 74, 101, 105, 106, 113]. For an overview of table processing paradigms, refer to the surveys by Hurst [50], Zanibbi et al. [115], Embley et al. [35] and Silva et al. [98]. Table-related tasks depend on the analysis of particular table structures, including the segmentation of cells in visual media and PDF documents [12, 44], classification of tables as relational or non-relational [20, 22, 109], understanding nested table headers [53, 95, 107, 108], detecting row patterns [81], tables in the form of lists [34], recognizing table roles [27], and summarizing table contents [23]. In addition to research publications, Google has experimented with public search systems such as Google's experimental Table Search[2] and the now-deprecated Google Squared project[3].

[1] The term "presentational data tables" is used to distinguish tables found in spreadsheets, HTML pages and PDF or Microsoft Word documents from table data structures in relational databases. We generally shorten this to "data tables" or simply "tables" in this dissertation when the meaning is clear from context.

[2] https://research.google.com/tables

[3] http://googleblog.blogspot.com/2009/06/square-your-search-results-with-google.html

Interestingly, different data table formats make different sets of mechanisms available and tend to adhere to different structural and stylistic norms. Spreadsheets, in particular, contain various formatting and content generation mechanisms that are not available in other table media. One aspect unique to spreadsheets, in contrast to document formats which contain tabular elements, is that the grid of cells within a spreadsheet constitutes the entire space for communicating information. While HTML pages can contain explanatory text accompanying an HTML data table and PDF documents frequently contain figure captions that describe the contents of a table, spreadsheet authors must include explanatory text or contextual information within spreadsheet cells. This leads to ambiguity during the automated parsing of spreadsheets as data tables, because there are frequently non-data values alongside the data values. This is a primary factor contributing to the lack of attention paid to the spreadsheet parsing task in prior work. More generally, tables containing non-data elements have been overlooked in much prior work because the quantity of simpler tables has been viewed as a large enough corpus for many methods. HTML tables, meanwhile, have their own set of distinguishing characteristics. One of these is the presence of the `<th>` tag to indicate a header cell. Unfortunately, these are frequently misused or omitted so they cannot be used as reliable indicators of structure. One of the aims of this work is to increase the pool of usable tables by removing restrictions on complex table structures.

Most of the earliest references to mining presentational tables focus on *textual tables*, which are tables created by aligning values into columns using a fixed-width font. This was the predominant way of presenting tables in computer-based documents before rich text document formats became more commonly used. Early research explored several challenges raised by textual table extraction [50, 81, 115]. First, text documents without markup contain no uniform indicators for the beginning and end of tables, their presence and bounds must be decided based upon visual differences. A second challenge in this setting is segmenting cells; since whitespace is used to break both words and cell values, its presence can be ambiguous.

Table 3.
CBO's Budget Projections for Fiscal Year 1996
(In billions of dollars)

	May	August	Change
Revenues			
Individual income taxes	636	652	16
Corporate income taxes	169	170	1
Social insurance taxes	504	508	5
Excise taxes	52	54	2
Other	67	65	-2
Total	1,428	1,450	16

Mean elevations listed by from highest to lowest. (List by state, highest, lowest, or mean elevations)

Rank	State	High point	Low point	Mean elevation ▼
1.	Colorado	14,440 feet	3,315 feet	6,800 feet
2.	Wyoming	13,804 feet	3,099 feet	6,700 feet
3.	Utah	13,528 feet	2,000 feet	6,100 feet
4.	New Mexico	13,161 feet	2,842 feet	5,700 feet
5.	Nevada	13,140 feet	479 feet	5,500 feet
6.	Idaho	12,662 feet	710 feet	5,000 feet
7.	Arizona	12,633 feet	70 feet	4,100 feet
8.	Montana	12,799 feet	1,800 feet	3,400 feet
9.	Oregon	11,239 feet	Sea level	3,300 feet
10.	Hawaii	13,796 feet	Sea level	3,030 feet
11.	California	14,494 feet	-282 feet	2,900 feet
12.	Nebraska	5,424 feet	840 feet	2,600 feet
13.	South Dakota	7,242 feet	966 feet	2,200 feet
14.	Kansas	4,039 feet	679 feet	2,000 feet
15.	Alaska	20,320 feet	Sea level	1,900 feet

	A	B	C	D
1	**HAL 2012 Seattle Ship Itinerary**			
2				
3	**Amsterdam 14 Day Explorer**			
4	**Day**	**Port**	**Arrive**	**Depart**
5	Friday	Seattle		1700
6	Saturday	At Sea		
7	Sunday	Ketchikan	800	1700
8	Monday	Tracy Fjord		
9	Tuesday	Juneau	800	2200
10	Wednesday	Icy Strait Point	700	1600
11	Thursday	At Sea		
12	Friday	Anchorage	700	2300
13	Saturday	Homer	1000	1800
14	Sunday	Kodiak	700	1500
15	Monday	Hubbard Glacier		
16	Tuesday	Sitka	1000	1700
17	Wednesday	At Sea		
18	Thursday	Victoria BC	1200	2300
19	Friday	Seattle		700
20				
21	**Westerdam**			
22	**Day**	**Port**	**Arrive**	**Depart**
23	Saturday	Seattle		1600
24	Sunday	At Sea		
25	Monday	Juneau	1300	2200
26	Tuesday	Glacier Bay		

County Name	County Code	Tax Type	Resident Rate	Nonresident Rate
Adams	01	COIT/CEDIT	0.01124	0.00674
*Allen	02	COIT/CEDIT	0.0135	0.006375
Bartholomew	03	CAGIT/CEDIT	0.0125	0.005
Benton	04	CAGIT/CEDIT	0.0229	0.0054
Blackford	05	CAGIT/CEDIT	0.0136	0.0061
Boone	06	COIT	0.01	0.0025
Brown	07	CAGIT/CEDIT	0.022	0.005
Carroll	08	CAGIT/CEDIT	0.017039	0.004
Cass	09	CAGIT/CEDIT	0.025	0.005
Clark	10	CAGIT/CEDIT	0.02	0.0075
Clay	11	CAGIT	0.0225	0.0025
Clinton	12	CAGIT/CEDIT	0.02	0.0075
Crawford	13	CAGIT/CEDIT	0.01	0.005
Daviess	14	CAGIT/CEDIT	0.0175	0.0075
Dearborn	15	COIT	0.006	0.0015
Decatur	16	CAGIT/CEDIT	0.0133	0.0058
DeKalb	17	CAGIT/CEDIT	0.015	0.0075

Figure 1.1: Sample data tables in a variety of formats

HTML tables were addressed in subsequent research, initially by Chen et al. [22]. As with many approaches discussed here, theirs involves a multi-stage processing pipeline. In the first stage, a simple filter is used to weed out "non-tables", which are tables that are used for layout, forms, or menus but which do not contain data. The filtering process involves counting the total number of cells in the table and the number that contain hyperlinks, form elements, or images. A threshold is applied to each of these counts in order to classify each encountered table. In the second stage, cell values are compared to the values of neighboring cells and deemed "similar" if they match according to one or more criteria—by containing similar characters, containing similar value formats, or both containing numbers. If the total number of neighboring cells exhibiting similarity does not exceed a selected threshold, the table is regarded as a "non-table" and discarded. The final phase uses a set of heuristics to extract value-attribute pairs. In an evaluation on 918 Chinese airline website tables, the table recognition phases achieve an F-measure of 86.50%. The experiment incrementally adds similarity metrics and unsurprisingly observes the largest jump in recognition performance when neighboring numeric cells are treated as similar.

Wang and Hu [109] build on the "non-table" filtering work of Chen et al. to focus on the task of detecting "genuine" vs "non-genuine" tables. They define genuine tables as tables where "a two dimensional grid is semantically significant in conveying the logical relations among the cells". This is a crucial distinction for any system that makes use of HTML tables, since the vast majority do not contain data. Rather, tables have commonly been used as a means of controlling layout on HTML pages, aligning web forms, displaying calendars, positioning menus, and many other uses that do not present relational data. Wang and Hu define layout, content type, and word group features as input for a classifier. Both decision tree and support vector machine models were trained on a manually annotated set of tables and evaluated for classifying genuine from non-genuine tables, and the results showed a substantial improvement over a similar classifier based on heuristics.

An important early contribution to the structural analysis of tables was the introduction of a notation to describe the relationships between nested headers (for both rows and columns) and the individual attribute values that they apply to, which is now known as Wang notation after its creator [107]. The key idea of Wang notation is to capture an abstract, layout-independent view of a table's data. The presentation form of a table is the concrete grid of cell values that serves as an instantiation of the abstract table. Later efforts make use of this notation for user-guided conversion of abstract tables from concrete tables [53] and for extracting category trees among column and row headers that can be used to index individual data cells [95].

Cafarella et al. [19, 20, 21] introduced WebTables, a system for extracting data from the 14.1 billion HTML tables found by Google's general-purpose web crawler. The WebTables extraction method uses a pair of rule-based classifiers to make two decisions regarding each encountered table. First, tables are determined to be "relational" or "non-relational", using similar definitions as the genuine vs. non-genuine decision of Wang and Hu and the actual vs. non-table decision of Chen et al. Non-relational tables are discarded at this stage. Next, the first row of each relational table is classified as a "header" or not; based on a hand-marked sample, they find that 71% of true relational tables include a header in the first row. Importantly, the classifier used in the WebTables method "is tuned to give very high recall at the cost of lower precision" to allow as many tables as possible to pass through to downstream applications, which can perform additional filtering if the high quantity of non-relational tables is detrimental. This emphasis leads to an observed precision of only 41% for recognizing relational tables in an evaluation corpus.

The WebTables method's prioritization of recall over precision is well-justified in the sense that applications that make use of the extracted table data must make decisions about which tables are relevant, but cannot make such a decision if the table is judged to be non-relational and hence discarded to begin with. A contribution of this dissertation is a method

for further increasing the pool of usable tables beyond the work of WebTables. In particular, our table extraction method focuses on structural components of spreadsheets that have thus far prevented them from serving as high-quality sources of structured data for data mining and retrieval. Given that the WebTables method serves as the table extraction technique underlying several other recent research efforts [105], we expect that improvements to its capabilities will be valuable for future work.

1.3 Table Geotagging

Place names commonly occur within lists and data tables, whose authors frequently omit qualifications (such as city or state containers) for place names because they expect the meaning of individual references to be obvious from context. This results in ambiguity, since when that context is not included, the toponyms must be disambiguated solely by looking at other toponyms in the list or table. Consequently, a transformation from toponym to geographic coordinates must occur before any spatial processing is possible. By interpreting each place name as a specific geographic entity (a process known as *toponym resolution*), the document containing the list or table is *geotagged* with the locations it references. While toponym resolution and geotagging are common topics in current research, the results can be inaccurate when place names are not well-specified (that is, when place names are not followed by a geographic container, such as a country, state, or province name). One aim of this work is to utilize the context of other places named within the table or list to disambiguate place names that have multiple geographic interpretations.

Geographic references are a very common component of data tables that can occur in both (1) the case where the primary entities of a table are geographic (such as demographic tables where the entities are nations and attribute values are the populations of those nations) or (2) the case where the primary entities of a table have geographic attributes (such as a table of marathon results where the entities are the runner's names and the runners' hometowns are

listed as attribute values). In many cases, the geographic references are not well qualified (for example, when "Paris, Texas, USA" is being referred to in a list of other towns in northern Texas, it may simply be presented as "Paris"). In order to resolve the location references to their intended geographic interpretations, we must make use of the context, which in this case is composed of the other toponyms in the same table column. In this sense, the table geotagging task is differentiated from the task of geotagging place names that are found in plain-text documents, as the place names in table columns are typically more homogeneous. In particular, we expect that there is an underlying *place category* which can describe the toponyms within a single column. Examples of place categories include *"states/provinces in North America"*, *"large cities in Bavaria, Germany"*, or *"airports in Italy"*. Figure 1.2 shows an example list of toponyms and potential place categories for different geographic interpretations of those toponyms. By identifying likely place categories for toponym lists, we can reduce the ambiguity of resolving individual place names.

Our approach for table geotagging relies on a category formulation that we call *"combined hierarchical place categories"* in order to geotag tables that contain ambiguous place names with little or no qualifying context. We use a Bayesian likelihood model to assign geographic categories to toponym lists or individual table columns. This ensures coherence among the interpretations of toponyms that are expected to have a consistent theme (called *column coherence*), due to the tabular structure in which they were found. For example, assigning a coherent category to a list improves the odds of resolving "Washington" to mean "the State of Washington" when it appears in a list containing the values [Washington, Idaho, Oregon] (which are all names of American states) while resolving "Washington" to signify "Washington, DC" when it appears in a list containing the values [Washington, New York, San Francisco] (American cities).

We use a gazetteer (specifically, the GeoNames geographical database [39]) to identify possible geographic interpretations for each toponym in a table. For each entity in the

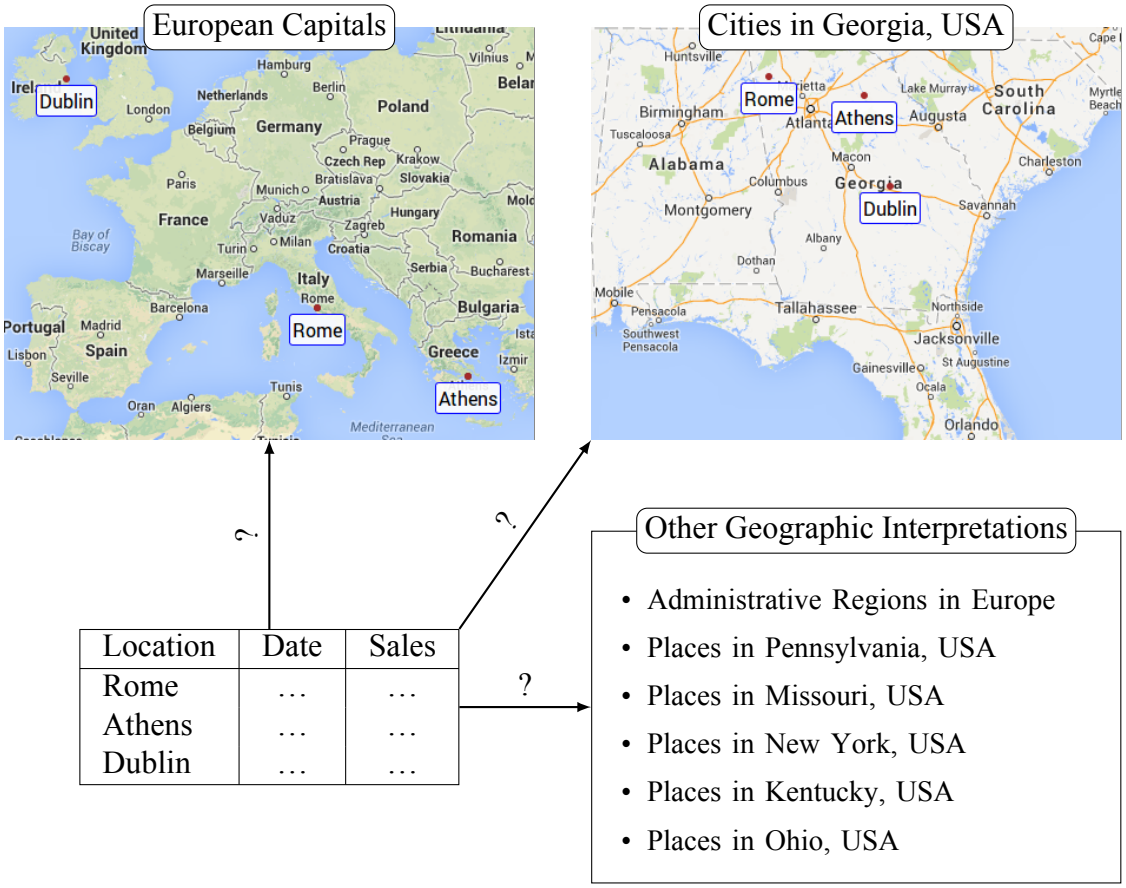


Figure 1.2: Plausible categories of geographic interpretations for a sample table. Many toponym sets, such as the one shown here, have several plausible geographic interpretations that fall in distinct place categories, whereas only one describes the true category of places that was intended by the table's author. In this case, each toponym has an interpretation as a capital city in Europe or as a city in Georgia, USA. Other, less likely categories of geographic interpretations are listed. Our method assigns likelihood estimates to each category of geographic interpretations based on features of the categories and how well it fits the toponym list.

database, the gazetteer provides several descriptive attributes, such as the place type (e.g., "capital city"), geographic containers (e.g., the country in which the interpretation is found), and the population of the place. We create what we call a *category taxonomy* that represents all possible combinations of these three attributes. The values of each attribute are encoded in a tree where the root can describe all interpretations found in the gazetteer, whereas lower levels can only describe subsets of the interpretations. Using the category taxonomy, we find all possible categories that describe an individual place name interpretation by finding each attribute of the interpretation in the attribute trees and taking the Cartesian product of all possible path prefixes of each attribute. For example, the city of Washington, DC can be described as a "place on Earth with population ≥ 0", but that is not a very useful category. It could also be described as a "city in North America with population ≥ 100", which is a slightly more discriminating category. There are many other ways to describe it within our system of categorization that lead up to the most discriminating category for Washington, DC, which is "Capital of an independent political entity, located in the District of Columbia, USA, with population $\geq 100{,}000$". The key concept is that many categories within our taxonomy can be used to describe multiple interpretations from our gazetteer, but for a given list or column of place names, there is one category that is simultaneously both discriminating and broad enough to describe an interpretation for each place name.

The Bayesian approach of our method involves measuring certain characteristics of a small sample of training data and using that data to identify characteristics that are more likely to occur in true categories for a list or table column. As a concrete example, the statistics taken from our training data show that interpreting a set of toponyms in a way that they all have populations greater than 10,000,000 is about 18% more likely to be the expected way of interpreting them, rather than interpreting them as places that all have populations greater than 1,000,000, when the population is viewed as an isolated feature. The algorithm computes several such likelihood values and combines them for an aggregate likelihood score

that a specific category of interpretations leads to the expected geotagging results.

Traditional systems that use geotagging, such as Web-a-Where [11], STEWARD [65], and NewsStand [102], along with general systems for mapping Web content [73], accept plain-text documents or web pages as the input for a geotagging algorithm. Due to various types of ambiguity that have been identified [83], geotagging plain-text documents commonly involves some level of natural language processing (NLP) to accurately identify individual toponyms and reason about the relationships between them. Various forms of evidence and inference have been applied to geotagging problems [18, 78, 87]. In particular, geotagging accuracy improves when incorporating the assumption of coherence between place names in several ways. For example, some systems attempt to infer a geographic focus of individual document sources, known as a *local lexicon* that can be used to resolve otherwise ambiguous toponyms [63, 86]. In other work, incorporating the interpretation of toponyms that appear close together in text was shown to improve toponym resolution accuracy [61]. Additionally, some work has shown that sentence structure, such as place names appearing in comma-separated groups, can be utilized to improve accuracy [66]. However, all of these methods apply fairly loose definitions of consistency because plain-text documents are unstructured and heterogeneous. Our aim is to take advantage of the presumed coherence between entities and attributes that appear in a single data table by asserting a stronger notion of consistency in our approach.

1.4 Itinerary Tables

For anyone researching travel options for an upcoming vacation in a new part of the world, the advice and experience of previous visitors can be invaluable. Travel guidebooks, travel agencies, and online resources fulfill this role in many cases, however it is often difficult to get a sense of the wide variety of travel options available in a region of interest. Map-based interfaces for browsing uploaded travel itineraries could substantially improve

travel research methods, which currently involve searching for travel suggestions using keywords, then visiting each search result to verify that it matches the geographic constraints of the travelers.

Developing such an itinerary browser and search system requires a reliable method for recognizing and extracting travel itineraries from Web-accessible documents. No prior research has focused on the specific problem of itinerary detection and retrieval from tables or text documents. While itineraries exist in a variety of formats, including plain text, we focus on detecting and extracting itineraries from spreadsheets and tables, which have a more regular structure and are more likely to contain metadata associated with each stop along the route.

The primary challenge we address is differentiating between itineraries and other geographic tables. While textual clues (i.e., the presence of the word "itinerary" in the title of a worksheet or the caption of an HTML table) can serve as useful indicators, a classification technique based only on text features would identify many false positives and fail to identify many false negatives. Additional criteria, such as whether the table includes a column of dates, may also have a strong correlation with the type of table being processed, but is still far from conclusive evidence that a table is an itinerary.

Our hypothesis is that spatial analysis is the missing feature for enabling effective itinerary detection. Specifically, for humans, determining whether or not a table contains an itinerary frequently becomes easier when the locations in the table are viewed on a map, with lines connecting consecutive locations, because a variety of real-world constraints on time, money, and fuel encourage human travel that does not include unnecessarily long or inefficient routes. Instead, maps representing true itineraries typically follow spatially *efficient* routes (a concept that we formalize in Section 4.1.2). To measure the efficiency of an itinerary, our approach makes use of an optimization technique that was originally developed to generate approximate solutions for the traveling salesman problem (TSP). This optimiza-

tion technique, known as 2-opt, functions by removing two edges from a sequential path through n points and determining whether a shorter overall path can be achieved by substituting edges with swapped endpoints [28, 68]. In Section 4.1, we restate this optimization in terms of *reasonably ordered* subpaths—subpaths that, when reversed, lead to a longer total path length—and show how the presence or absence of such subpaths is a powerful feature for determining whether a table contains an itinerary.

In addition to harnessing spatial properties of itineraries, a separate challenge is the sparsity of itineraries, as a fraction of documents on the Web, or even as a fraction of geographic tables on the Web. This level of sparsity suggests that amassing a reasonably-sized collection of itineraries will require crawling large portions of the Web. Furthermore, reliably extracting table data and assigning geographic interpretations to place names within the tables are prerequisites for accurate identification and extraction of itineraries.

Our work on itinerary detection complements other work in information retrieval that seeks to expose geographically rich content. While we primarily focus on the spatial, rather than temporal, aspects of itineraries, research on document-based spatio-temporal extractors addresses some related tasks. Strötgen et al. [99] described a system for extracting ⟨time, location⟩ pairs from unstructured text documents, to support browsing the documents as trajectories (for example, following the path of explorers as described on their Wikipedia pages). A trajectory browser displays the extracted trajectories on a map and provides relevant text snippets for selected stops. The emphasis of this work is on building accurate spatial and temporal profiles of targeted documents, so it does not address ways of identifying which documents contain trajectories. Spatio-temporal extraction systems also exist for a variety of other source documents, such as RSS feeds [72].

Systems for inferring itineraries from metadata, rather than documents, have been developed on top of various data sources, including geotagged photo streams [31] and GPS tracks [114]. These efforts have a rather different focus than ours, stemming from the fact

that the locations in these efforts are specified numerically (e.g., as latitude / longitude pairs) rather than textually, and are presumed to be personal itineraries based on the nature of the data source. Yoon et al. [114] use GPS logs to identify stops (called "stay points") and transitions between clusters of stops, which allow them to recommend itineraries based on time constraints and the popularity of the stops. The periodic and granular location information provided by GPS tracks make them a valuable source for itinerary data. However, capturing this data requires that users upload large quantities of GPS tracks to the system or to a public location, so privacy concerns may hinder its accessibility. Additionally, extracting segments of GPS tracks that are relevant as itineraries requires addressing a whole other set of challenges.

1.5 Point Set Queries

Existing spatial search systems (e.g., Google Maps, Yelp, Zillow) allow users to find points of interest by using a proximity query, such as "find the 3 nearest gas stations with respect to one's location $\boldsymbol{q}$". This problem can be formalized as the nearest neighbor (NN) query problem. That is, given a set $\mathcal{D}$ of gas stations and a query location $\boldsymbol{q}$, the nearest neighbor (NN) query [47, 91] identifies a point $\boldsymbol{p}$ in $\mathcal{D}$ which minimizes the distance $\textsc{Dist}(\boldsymbol{q}, \boldsymbol{p})$.

The NN query can also be generalized to the *aggregate NN query* [79] and the *distance join query* [25, 48, 97, 112]. The aggregate NN query finds the point in a point set that minimizes the aggregate distances to all points in the query set, while the distance join query finds the point in a point set that is closest to any of the points in the query set. The queries are similar in that they both involve identifying a closest point $\boldsymbol{p}$ in $\mathcal{D}$ that minimizes the distance to a query object Q which is a set of locations rather than a single location. For example, given a set Q of stations of a train line and a set $\mathcal{D}$ of bus stops of a bus route, find a stop $\boldsymbol{p}$ in $\mathcal{D}$ which minimizes the distance to any train station in Q for transfer purposes. Both queries can also be extended to top-k variants where multiple results are returned, ranked by

the appropriate distance measure.

To support similarity search between extracted point sets, we extend the concept of NN search further, to the case where (i) the dataset $\mathcal{D}$ comprises sets of locations, (ii) the query object Q is also a set of locations, and (iii) we want to find the k most similar sets in $\mathcal{D}$ with respect to Q using measures described later in this section. Example applications that may benefit from this method include:

- Given a geographical distribution of a current disease outbreak represented as a location set Q, an epidemiologist may wish to find k occurrences of outbreaks (from a set $\mathcal{D}$ of historical outbreak distributions) that are most similar to Q. These results can then be used to help identify correlations between the outbreak in question and other outbreaks.
- Let Q denote a location set of warehouses of one logistics company and $\mathcal{D}$ denote a collection of location sets of gas stations where each location set contains locations of gas stations owned by a specific gas company. To form a partnership with a gas company, the logistics company may wish to find the gas company whose location set S minimizes the average distance from each warehouse in Q to the nearest station in S.

The similarity between point sets can be measured using the Hausdorff distance, a measure that is commonly employed in spatial and geometric matching problems in a variety of contexts, such as shape- or image-matching, geometric modeling, model rendering, and image recognition [9, 41, 76, 100]. In these contexts, the Hausdorff distance is used to measure how well two shapes, images, or polygonal meshes resemble each other (i.e., A matches B within a maximum discrepancy of δ). The problem of computing Hausdorff distances between pairs of point sets, polygons, or meshes is a well-studied problem. For two point sets with total cardinality $O(n)$, the naive approach computes all pairwise distances to find the MaxMin distance of the two sets, which has a running time of $O(n^2)$. Some approaches to achieve more efficient performance involve calculating the Voronoi diagram of one point set,

and then performing plane sweep [10]. In applications where object locations are not fixed such as image/shape matching, a more robust use of the Hausdorff distance involves the calculation of the minimum Hausdorff distance under rotation and translation [9, 51]. These approaches become computationally intractable in high dimensions [100]. Some methods attempt to improve efficiency by introducing randomization [10] or providing approximate solutions [100].

Computing the Hausdorff distance is naturally related to executing a nearest neighbor query, as the Hausdorff distance from point set A to point set B is determined by the maximum distance of a point in A to its nearest neighbor in B. Thus it is not surprising that an approach similar to the one used for solving the k-nearest neighbors (kNN) problem can also be useful in calculating the Hausdorff distance between two point sets. Nutanong et al. [76] proposed an algorithm for computing the Hausdorff distance in the context of trajectory matching, which is based on the branch-and-bound approach used for solving the k-nearest neighbors (kNN) problem. The branch-and-bound search method involves incrementally proceeding through a search tree, and re-ranking and pruning the candidate solutions as the process continues. Tang et al. [100] discuss the difficulties of computing the exact Hausdorff distance between polygons efficiently in $\mathbb{R}^3$, and present an approximation algorithm that uses a similar branch-and-bound technique that stops when the bounds are within the specified approximation factor.

In order to support similarity search queries, our goal is not one of finding the most efficient method of computing the Hausdorff distance, but instead one of reducing the number of times that the Hausdorff distance is computed. This is done by improving Hausdorff distance estimates used in a branch-and-bound search to provide a greater pruning capability.

The branch-and-bound principle is widely adopted for similarity search problems [49]. Classic examples of branch-and-bound search in spatial databases are the depth-first [91] and best-first [47] algorithms to search for nearest neighbors (NNs) over a point set indexed in a

hierarchical index, such as the R-Tree [14, 42]. These algorithms use an optimistic estimator to provide the order in which index nodes are visited and to disregard index nodes containing points that clearly cannot be resultant NNs.

For example, the best-first NN algorithm uses a priority queue to sort index nodes N according to the minimum distance MINDIST from the query point $\boldsymbol{q}$, which serves as an optimistic estimate of the distance from $\boldsymbol{q}$ to any object in N. In this way, index nodes with large MINDISTS are scheduled to be visited later than those with smaller MINDISTS. When a data point is retrieved from the priority queue, the MINDIST estimator guarantees that none of the nodes currently in the priority queue can produce an object closer to $\boldsymbol{q}$. As a result, the search can be used to incrementally find NNs and terminate when a desired number k of data points are retrieved from the priority queue.

The best-first search principle can also be used to process aggregate NN queries [79], which are multiple query point generalizations of the NN query. Specifically, given a dataset $\mathcal{D}$, the aggregate NN of a query point set Q is the data object $\boldsymbol{p}$ in $\mathcal{D}$, which minimizes the distance to Q according to an aggregate function: MIN, MAX, or SUM. One can calculate an optimistic estimate as the smallest possible aggregate distance of any data point in the node N to Q. For example, an optimistic estimate of MAX-aggregate from Q to objects in a node N is given as $\text{MAX}\{\text{MINDIST}(\boldsymbol{q}, N) : \boldsymbol{q} \in Q\}$. A best-first search can then be conducted by visiting nodes N in ascending order of the optimistic estimates. In the next subsection, we show how the same concept can be applied to similarity search over a collection of point sets.

1.6 Tabular Data Visualization

Geographic data tables can be useful as standalone tables, but as with most geographic systems, a spatial visualization can quickly boost understanding of the data. We explored three separate tabular visualization problems, along with the ways in which table context can be exploited for this purpose.

The first domain we analyze are itinerary visualizations. Due to the tabular sources from which the itineraries are extracted, we can obtain a listing of stops, but unlike GPS routes or trajectory visualizations, we do not have knowledge of intermediate points or the paths taken between those stops. In fact, we argue that these are not as important to communicate within the framework of an itinerary diagram as the set of stops, their names, and their connectivity. Our goal is to overlay an itinerary visualization over a map in a way that allows viewers to easily discern the ordering and location of points on the itinerary. The process of creating an itinerary visualization is equal parts graph drawing and map labeling, with additional constraints and optimization criteria that differentiate it from both. A fundamental difference from many graph-drawing scenarios is that the locations of the waypoints are fixed to correspond to their locations on a map, whereas graph-drawing techniques typically allow for moving nodes to optimize the layout. Another difference is that we allow for the usage of curved edges for itineraries, which is uncommon in graph-drawing contexts. Unfortunately, presenting itineraries from these datasets requires a more useful visualization than a simple map mash-up, which does not communicate the connectivity or the sequence of visited locations.

Automated itinerary layout bears a resemblance to several existing problem domains in computer science, including the well-studied areas of graph drawing and map labeling and the more specific sub-area of route rendering. The canonical graph drawing problem [13, 37] takes as input a graph $G = (V, E)$ (with vertices V and edges E) and seeks to find an appropriate visual representation of the graph, usually by assigning each vertex $v \in V$ coordinates in the 2D plane. While finding an optimal visual representation of a graph is a somewhat subjective task, there are common criteria that are used to approximate visual and aesthetic priorities that are emphasized in manual graph drawing. Some of the primary considerations are to have an even distribution of nodes and edges, to use uniform or near-uniform edge lengths, to display isomorphic substructures uniformly, and to minimize the

number of edge crossings [45].

A visualization's utility depends on the amount of information it communicates to viewers. Usability studies have evaluated user preferences for graph layouts that emphasize specific priorities. Purchase [82] concluded that "reducing the crossings is by far the most important aesthetic, while minimizing the number of bends and maximizing symmetry have a lesser effect." This is especially relevant in the context of the itinerary layout problem since translating waypoints is not possible, meaning the only avenue for reducing edge crossings is by adding bends (i.e., curvature) to the edges.

One approach to drawing general graphs is to use a force-directed layout in which node and edge pairs are assigned attractive and repulsive forces and a physical simulation is performed to obtain an appropriate layout [37]. Another uses simulated annealing, whereby layouts are iteratively chosen to either improve upon a previous layout or with some probability that approaches zero as the process continues [55]. Simulated annealing is designed to model the physical process of heating a material, then cooling it until it reaches a stable equilibrium state.

Several efforts have looked at automated map layouts for specialized purposes. Route map generalization [8] combines aspects of both graph drawing and map labeling and aims to improve the usability of computer-generated route maps by applying cartographic principles to their design. The focus of these maps is to communicate information that is helpful for navigating between places. LineDrive [8] uses simulated annealing to optimize the route representation, based on a variety of cartographic criteria. Similar efforts have been made with bike maps [88]. More general "origin-destination" maps, including the bicycle flow maps [111] strive to show dense connectivity between nodes overlaid on a map, using asymmetric Bézier curves to emphasize imbalances in bike traffic between two stations. Unlike in itinerary visualization, these maps do not attempt to avoid overlaps between edges or to provide labels for important locations.

The second visualization setting that we explore is for point set similarity search. Many existing online map services allow users to interact with locational data in an intuitive manner. For example, users can zoom into an area of interest and issue a query to display primary schools in the area. The system then displays results ranked according to the distance from the center of the zoomed area. This type of system works well with cases where each data entry can be represented as a single location (e.g., a school or a gas station). However, for cases where each data entry may comprise multiple locations, a single query point or a simple range query may be insufficient to fully specify a search query. For example, a search for disease outbreaks that spread geographically in a similar manner to a new outbreak requires entering a set Q of locations from the new outbreak and comparing Q to a database of previous outbreaks. There are many other circumstances where doing a full *"point set to point set"* query fits the problem definition most closely. To accommodate queries such as these, we developed a geographic search system called GeoXLS, based on the point set similarity search methods outlined in Section 1.5.

The third type of visualization we present is called the *parallel detail view*. The view turns a geographic table into a collection of satellite images of the table's locations, which allows inspection of common visible attributes of each geographic location. For example, this type of view could be applied to a table of Major League Baseball stadiums, state houses, or islands in Boston Harbor. The resulting visualization is composed of a grid of static satellite images, one centered on each location listed in the table. In contrast to typical map-based visualizations of geographic location sets, which enable analysis of the spatial distribution of the locations, the parallel detail view promotes comparisons between the local geographic features of each location. Users can draw conclusions about the orientation of baseball stadiums, the relative sizes of state houses, and the range of human development present on islands. One challenge addressed in the development of this visualization was the need for precise geotagging. Additionally, different types of geographic features are visible at differ-

ent zoom levels, so consideration must be given to the relationship between zoom level and feature type.

1.7 Organization of the Dissertation

The remainder of this dissertation is organized as follows. First, we explore table structures and patterns of row organization in Chapter 2. Chapter 3 details a method for geotagging tables by making use of the consistency we expect of entities found in a single table column. In Chapter 4, we make use of the geotagging results to identify a particular class of geographic table, the travel itinerary. A novel method of point set similarity search is explored in Chapter 5, with applications for large collections of point sets that result from assigning locations to table rows. Chapter 6 demonstrates several visualization methods for tabular geographic data, while Chapter 7 provides conclusions and directions for future work.

Chapter 2

Structured Extraction of Tabular Data

Existing systems for extracting structure and information from data tables, such as the well-known technique of Google's WebTables [20, 21], generally rely on the high quantity of data tables that have simple structure, in order to avoid dealing with those data tables with more complex structure. We say a table has *simple structure* if it consists of one row of header values, followed by one or more rows containing data values. The values in the header row describe the domain of values in the data rows beneath it, while the data rows contain tuples that roughly correspond to rows in a relational database. Many tables exhibit simple structure, which may justify the focus placed on them by WebTables and others. However, many tables are more complex. As one example of a more complex structure, tables that contain geographic data about cities are commonly organized into groups by state/province. These groups may be designated in several ways, but one popular technique is to insert a *group header row* containing the name of the state/province as a single value, directly above the data rows that it includes. This row is not a data row like the others in the table. Our approach involves classifying each row based on its function to guide our extraction process. The row functions that we consider are given in Section 2.3.1. In addition to achieving high accuracy rates when classifying the function of individual rows, our focus on handling complex structure results in very high accuracy rates for correctly interpreting full tables.

Our method utilizes visual attributes of table cells as input for a trained classifier based

on conditional random fields [56], a graphical model that can be used for sequence labeling tasks. The classification procedure involves extracting attributes of individual cells, combining them using a novel logarithmic binning method that we introduce, and processing them with the classifier. The classifier's output allows us to discern between relational and non-relational tables, and furthermore to identify the structural composition of the relational ones — that is, to classify the constituent rows of each relational table as either header rows, data rows, non-relational metadata rows, or rows of a few other types that we have identified. As our experimental evaluation reveals, we improve on the commonly-used WebTables extraction method by accounting for more complex table structure so that any row can be recognized as a header row. We perform tests on a new corpus of spreadsheet tables and HTML tables that we crawled from the Web, along with an existing corpus collected by other researchers [67]. In tests on both our new corpus and the existing corpus, the ability of our method to correctly interpret full tables without error is particularly apparent when compared to alternatives.

This chapter includes the following contributions. First, we define a set of row classes that encompasses a broader range of row functions than previous work, enabling more complete post-processing techniques for using the data in other applications. Second, we apply a novel logarithmic binning scheme to encode collections of individual cell attributes as row features and show that this performs better than techniques presented in previous work [20, 81]. Third, in addition to HTML tables, we apply table extraction techniques to spreadsheets, which we believe are a more challenging, but potentially more rewarding, source of tabular data. Fourth, our method is self-contained and does not depend on outside knowledge bases or lookup tables as some prior methods have [20]. Finally, our evaluation shows substantial improvements over alternative methods, especially when measuring our method's ability to correctly interpret full tables (that is, correctly classifying every row in a table with no errors). We believe that these aspects of our approach, in combination, provide the best

available technique for extracting structured data from the enormous collection of available data tables.

The rest of this chapter is organized as follows. Section 2.1 provides background on table processing. Section 2.2 surveys prior work and describes our chosen classification method, conditional random fields. Section 2.3 presents our schema extraction method while Section 2.4 discusses common table patterns and their relevance for table extraction accuracy. Section 2.5 provides an evaluation of our method's application to spreadsheets and HTML tables. Section 2.6 presents a description of properties for describing table columns. Finally, Section 2.7 highlights the benefits of our approach and adds concluding remarks.

2.1 Extraction Preliminaries

Documents in a variety of formats can contain one or more tables. These typically correspond to individual worksheets of a spreadsheet or individual HTML tables in an HTML page. Each relational data table has a *schema*, which, in our context, consists of attribute names, values, and types, where attribute names are column titles, attribute types are the types of values in the column, and attribute values correspond to data values in the column's cells. Column names are stored in a special row or rows, usually near the head of the table, called *header rows*, while the data is stored in rows referred to as *data rows*. The data table may also contain descriptions of data, which we refer to as *metadata*. Metadata appears for several reasons, such as table titles, notes about the original source of the data, and aggregations such as column subtotals. Some metadata rows are valuable in interpreting the table's data, such as when table data is grouped and a row with a single value serves as a group header, describing a category for the following rows. The goal of our schema extraction process is to identify valid data tables along with their associated schemas and data values, which can be subsequently stored in a relational database or processed for a custom application.

Both spreadsheets and HTML pages can contain tabular data. While spreadsheets con-

sist entirely of data tables, HTML pages can contain additional content, as well as data tables in the form of *HTML tables*. HTML tables are defined by the `<table>` tag, and consist of a collection of rows and cells, denoted by `<tr>` and `<th>/<td>` tags, respectively. HTML tables have similar layout options to spreadsheets, including merged cells (through the use of `rowspan` and `colspan` attributes), header positioning, and metadata row inclusion. In addition, unlike spreadsheets, HTML tables support the inclusion of some additional structure relevant to our task, namely the `<thead>` and `<th>` tags, which serve to mark header regions and cells in tables. However, while the presence of these tags is a useful feature in determining the type of rows they occur in, they are often used for controlling layout and style, rather than for defining relational header rows in HTML tables. We also found that many HTML tables do not make use of these tags, as was also observed in prior work [80, 115]. An added complexity when processing HTML tables is the determination of which visual styles apply to the text in each cell. Our system for importing HTML documents incorporates Cascading Style Sheets to address this, including inline, embedded, and external style sheets, when evaluating the visual attributes of table cells. However, we observed that the external style sheets only yield modest improvements in accuracy, at the cost of slower processing speeds due to the increased number of resources that must be downloaded for each document.

Another consideration that must be made is how to handle tables that are not a simple n x m grid of table cells. Merged table cells (via the `colspan` and `rowspan` HTML attributes or a spreadsheet's *merge cells* feature) and nested HTML tables are two ways that this situation arises. Tables with either of these structures are accepted by our implementation. Our classifier includes features indicating the presence of merged cells, and after classification takes place, multiple copies of merged cell values are substituted for the original cell in header and data rows, so that each cell contains an appropriate string. Finally, unlike some previous work [109], we attempt to extract schemas from all HTML tables, including nested tables (i.e., not just leaf tables).

2.2 Related Work

Many techniques for processing data tables have been presented (e.g., [22, 38, 52, 64, 70, 81, 105, 109]. See [35] for a survey). The initial challenge of processing tables is to determine whether a given table contains usable data that is in a suitable format for extraction. In database terms, we call such tables *relational* and other tables *non-relational*. Relational tables are defined by the relational model introduced by Codd [24], wherein a relational table consists of a set of attributes (typically found in a header row, such as "Name, Gender, Age") and a collection of one or more "tuples" (represented as data rows where each cell value is a member of its column's attribute domain, such as "John, Male, 30"). Tables containing relational data can also include values that do not fit into the relational model (i.e., non-relational values), such as a title, blank rows, aggregations (e.g., subtotals), and more. In this chapter, we regard these tables as relational, whereas most prior work has treated these tables as non-relational. In our view, non-relational values that serve to supplement a relational table should not disqualify the table from being relational. As an example, we do not want to discard tables in which the first row contains a table title as non-relational. However, there are also many tabular grids that are created for purposes other than displaying relational data, such as HTML tables used for page layout and spreadsheets used as forms. These table types do not contain relational data that can be processed using relational operators such as projection or selection, so they should be discarded.

Several techniques focus on the relational/non-relational decision problem. Chen, Tsai, and Tsai [22] count the number of cells that are similar to their neighbors in terms of text length and datatypes, and use thresholds on these values to perform relational/non-relational classification on a set of tables from airline web sites. Instead of using a rule-based approach, Wang and Hu [109] treat the task of identifying relational data tables as a machine learning classification problem. They develop several features based on the layout of an HTML table and the distributions of cell lengths and content types, which serve as inputs to decision tree

and support vector machine classifiers. In other work, relational tables are alternatively called "genuine tables" [109] or tables containing "true relations" [21].

Other methods go further and attempt to categorize the data found within data tables based on its function. The well-known approach of WebTables [19, 20, 21] builds upon prior work by using a machine learning classifier to distinguish between relational and non-relational HTML tables on the Web. The classifier is tuned for recall over precision, counting on subsequent processing to filter out non-relational table values. For relational tables, it attempts to extract any embedded metadata such as the labels and types of the columns, thereby learning the schema for the table. However, in WebTables, only the first row of a table is considered a candidate for the table header. In essence, this is due to the simple table structure assumed by WebTables and derived methods, including the work of Limaye et al. [67], which extracts relations from HTML tables, but depends on HTML formatting methods to indicate header cells. As we show in our evaluation, such an assumption discards a significant number of tables that have more complex structures. Our approach is designed to handle such complexities, thereby increasing the number of tables available for systems that process them. Some work on table extraction also relies on external knowledge sources, such as YAGO [67] or custom fact databases extracted from text on the Web [105]. Our approach does not include such data, thereby allowing it to run accurately even on tables with contents that are not so globally relevant that they appear in knowledge bases.

Systems that rely on techniques like these are proliferating. Such systems use data extracted from data tables for different purposes, from search engines over structured data documents [80], to database augmentation using "facts" garnered from web tables [113], to automated ontology construction [52]. These systems handle tables with simple structure only, and consequently exclude large quantities of valuable table data. The approach presented in this chapter is more comprehensive, and would provide larger and cleaner sets of data tables for applications such as these, or more general attempts at schema matching across

a heterogeneous collection of data tables from the web, such as those introduced by Bernstein et al. [16].

2.3 Table Row Classification

2.3.1 Row Classes

The vast majority of table rows serve functions that can be divided into a small set of *row classes* (each identified by a single-letter *row label*), which we describe here. Each represents the classification for individual rows of a data table, as defined in Table 2.1. A minimal fraction of rows we observed (less than .01%) did not fit cleanly into one of these classes, such as rows containing notes beside cells that otherwise contain data values. In these cases, we assign the row label of the dominant class of the row.

Table 2.1: Row labels and classes.

Label	Description
H	Header rows contain cell values that describe the values contained in the subsequent data rows of that column.
D	Data rows contain data records (or *tuples* in relational parlance).
T	Title rows describe the entire data collection found in the data table.
G	Group header rows provide categories for subsequent rows, for example a table containing demographic data about cities may be grouped by country.
A	Aggregate rows contain (typically numeric) summaries of preceding rows, such as totals/subtotals.
N	Non-relational metadata, such as a note, clarification, or any text that does not contribute data or structure to the data table.
B	Blank rows contain only empty cells.

2.3.2 Features

As with many machine learning tasks, a large factor in our classifier's accuracy is the quality of the input features. A difficulty with the current formulation of our extraction problem is that we desire a label for each *row*, yet each row consists of constituent *cells*, which can exhibit differing sets of attributes. Our approach to feature selection involves extracting a collection of attributes for individual cells, then combining attributes from all cells in the row using a novel binning scheme, in order to construct a set of row features. We first describe the cell attributes, followed by our method for combining them into row features.

2.3.3 Cell Attributes

We use a large battery of individual cell attributes as the basis for our features. These cell attributes fall into the following three broad categories, with specifics of each feature given at the end of this subsection.

- **Layout attributes.** Header rows often contain merged table cells with centered text, whereas data rows rarely contain merged cells and are typically right- or left-justified, so we make sure to include these properties as cell attributes. Title rows tend to appear among the first non-blank rows of a table, and often contain values in only a few of the columns. Group header rows serve as a divider between groups of data rows, and typically contain only one or two non-blank cells. Additionally, for HTML tables, we incorporate the effects of tags like `<thead>` and `<th>` tag on the alignment of text within their cells (centered by default).

- **Style attributes.** Various font styles are more common in header rows or title rows than data rows, such as bold, italic, or underlined text. Differences in text and background colors and variations in date and number formats can also be used to differentiate be-

tween rows of distinct classes, so such attributes are extracted in our method.

- **Value attributes.** Header rows often contain relatively short textual values, rather than numbers or dates. While data rows may contain many empty cells, header rows typically contain few to zero blank cells across the full width of the table. Aggregate rows often indicate that the row contains a total or subtotal, so we include an attribute to match the case insensitive string `total`. All of these traits are extracted from data tables and used as cell attributes.

- **Similarity attributes.** Similarity attributes are formed as conjunctions of attributes of neighboring cells. For each cell c, neighboring cell c' (within the same column), and attribute α, we compute two similarity attributes for c, $c_{\alpha,A}$ and $c_{\alpha,B}$. We define $c_{\alpha,A} = c_\alpha \wedge c'_\alpha$ and $c_{\alpha,B} = c_\alpha \wedge \neg c'_\alpha$. These similarity attributes express similarities and differences between cells in neighboring rows, which are good indicators for row classes. (Note that for non-Boolean attribute α, $c_{\alpha,A} = 1$ iff $c_\alpha = c'_\alpha$ and $c_{\alpha,B}$ is not defined).

The full list of individual cell attributes is given below. Boolean attributes are prefixed with "Is" and end with "?". Other attributes take discrete textual values. The short text and long text attributes are given to text cells that contain text shorter or longer than one standard deviation from mean cell text length in a sample of data tables. Similarity attributes are conjunctions of individual cell attributes across multiple rows and are not listed below.

2.3.4 Logarithmic Binning

As with any machine learning-based approach, the goal of the training process is to generalize the training data so that new, previously unseen data can be correctly classified. In the case of classifying rows in data tables, we encounter additional challenges to achieve high generalization of our classifier. As an example, upon encountering a 10-column spreadsheet where one of the rows contains 7 columns with centered text, a classifier should make

Table 2.2: Cell attributes by type.

Layout	Style	Value
IsMerged	IsBold?	IsEmpty?
Alignment	IsItalic?	IsText?
	IsUnderlined?	IsNumeric?
	IsColored?	IsDate?
	Font	IsShortText?
	Format	IsLongText?
		IsTotal?

an informed decision on how to label this row based on rows from a training set that share similar characteristics. In this case, it is important that other 10-column rows with 7 centered columns share a feature with this row, but we may also want an 11-column row with 8 centered columns to help inform the classifier's decision. Classifiers achieve highest accuracy when fed discriminative features, so we must decide on a feature encoding that properly respects these similarities.

Table 2.3: Example feature encodings for raw feature value "3 of 11 row cells are numeric". The linear encoding method uses real-valued features, while the other methods use Boolean features.

Encoding	Feature Name	Value
Linear	"Percent numeric"	0.27
Threshold	"< 50% of cells are numeric"	1
Direct	"3 of 11 cells are numeric"	1
Logarithmic	"2 to 3 cells are numeric of 8 to 15 cells"	1

One way to encode cell attributes into row features is with a *linear encoding*. That is, the percentage of cells in a row that share a cell attribute is used as a real-valued (i.e., non-Boolean) feature, with a value between 0 and 1. Linear feature encodings are used in several previous approaches to data table extraction, including the "CRF Continuous" method described by Pinto et al. [81]. However, using percentages for feature values can have some drawbacks. Indeed, the distribution of row classes is unlikely to change linearly in step with

the fraction of row cells that exhibit a specific attribute.

An alternative is to use a *threshold encoding*, which converts these fractional values into Boolean features. This encoding is used in the "CRF Binary" method presented by Pinto et al. [81]. Specifically, their system sets thresholds for various percentage features, such as an "Alphabet Characters" feature which is considered true if more than 60% of characters in a line are letters from the alphabet. The choice of threshold values for their system appears to be ad hoc for each feature, not systematic. More importantly, a single percentage-based threshold ignores differences between uniform and non-uniform rows.

Both the linear and threshold encodings can be problematic when used universally across tables with varying widths. "Narrow" tables with few columns exhibit different characteristics from "wide" tables with many columns, and should be treated differently when encoding features. That is, the cardinality of the columns exhibiting a specific attribute and the total number of columns in a row both matter, not just their ratio. For example, when 33% of a row's cells share an attribute in a 3-column table, the distribution of row labels can differ greatly from rows where 33% of a row's cells share the same attribute within a 15-column table.

To account for the drawbacks of linear and threshold encodings, we could use a *direct encoding*, which assigns a unique feature to each unique combination of (c, r), where c is the number of cells exhibiting an attribute and r is the number of cells in the row. However, this encoding greatly increases the number of possible row features and thus reduces the chances of each feature occurring in the tables of the training set, which will significantly reduce row classification accuracy, especially on wide tables.

We address the drawbacks of linear, threshold, and direct encodings by introducing a new "binning" scheme. A binning scheme partitions the space of possible raw feature values into bins, where each bin is assigned a representative feature value. The objectives of our binning scheme are given here.

- **Differentiate Between Table Widths.** Since tables with different widths generate different distributions of row labels, their features should be divided into separate bins.
- **Aggregate Wide Tables.** Tables with 2–6 columns are much more prevalent than wider tables. To account for the sparsity of some features on wider tables, bins for wider tables should also be larger.
- **Highlight Uniform Rows.** Rows where all cells either lack or share an attribute α should be in separate bins from non-uniform rows. Also, nearly-uniform rows should be in separate bins from more heterogeneous rows.

These goals led to the encoding we present here, which we call *logarithmic binning*. The objective of logarithmic binning is to find an appropriate definition of "similar" in the table classification context. In the end, we say that two rows R_x and R_y are similar with respect to a certain cell attribute α if the logarithms of their widths are equal and the logarithms of the number of cells exhibiting or lacking attribute α are equal (all logarithms are base 2). This is implemented by assigning each row a representative feature for each attribute, based on these two log values. Formally, for row R_i of length r in which c cells exhibit a specific cell attribute α, we assign feature "$R^\alpha = (a, b)$" to R_i ((a, b) is denoted as its *bin*), where a and b are computed as follows.

$$a = \begin{cases} 0, & \text{if } c = 0 \\ \lfloor \log_2(c) + 1 \rfloor, & \text{if } 0 < c \leq r/2 \\ \lfloor \log_2(r - c) + 1 \rfloor^-, & \text{if } r/2 < c < r \\ 0^-, & \text{if } c = r \end{cases} \tag{2.1}$$

$$b = \lfloor \log_2(r) \rfloor \tag{2.2}$$

The superscript minus sign in some values of a represents that it is computed based on $r - c$ rather than c.

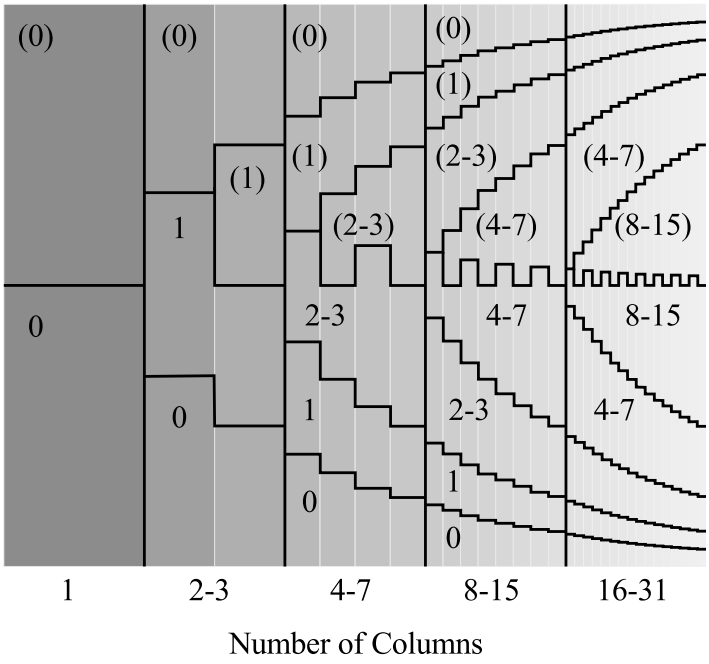


Figure 2.1: Graphical representation of bin boundaries for the logarithmic binning encoding. For a given attribute α, two rows share a bin (and thus, have a common feature value) if the logarithm of their r values are equal and the logarithm of their c values are equal (or $r - c$ values, if $c > r/2$).

Figure 2.1 provides a visual representation of the bin layout induced by logarithmic binning. In particular, it gives a visualization of the region of combinations of c and r values that are represented by each bin. The horizontal axis represents r and the vertical axis represents c/r. Horizontal partitions are made based on the logarithm of the number of columns in a row, while vertical partitions are made based on the number of cells within a row that exhibit (or lack) an attribute, given by the definition of a above. So R_1 from the previous example would fall into the bin for rows containing 4 to 7 cells ($b = 2$), of which 2 to 3 cells exhibit a specific attribute ($a = 2$). R_2 would also fall into this bin, and thus the two rows share a feature under this binning scheme.

In the table extraction setting, the goal of logarithmic binning is to generalize the feature distributions better than the other encodings, and our experimental evaluation confirms that

the schema extraction process benefits from its use.

2.3.5 Classifying with Conditional Random Fields

After collecting the feature set for each row in a table, we can assign a row label to each row using a CRF-based classifier. CRF's are a popular method for classifying sequences in various domains such as natural language processing and computational biology.

More formally, conditional random fields (CRFs) are undirected graphical models introduced by Lafferty et al. [56] that can serve as classifiers for sequence labeling tasks. Frequently used for natural language processing, such as part-of-speech tagging, CRFs have become a popular technique showing accuracy improvements over Hidden Markov Models (HMMs) and Maximum Entropy Models (MEMs) in many scenarios [56, 81].

CRFs are designed to maximize the probability of a sequence of labels $\boldsymbol{Y}$, given an observation sequence $\boldsymbol{X}$. The estimated joint probability is defined to be:

$$P(\boldsymbol{Y}|\boldsymbol{X}) = \frac{1}{Z(\boldsymbol{X})} \exp\left(\sum_j \lambda_j f_j(\boldsymbol{Y}_{i-1}, \boldsymbol{Y}_i, \boldsymbol{X}, i) + \sum_k \mu_k g_k(\boldsymbol{Y}_i, \boldsymbol{X}, i)\right).$$

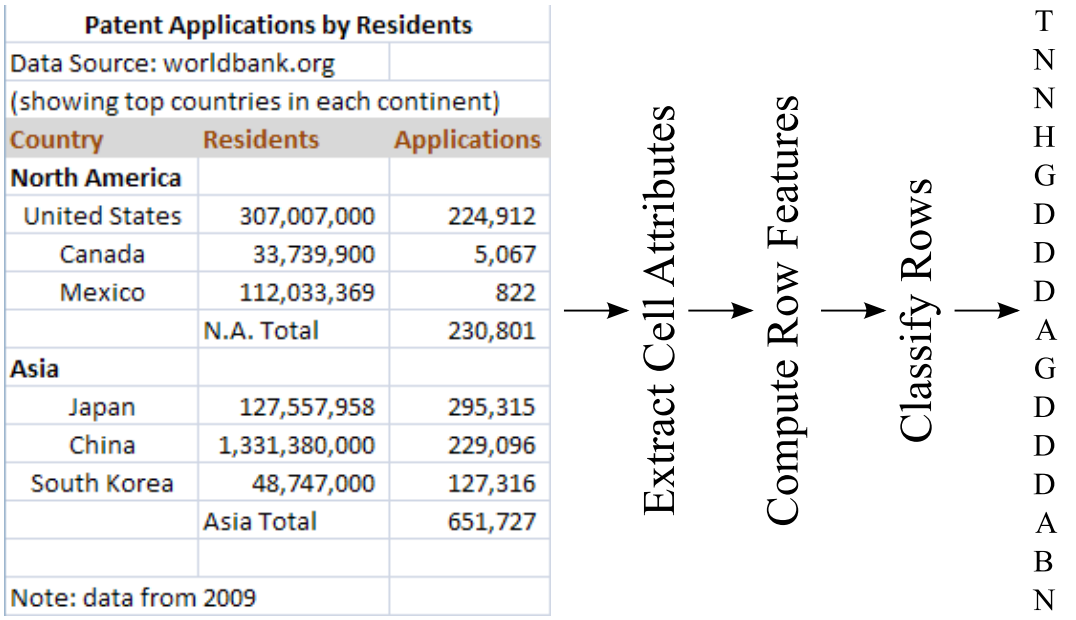
Here i is an index for the sequences $\boldsymbol{X}$ and $\boldsymbol{Y}$. $F = \bigcup_j f_j$ and $G = \bigcup_k g_k$ are families of binary-valued feature functions. Although arbitrary functions may be used, we employ the independence assumptions of *linear chain* CRFs, where each f_j is active for a distinct bigram of labels (i.e., a distinct pair of consecutive labels) and each g_k is active for a specific combination of a label and observation. Values λ_j and μ_k are estimated model parameters that are computed from training data. Normalization factor $Z(X)$ ensures that the probabilities of all label sequences sum to 1 for a given observation sequence $\boldsymbol{X}$. As Lafferty et al. showed, computing $Z(X)$ is tractable without considering every possible observation sequence [56]. In the data table scenario, $\boldsymbol{X}$ represents the list of rows in the table, and $\boldsymbol{Y}$ represents the

corresponding row classes. As an example, a transition function f_j may correspond to the transition from a header row to a data row, while a state function g_k could be active for header rows that contain no numeric values (the row features used by our method are described in Section 2.3.2).

Training a CRF model involves estimating the λ_j and μ_k parameters using one of several training methods. For our table extraction procedure, we use the limited-memory BFGS algorithm [96], an iterative process that scales well for large datasets, as implemented by the CRFsuite toolkit [77]. After a CRF is trained, classifying a new sequence is straightforward and fast, using a dynamic programming algorithm. In previous work, Pinto et al. [81] applied CRFs to the task of labeling rows in data tables, but limited their focus to ASCII tables from plain text documents. ASCII tables are useful, but we believe the quantity of other sources of data tables and the ease of identifying and extracting data from HTML tables and spreadsheets makes them more intriguing targets. Their study includes many row types that are specific to ASCII documents, while other generic row types, such as aggregate rows (e.g., containing a subtotal) are not treated separately from data rows. In addition, the variety of structures that are possible in other formats introduces different extraction challenges that are discussed later in this thesis. In Section 2.3.6, we detail our method for using CRFs in the table extraction task.

We use CRFs in our system to assign row labels to each row in a new table. First, we collect a large collection of tables with human-annotated row labels to serve as a training set. The CRF is trained on this data using the limited-memory BFGS method [96] to determine optimal values for the model parameters. The logarithmic binning scheme results in a large number of possible features, however, the training time remains relatively low. One attribute of CRF classifiers that is beneficial given our choice of cell attributes is that CRFs can perform well even when multiple features are statistically correlated [96], as ours likely are. Using the CRF model parameters, dynamic programming is used to compute an optimal

sequence of labels to assign to new data tables. This results in row label sequences, such as "TNNHGDDDAGDDDABN" for Figure 2.2, which we process in the following schema construction phase.



Patent Applications by Residents		
Data Source: worldbank.org		
(showing top countries in each continent)		
Country	Residents	Applications
North America		
United States	307,007,000	224,912
Canada	33,739,900	5,067
Mexico	112,033,369	822
	N.A. Total	230,801
Asia		
Japan	127,557,958	295,315
China	1,331,380,000	229,096
South Korea	48,747,000	127,316
	Asia Total	651,727
Note: data from 2009		

Figure 2.2: The row labeling process for a sample data table. The list of values on the right are the resulting row labels. In this case, the classifier correctly identified the first row as a title row, followed by two non-relational rows, followed by a header row, etc.

2.3.6 Row Classifier

The crux of our technique involves extracting relevant *row features* to represent each row of a data table, and then classifying each row using one of several *row classes* that we have defined. As with natural language sentences, data tables are constructed in accordance with general principles that make it easy for viewers of the tables to interpret the data without explicit instructions for how to do so. These widely-understood principles allow the communication of schema information based on implicit rules for formatting and positioning of data cells. We introduce a collection of features that preserves the formatting information present in data tables and a collection of row classes that more formally encapsulates these principles. A system built on these components can extract schema information that describes the

structure of a data table in a fully automated manner.

To find data blocks and extract their schemas, we employ techniques similar to, though more general than, those used by WebTables [21]. WebTables determines whether a given table is relational by using machine learning techniques to train a classifier that labels tables as relational or non-relational, based on several holistic features of the table. In WebTables, tables classified as non-relational are discarded. Then, tables classified as relational are subjected to additional machine learning-based classification to detect whether the first row of the table serves as a header for the table. As we see in collecting a corpus of data tables from the Web, assuming that column headers appear in the first row of a table disregards approximately 32% of tables, which are more complicated, in a collection of annotated HTML tables. Furthermore, this assumption is invalid for an alarming 75% of spreadsheets. In other words, the WebTables approach only applies to a fraction of all data tables (68% and 25% of HTML tables and spreadsheets, respectively), due to its assumptions of simplicity for the structure of the data being processed.

From our analysis of a large number of downloaded spreadsheets and HTML tables, we observed that headers, data, and other row types are frequently intermixed throughout a single data table. As a result, our approach is based on a collection of *row classes* that we have created to describe the function of rows in nearly all true relational data tables that we have encountered. For instance, one row class represents data rows with the label "D", while the label "H" represents a header row. Based on a large collection of hand-annotated data tables that we use as a training set, we use supervised classification in the form of a CRF to estimate the correlation between row features and row labels, and also to determine the relative likelihoods of transitioning from one row class to another. These estimates are used by the CRF to assign labels to rows outside of our training set.

Processing individual tables with our CRF-based classifier allows us to determine which tables contain truly relational data and to extract schema information by utilizing assigned

row classifications. The first distinction is necessary because people use spreadsheets and HTML tables for many creative purposes that do not communicate structured data. For example, spreadsheets can be used as forms or instruction manuals, while HTML tables are used far more often for controlling page layout than for actual data representation. We consider these tables to be *non-relational*, since they do not contain records of consistent types, and they are not useful to our schema extraction endeavor. As a result, given an input document, our technique can be applied to all data tables, but low scoring tables are treated as non-relational and discarded.

After determining that a table is relational, the row classes assigned by our classifier can be used to augment the original table with a more strictly relational structure, which can then be processed in a variety of ways. The most obvious approach is to concentrate on the data by simply discarding specific types of rows that are not relevant. In this way, our procedure can be used as a preprocessing phase to maximize the pool of usable tables that are accessible to downstream applications. This is the primary motivation for our method, as it allows the handling of tables that are outside the scope of the WebTables method. Of course, the results of our method are useful beyond serving as a simple filter for data rows. For example, when a table of city-level statistics is partitioned by state and this is indicated by a "group header row" before each partition, the state names will be lost if data rows are treated individually. This is analogous to database normalization, in a sense, because the state name appears only once, yet it applies to many data rows. Thus, to output a more complete table, one can "denormalize" this data by appending a column to the resulting table that contains the state name. The conversion of tables to a fully relational structure is not the primary focus of this work, but should be an interesting area to explore in the future.

The schema extraction process is visualized in Figure 2.2. In the rest of this section, we introduce data table terminology and concepts, define row classes that we attempt to identify within each table, describe the row features that we use as input for our classifier along with

the details of the classifier itself, and finally outline how the output of the classifier can be used to interpret the data tables.

2.4 Common Row Patterns

In this section, we present common row patterns. Using a large corpus of data tables with human-judged row labels, we analyzed the sequences of row labels for patterns (the corpus is described in detail in the next section). We are interested in common table structures in the form of row label sequences.

Table 2.4: Most common row patterns in data tables.

	HTML	Spreadsheets
1	HD	THD
2	THD	HD
3	HDA	TBHD
4	THDA	THDN
5	H(GD)*	HDN
6	H(BD)*	TBHD(BN)*

Table 2.4 shows a listing of the six most common row label sequences for both HTML and spreadsheet tables, using the row labels defined in Table 2.1. Consecutive instances of the same label are omitted to make the patterns more obvious. Patterns that involve repeated subsequences denote this with an asterisk. As expected, tables with one or more header rows followed by one or more data rows are very common in both table formats. Additionally, tables with a title row preceding the header and data are also common. Spreadsheets are more likely to contain blank or non-relational rows, so the other patterns are not common to both lists.

2.5 Evaluation

In this section, we present details of our evaluation of the row classification method as applied to datasets of spreadsheets and HTML tables from the Web.

2.5.1 Datasets

Since previous work has concentrated on extracting data from tables with simple structure, we found no previously created dataset that was adequate to test the accuracy of our method on complex tables. Instead, we created a new dataset of spreadsheets and HTML tables downloaded from the Web. In collecting our datasets, we sought a sample of tabular data from many different web sites and data sources so that we could test our methods' applicability across different data domains. To find relevant web sites containing tabular data, we performed several keyword searches, such as "spreadsheet" and "data table". For spreadsheets, we also explicitly searched for spreadsheet files with a .xls extension. These searches resulted in a list of Web sites, which we then crawled for spreadsheets as well as for pages containing HTML tables. Because spreadsheets and HTML pages often contain multiple tables, we consider them separately. Also, note that our current implementation accepts spreadsheets in .xls format, the default file format through Microsoft Excel 2003. The newer .xlsx file format is an ISO standard based on XML and serves as the default for newer versions of Microsoft Excel. However, based on several simple search engine queries at the time of writing, Web-accessible spreadsheets in the older .xls format appear to outnumber those in the newer .xlsx format by a ratio of roughly 50 to 1, meaning that we draw our corpus from a pool containing the vast majority of spreadsheets on the Web.

Characteristics of our collected datasets are listed in Table 2.5. Both are of comparable size in terms of number of documents and number of tables. However, several differences are apparent. The HTML collection contains many more tables per document than the

Table 2.5: Dataset characteristics.

	Spreadsheets	HTML
Document count	14669	7883
Table count	46408	63009
Row count	5113070	215735
Unique domains	3636	4957

Table 2.6: Top-level domains for table sources.

Spreadsheets		HTML	
.gov	3476	.com	4835
.us	2829	.org	795
.uk	2710	.edu	576
.com	1592	.net	438
.org	1363	.gov	412
.edu	546	.uk	241

spreadsheet collection, due to HTML tables' frequent use for page layout, rather than for data presentation. Also, the total number of rows in the spreadsheet collection far outnumbers the number of rows in the HTML collection, since spreadsheets tend to be used for data manipulation and aggregation, while HTML tables tended to be used for data presentation. Furthermore, analysis of the domain suffixes of data sources in each collection (shown in Table 2.6) reveals that most spreadsheets were collected from government sources, while HTML tables were more frequently collected from commercial and organization domains. However, both collections have good source and content variety. Spreadsheets were sampled from 3636 distinct sources, and HTML tables from 4957 distinct sources. The data in these collections come from a broad range of websites, and span many different topics, including election results, product lists, radio stations, parking lot lat/long values, demographic data, and many others.

After creating our collections of spreadsheets and HTML tables, we randomly sampled

a subset of these tables for manual annotation. An additional goal in the creation of our testing dataset was to exceed the size of hand-annotated table datasets used in previous work. The largest datasets found in work that we surveyed came from the WebTables paper and the work of Limaye et al. [67]. The former was tested on "a large sample of 1000 relations" judged by humans [20], while the latter was evaluated on collections of 437 hand-annotated tables, along with 6,085 machine-annotated tables. Our experimental dataset size is far larger, as we hand-annotated 16,048 tables (2,259 from spreadsheets and 13,789 from HTML documents), of which 1,976 were relational (1,048 and 928, respectively) and required manual annotation of each row. We annotated each row of each table with the label corresponding to its row class: "H" for header, "D" for data, etc. Furthermore, we annotated the entire table as relational if it contained at least one header and one data row, and non-relational otherwise. Statistics from our annotations are listed in Table 2.7.

While the number of annotated documents is similar for both sets, there are considerable differences as well. Spreadsheet tables are much more frequently relational than HTML tables, consisting of 46% and 7% relational tables, respectively. This large difference is not overly surprising, since most HTML tables were used for page layout or other purposes. Furthermore, 7% relational HTML tables seems at a first glance quite small, but is comparable to the 1% relational HTML tables of WebTables [20]. The larger percentage is likely due to the targeted manner in which our datasets were created. Examining the number and types of annotated rows shows that spreadsheet tables tended to be far larger, in terms of the number of rows, than HTML tables in the collection. We also studied the schema complexity of relational tables in our collections, in terms of several factors. First, we searched for tables with *simple schemas*—tables with a single header row followed by one or more data rows—and found that there was a much larger fraction of simple HTML tables than simple spreadsheet tables. We also found that more spreadsheet tables contained multiple header rows than HTML tables, and that more spreadsheet tables contained other (i.e., non-data, non-header)

Table 2.7: Tables annotated by human judges.

	Spreadsheets		HTML	
Annotated documents	1117		1204	
Annotated tables	2259		13789	
Relational tables	1048	(46%)	928	(7%)
Non-relational tables	1211	(54%)	12861	(93%)
Annotated rows	435160		20537	
Header rows	1479	(<1%)	978	(5%)
Data rows	425195	(98%)	18906	(92%)
Other row classes	8486	(2%)	653	(3%)
Relational tables:				
"Simple" schema	257/1048	(25%)	632/928	(68%)
Multiple header rows	157/1048	(15%)	63/928	(7%)
Other row classes	784/1048	(75%)	263/928	(28%)

row classes than HTML tables. These three observations indicate that spreadsheet tables tend to have more complex structure than their HTML counterparts.

Note that the schema classifications as relational or non-relational are somewhat of a simplification, as in reality, examined tables were often not completely relational or non-relational. For example, we found a multitude of nearly-empty spreadsheet tables that were intended to be printed out and used as fill-in forms, which we classified as non-relational. Web-based calendars likewise use HTML tables for formatting, and have a quasi-relational structure but do not contain usable data. Additionally, some tables could be formatted as relations, but are not, such as tables that present street addresses using multiple table rows instead of multiple columns, and again we classified these as non-relational for our purposes.

2.5.2 Experimental Setup

We performed 10-fold cross validation on our collection of relational HTML and spreadsheet tables to obtain our classification results. The dataset was divided into 10 equally-sized

groups, then 10 classifiers were trained on each subset of 9 groups, before running experiments on the remaining group. The reported results are averages from the 10 testing runs. The HTML and spreadsheet tests were performed separately in order to expose differences between the two table formats. Results were recorded for the following four separate classification methods.

- **WT**. WebTables results were obtained using the "Header Detection" features and rule-based classifier from the Weka toolkit as described in the original WebTables paper [20]. Although the WebTables features for header detection are only defined for the first row, the extension to cover other rows in a table or spreadsheet can be performed by recomputing the WebTables features for rows after the first. An additional contribution of our work is the adaptation of the WebTables classifier to work with spreadsheets along with the HTML tables that they were originally designed to process.

- **B+A**. The "Bayes + Automaton" method serves as a baseline method that incorporates global table structure. As in WebTables, a Bayesian classifier computes the estimated likelihood of each row being assigned each row label. But rather than choosing the most likely row label in isolation, a custom automaton is used to find the sequence of row labels with the highest aggregate likelihood that also adheres to the common table patterns discussed in Section 2.4, such as H(GD)*.

- **CRF-C**. The CRF method with continuous features uses a linear feature encoding to form the input to a CRF. This is a similar encoding to the one used by the "CRF Continuous" method tested by Pinto et al. [81], but uses the cell attributes and row classes we developed for spreadsheets and HTML tables.

- **CRF-B**. The CRF method with binned features uses the full row classification method given in Section 2.3, including the use of the logarithmic binning from Section 2.3.4 before performing row classification with the trained CRF.

The cross-validation sets are partitioned by document (not by table) to avoid using one table in a training partition when another similar table in the same document is in the testing partition. The reported accuracy, precision, and recall rates are averages over the 10 training and testing runs using each method.

2.5.3 Row Classification Evaluation

For our first evaluation, we tested the accuracy of the classifiers at the row classification task described in Section 2.3. This process yields the assignment of a row class to each row that represents our best guess of the row's purpose within the table (selected from our set of row classes in Section 2.3.1). The correct classification of table rows is a difficult task because a large number of tables in our corpus are not in a simple tabular format, such as those with one header row followed by a sequence of data rows. Instead, we found that many blocks are bordered by non-relational metadata, contain multi-row headers, or include subtotals or other noisy sections within a single schema block. However, the purpose of our row labeling procedure is to handle these cases, and our experiments show that it performs well.

Table 2.8: Test set rows classified correctly.

	WT	B+A	CRF-C	CRF-B
Spreadsheets	97.6%	96.7%	99.3%	99.3%
HTML Tables	92.3%	92.7%	98.2%	98.1%

As shown in Table 2.8, the CRF-based methods obtain the best classification accuracy for both spreadsheets and HTML tables, ahead of WT and B+A. The higher scores on spreadsheet rows is a result of the higher proportion of data rows found in spreadsheets, as listed in Table 2.7. The row-level accuracy is an important metric for schema extraction. However, the gains are not fully illustrated when all methods have such high scores. If a hypothetical classifier were to label all rows as data rows (D), it would achieve a 97.7% accuracy rate on

spreadsheets and 92.1% accuracy rate on HTML tables, since those are the fractions of rows in our test corpus that are data rows. Yet this classifier would be useless in the sense that all tables would contain misclassifications for non-data rows. We conclude that high row-level accuracy may hide the true power of these methods so we now look at full-table accuracy rates.

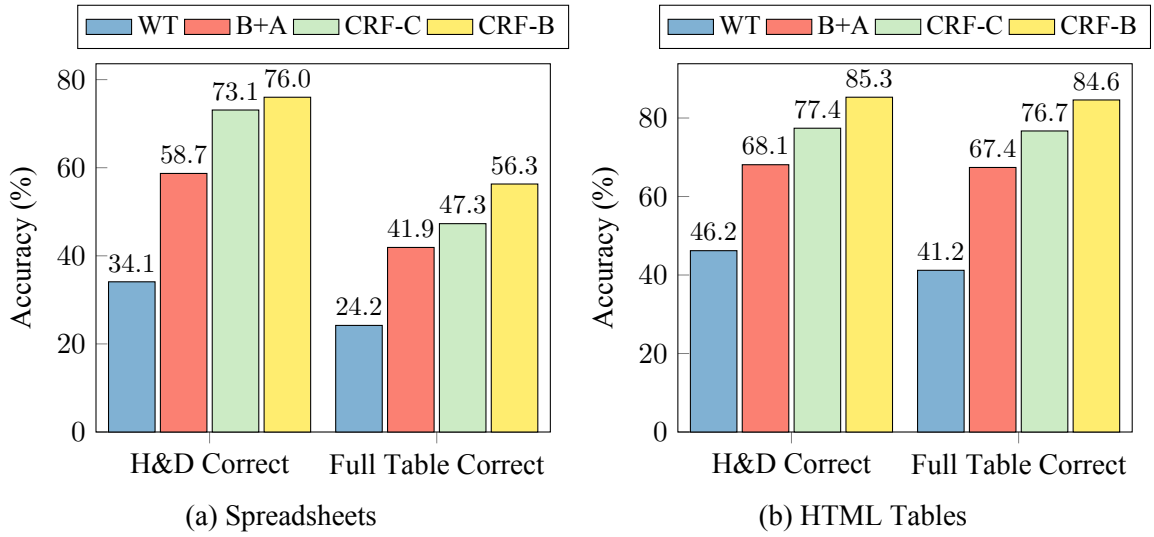


Figure 2.3: Accuracy of classifiers on full spreadsheet tables and full HTML tables. The accuracy is measured as the percentage of tables in which correct labels are assigned to (i) all data and header rows, and (ii) all rows in the table.

2.5.4 Full Table Accuracy

We evaluated the accuracy of the four classification methods on full data tables, because while individual row accuracy is important, errors anywhere within a document may well affect the usability of the data table in downstream applications. Correctly classifying individual rows is important. However, even a small number of incorrect row classifications can potentially lead to significant errors during schema extraction. Thus, we focused this experiment on the performance of our classifier on entire data tables. The goal, of course, is perfect classification of each row. However, the schema extraction applications we envision require highest accuracy for data rows, followed by header rows. If all of those are correctly classified, we can interpret the table relationally, even without correctly classifying

the remaining rows (alternatively, we can treat them all as "non-relational").

The full table accuracy test measures the number of tables in which all rows are correctly classified and the number of tables in which all data and header rows are correctly classified (i.e., no false positives or false negatives for those classes). We make a distinction between these two cases because errors in header and data are critical to classify correctly for many purposes, while other row types may be "nice-to-have" for some applications, but are not always crucial. The results are shown in Figure 2.3. For both spreadsheets and HTML tables, the CRF-B method achieves the best accuracy for parsing full tables correctly. Even when only header and data rows are considered, which are what the WebTables features are designed to distinguish between, the WT method is significantly outperformed by the others.

It is also noteworthy that the document-level accuracy is higher for HTML tables, for all methods, despite the lower row-level accuracy in Table 2.8. This is likely due to the higher proportion of "simple tables" that exist in the HTML corpus, in combination with the lower overall proportion of rows that are data rows.

2.5.5 Effects of Feature Binning

The accuracy benefits of the CRF-based methods over WebTables and our automaton method are clear from the previous subsection. Now we examine the differences between the CRF methods, CRF-C and CRF-B in order to measure the effects of logarithmic binning. Since the only difference between the two methods is the feature encoding scheme, we can conclude that it is the sole cause of any changes in accuracy. The results of this test are displayed in Table 2.9. We use the standard precision and recall definitions, where precision is the proportion of all classifications of a specific row class that are true members of that row class (based on the human annotations), and recall is the fraction of all true members of a row class that were classified as such. The F_1 score is computed as $(2 \cdot P \cdot R)/(P + R)$ for precision P and recall R, and the change in F_1 score between CRF-C and CRF-B is displayed.

Table 2.9: The precision and recall for the CRF-C and CRF-B classification methods on spreadsheets and HTML tables. The change in F_1 score that results from using the logarithmic binning scheme of the CRF-B method is also shown. Row classes are ordered by average frequency across table types.

Row Class	Count	CRF-C Precision	CRF-C Recall	CRF-B Precision	CRF-B Recall	Change in F_1 Score
Spreadsheets						
D	425376	.999	.999	.998	.998	−.001
H	1486	.937	.915	.945	.915	+.007
B	3792	.874	.862	.908	.974	+.071
T	702	.739	.756	.766	.822	+.046
G	1312	.669	.480	.758	.385	−.048
N	1877	.576	.709	.446	.639	−.111
A	615	.965	.703	.991	.890	+.123
HTML Tables						
D	18920	.988	.995	.991	.995	+.001
H	979	.921	.908	.911	.939	+.011
B	214	.852	.719	.984	.953	+.188
T	154	.702	.717	.875	.913	+.184
G	112	.667	.353	.545	.176	−.195
N	69	.667	.095	.120	.143	−.037
A	89	.059	.074	.706	.444	+.479

The results show some increases and some decreases in the F_1 scores for individual row classes. The high precision and recall of data rows is a result of their frequency in both spreadsheets and HTML tables. Both classification methods produce similar F_1 scores on these rows. At the opposite end, we note that the low precision and recall rates for group header rows, aggregate rows and non-relational rows in HTML tables is mainly due to the small number of tables containing rows of these types—which is the case because HTML table authors have the ability to place notes or explanations or other metadata within the document, but outside of the table. In contrast, all visible text in a spreadsheet is placed in spreadsheet cells, so there is no other place for notes to appear.

Although the use of logarithmic binning does not universally increase the F_1 score across all row classes, we emphasize that it does achieve our main objective of increased full table accuracy. The primary value of these precision/recall results is to examine how row-level accuracy is affected by the use different feature encoding methods. To that end, we see that for both table types, CRF-B improves on CRF-C for blank rows, header rows, title rows, and aggregate rows, while the F_1 score is reduced in the cases of group header rows and non-relational rows. The logarithmic binning scheme was not designed for the recognition of specific row classes, so the fact that the same row classes show improvements and reductions in accuracy for both spreadsheets and HTML tables is somewhat surprising. A common trait of both non-relational rows and group header rows is that they often contain a single non-blank cell in the leftmost column of a table, while the other cells are blank. Consequently, the divisions of cell features based on the number of total cells in a row may impede feature generalization and reduce the accuracy for these row classes. Examination of more complex binning schemes that address this is left as future work.

2.5.6 Row Class Ambiguity

One way to identify aspects of our approach that may need improvement is to examine the number of rows of each class that are confused for rows of each other class. We do this using the confusion matrix shown in Tables 2.10 and 2.11. Each cell of the matrix shows the percentage of all classified rows that were actually of the class with the label shown in the leftmost column, but were assigned the row label shown in the top row by the CRF-B classifier. The shaded cells along the diagonal show correct row classifications, while the surrounding cells show incorrect classifications. Row classes are ordered by average frequency across table types. Totals for each row and column are also displayed.

Error-free classification would result in zeroes for all of the non-shaded values (i.e., off of the diagonal). However, since our method does result in some misclassified rows, we

Table 2.10: Confusion matrix for CRF-B on spreadsheets.

Row label (true)	D	H	B	T	G	N	A	Row Sum
D	97.54%	0.00%	0.04%	0.00%	0.01%	0.16%	0.00%	97.75%
H	0.02%	0.31%	0.00%	0.00%	0.01%	0.00%	0.00%	0.34%
B	0.01%	0.00%	0.84%	0.00%	0.00%	0.01%	0.00%	0.87%
T	0.00%	0.00%	0.00%	0.13%	0.00%	0.03%	0.00%	0.16%
G	0.04%	0.00%	0.00%	0.00%	0.12%	0.14%	0.00%	0.30%
N	0.05%	0.01%	0.04%	0.04%	0.02%	0.28%	0.00%	0.43%
A	0.01%	0.00%	0.00%	0.00%	0.00%	0.00%	0.13%	0.14%
Col Sum	97.66%	0.33%	0.93%	0.18%	0.15%	0.62%	0.13%	

Row label (assigned)

can see where the errors lie. From the confusion matrix for spreadsheets, we observe that two of the non-diagonal cells have relatively high values. These are the cases of (D, N) (where the true row class is D, but the row class assigned by CRF-B is N) and (G, N). The fact that misclassified rows are often interpreted to be non-relational (N) is not surprising, since non-relational rows are fairly heterogeneous (especially in spreadsheets), and come in a variety of forms. That they are mistaken for data (D) and group header (G) rows is also not surprising, given the overall prevalence of data rows, and the frequent similarity between non-relational rows (such as notes) and group header rows, which both appear commonly with a single value in the first column of a row. True non-relational rows are also frequently misclassified.

The confusion matrix for classifying rows in HTML tables shows that the largest errors arise for (D, H), (G, N), (H, D), (A, D), and (N, D). These HTML row misclassification rates are all higher than their counterparts in spreadsheet rows. This has multiple causes, but is primarily due to the size differences between average HTML tables and average tables found in spreadsheets. Second, HTML tables are generally narrower, which results in less contrast between the row types and higher levels of confusion. Finally, although the relative

Table 2.11: Confusion matrix for CRF-B on HTML tables.

Row label (true)	D	H	B	T	G	N	A	Row Sum
D	91.64%	0.34%	0.02%	0.00%	0.02%	0.03%	0.08%	92.12%
H	0.24%	4.47%	0.00%	0.03%	0.02%	0.00%	0.00%	4.76%
B	0.05%	0.00%	0.99%	0.00%	0.00%	0.00%	0.00%	1.04%
T	0.00%	0.05%	0.00%	0.68%	0.02%	0.00%	0.00%	0.75%
G	0.08%	0.02%	0.00%	0.03%	0.10%	0.32%	0.00%	0.55%
N	0.19%	0.03%	0.00%	0.03%	0.03%	0.05%	0.00%	0.34%
A	0.24%	0.00%	0.00%	0.00%	0.00%	0.00%	0.19%	0.44%
Col Sum	92.45%	4.91%	1.00%	0.78%	0.18%	0.41%	0.28%	

Row label (assigned)

proportion of some row classes (i.e., G, A, and N) is higher in the HTML tables of our corpus than in the spreadsheet tables, the absolute number of rows of these classes is less, so there are fewer training examples. Many of the pairs of confused row labels for HTML tables are similar to those for spreadsheets. Additionally, both data rows and header rows are confused for each other, which can be attributed to the higher proportion of header rows in HTML tables. The errors on aggregate rows could potentially be reduced by incorporating more text indicators into our feature set, including words like “Average” and “Sum”, which we observed in aggregate rows in our corpus of data tables.

2.5.7 Application to Existing Table Dataset

Our final experiment illustrates the accuracy of the CRF-B method on one of the largest pre-existing published table datasets, from the work of Limaye et al. [67] (we found no such dataset for spreadsheets). The Limaye dataset contains four types of relations, three of which are full HTML tables containing headers (the “Wiki_Manual”, “Web_Relations”, and “Wiki_Link” collections). The vast majority of these exhibit simple table structure and contain only header and data rows. Since the original classification goals of this dataset were

different from ours, the existing annotations were not sufficient for our purposes, so we annotated each table in the dataset by hand for our row classification task. The statistics for each table collection are shown in Table 2.12. While Wiki_Manual and Web_Relations are small, Wiki_Link is very large, comprising nearly 6,000 relational tables. Tables with simple schemas make up a large percentage of each collection, which is expected since a method similar to WebTables was used to select them.

Table 2.12: Limaye dataset: table information.

Collection	# Relational	# Rows	% Simple Schemas
Wiki_Manual	38	1423	86.8%
Web_Relations	28	1837	78.6%
Wiki_Link	5753	120765	98.0%

We trained the CRF-B classifier on the dataset that was described in Section 2.5.1, and then tested that classifier on the three collections of HTML tables. The results, listed in Table 2.13, display the percentage of all rows that were correctly classified, along with the percentage of tables in which all H and D rows were correct and the percentage of tables that were entirely correct. Two important observations are worth highlighting. First, each collection in the Limaye dataset contains a larger percentage of simple schemas than is found in our own corpus of HTML tables. This means that using our method to detect and process tables could potentially increase the pool of accessible tables for applications such as the one described by Limaye et al. [67]. Second, even for the tables that were included in this dataset, and that are assumed to have simple structure, our method is valuable, since the full table accuracy numbers in Table 2.13 exceed the percentages of tables with simple schemas. This is because a non-trivial number of tables with aggregates, row groupings, and non-relational data rows are present in this dataset and are accurately detected by our CRF-B classifier, but are not dealt with by simpler methods.

As one example of the benefit of our approach, we examined a table in the Wiki_Link

Table 2.13: Limaye dataset: results using CRF-B classifier.

Collection	Rows	Tables - H&D	Tables - Full
Wiki_Manual	99.9%	100%	97.4%
Web_Relations	99.1%	89.3%	89.3%
Wiki_Link	99.9%	99.1%	99.1%

collection that contains data about population density in regions of Italy. There are columns for Region, Population, Area, and Density, and data rows for every Italian region, followed by an aggregate row for the same statistics for all of Italy, with the string "Italy" in the Region column. The final row is correctly detected as an aggregate row by our CRF-B method, due to its bold font formatting, which allows it to be treated differently from the other rows in the table. However, if the row is not segregated from the others, methods such as the least common ancestor (LCA) method evaluated by Limaye et al. would be adversely affected when determining the column type and cell entity assignments. Even algorithms that are flexible enough to handle multiple types in a single column could benefit from having less noise from aggregates such as this example, or group headers and non-relational row text. Thus, the use of our method as a preprocessor could improve accuracy of table extraction methods like this by filtering rows that do not fit with the data rows in the table.

2.6 Discussion of Column Properties

Here we present a collection of properties for describing table columns. While our primary concern has been the identification of table row patterns, thereby isolating data rows from other structural row types, table columns likewise exhibit a variety of functional properties, and the sequence of such properties follow common patterns. Unlike with row classes, which were used to describe each row in the raw table, the column classes apply only to the data portions of tables, and thus must be assigned after row classification has been performed.

- *Primary Entity Columns* are analogous to primary key columns in a relational database.

Typically they are a single column, but they can also be multiple columns or zero columns. Each row should contain a unique value or combination of values compared to the other rows of the table, but as with all data from the Web, this may not be strictly followed by all table authors.

- *Attribute Columns* constitute the majority of columns in data tables. Each cell contains a value that can be viewed as an attribute of the primary entity column value for the row. The majority of tables contain one or more primary entity columns and one or more attribute columns.
- *Ranking Columns* allow viewers to see the row's position within the table at a glance and typically consist of a sequence of non-decreasing integers (1, 2, 2, 4, …) or monotonic integers if there are ties by the ranking criteria (1, 2, 2, 4, …).
- *Ordered Columns* are columns whose values determine the ordering of rows in the table. Tables are commonly ordered alphabetically, numerically, or by date/time, but occasionally by an external ordering of entities (e.g., Sunday, Monday, Tuesday, …).
- *Secondary Entity Columns* are similar to foreign key columns in a relational database. They differ from primary entity columns because not all attributes in the table describe a secondary entity.
- *Cascading Columns* are sparse columns where values are understood to apply to multiple rows. For example, a table containing county-level demographic data may be organized by state and have the state column populated only for the first row where it applies.
- *Geographic Columns* contain references to geographic locations. Many data tables describe geography, often specified textually. The values within one geographic column are usually members of a single geographic type, such as cities, countries, or postal codes. Many tables include multiple geographic columns, in which case the geographic

cells in each row typically exhibit a containment relationship.

- *Non-relational Columns* contain values that are not associated with data values in the row (e.g., notes, asides).
- *Blank Columns* contain no data, but their presence may indicate transitions that are useful for determining the function of neighboring columns.

The primary entity and attribute columns are the most common and most useful to detect. The others vary in the degree of difficulty required for proper identification. Unlike with row classes, it is common for a single column to have more than one of these properties (e.g., when a column is both ordered and geographic). Many classifiers, such as the CRF model we employed for row classification, can be adapted to deal with multi-label data [32, 104].

2.7 Summary

We first developed a set of row classes that represent the most common functions of individual rows in a data table. We then identified cell attributes that are combined using the novel logarithmic feature binning technique to serve as input to a classifier based on conditional random fields. The classifier outputs the sequence of row labels with maximum likelihood based on these inputs, thus determining the row classifications that are used for extracting data and structure from the table. This method was shown to lead to a significant improvement over the existing WebTables approach, and our logarithmic binning scheme shows improvements over alternative feature selection methods. Specifically, our CRF method with logarithmic binning showed substantial improvement over all alternatives in the full table extraction test, which demonstrates our method's ability to process arbitrary tables.

Chapter 3

Structured Toponym Resolution

In this chapter, we explore a technique for place name disambiguation (also known as toponym resolution) that uses Bayesian inference to assign categories to lists or tables containing place names, and then interprets individual toponyms based on the most likely category assignments. The categories are defined as 3-tuples of nodes in a place taxonomy composed of three orthogonal hierarchies: place types (e.g., cities, capitals, rivers, etc.), geographic containers, and prominence (e.g., based on population). The algorithm is validated on a collection of geographic tables, where it is shown to achieve higher accuracy than alternative methods which do not make use of the full taxonomy. The table geotagging process associates geographic tables with metadata that is essential for enabling spatial indexing and browsing operations.

The rest of this chapter is organized as follows. Section 3.1 discusses various challenges involved in the interpretation of place names in tables. Section 3.2 presents the toponym resolution algorithm. Section 3.3 describes experiments showing our method's categorization and toponym resolution accuracy. In Section 3.4, a demonstration system is presented, which visualizes the geotagging process. Section 3.5 contains some concluding remarks.

Figure 3.1: Table with a location column containing $L = [\text{Alexandria, Arlington, Springfield, Vienna}]$, geotagged by Wolfram Alpha. Wolfram Alpha interprets each toponym as the most populated place with the name, so "Alexandria" is associated with "Alexandria, Egypt", "Vienna" with "Vienna, Austria", "Arlington" with "Arlington, TX, USA", and "Springfield" with "Springfield, MO, USA".

3.1 Geographic Tables

The table geotagging problem can be formalized as follows. Given a grid of data cells, each containing a character string, determine which cells contain geographic references (the *toponym recognition* task) and provide the most likely geographic interpretation for each selected cell (*toponym resolution*) in the context of the other cells in the grid.

Our method is motivated by the following model of how table authors construct lists and tables that contain geographic information. First, the author recognizes that one or more geographic locations are associated with each entity in the table. The entity for each row may itself be a geographic entity, but this situation need not be treated as a special case. After deciding on the geographic entities that will appear in the table, the author includes a column with the most descriptive geographic references for each entity. In some cases, the author includes multiple columns per geographic reference, with the additional columns providing geographic containers for entities in the primary geographic column. In other cases, the author assumes the contrast of nearby values can be used to disambiguate the toponyms [110].

Figure 3.2: List L, geotagged by our algorithm, which recognizes that the list of toponyms in L are likely to refer to a cluster of nearby cities in the American state of Virginia.

In the table setting, the context comes in the form of coherent categories of toponyms within the geographic columns, such as "provinces in Canada", or "parks in Texas" or "prominent cities in Europe". There are multiple dimensions to these categories: the *feature type*, *geographic container*, and *feature prominence*.

Unfortunately, many places share names, a well-known geotagging challenge known as entity-entity ambiguity [75] or geo-geo ambiguity [11, 15]. Some place names, such as "Victoria", "San Antonio", and "Rome"/"Roma", are reused for hundreds of places in dozens of countries. Additionally, even within a small geographic area, a name can be used to describe a variety of places, such as "Rappahannock", which describes a county, cemetery, mountain, and river in the American state of Virginia. The process of resolving this ambiguity is known as toponym resolution [11, 64] and in this section we explore a method for toponym resolution in the context of tables. This is in contrast to the related problem of toponym recognition [62], where we are interested in determining whether a reference is to be interpreted as a toponym or not (i.e., is "Jordan" the name of a person or a location).

In contrast to toponyms in plain-text documents, the toponyms that appear within a list or table column are much more likely to have strong consistency among their types, geo-

graphic containers, or prominence, or a combination of all three. Several publicly-available Web systems support geotagging data from a spreadsheet or list, such as Google Fusion Tables[1] and Wolfram Alpha[2], along with special purpose systems such as BatchGeo[3] and MapAList[4]. Each of these systems performs well when fed documents with well-specified locations, such as mailing addresses. But the results are poor for individual toponyms, as shown in Figure 3.1, which demonstrates the result of resolving each toponym individually, as done by Wolfram Alpha. This can be improved by categorizing possible interpretations, as shown in Figure 3.2. The more specific problem of geotagging data tables has been addressed in some settings, such as for ontology extraction [29] and entity discovery in Fusion Tables [85]. Additionally, some prior work by Lieberman et al. [64] investigates methods for geotagging spreadsheets, by employing heuristics to determine if a collection of toponyms can be viewed as either (1) all prominent, (2) all nearby, or (3) all similar place types. These methods use thresholds to determine which places were prominent or nearby, which we aim to eliminate using a probabilistic Bayesian method.

3.1.1 Problem Definition

We are given a data table $\mathcal{D}$ from a spreadsheet or HTML document that includes one or more columns of place names. The table contains a two-dimensional grid of data values, where $d_{i,j}$ represents the character string in the i-th column and j-th row of the data values. A gazetteer $\mathcal{G}$ is used to identify place names from the table. Each geographic entity $g_i \in \mathcal{G}$ is associated with several attributes by the gazetteer: *name*, *alternate names*, *feature type*, *geographic container*, *population*, and *coordinates*. The goal is to discover the mapping $F : \mathcal{D} \rightarrow \mathcal{G} \cup \varnothing$ that resolves each $d_{i,j} \in \mathcal{D}$ to a geographic entity $g_i \in \mathcal{G}$ or to nothing (indicating that the string value is not a reference to a place).

[1] http://tables.googlelabs.com/
[2] http://wolframalpha.com/
[3] http://batchgeo.com/
[4] http://mapalist.com

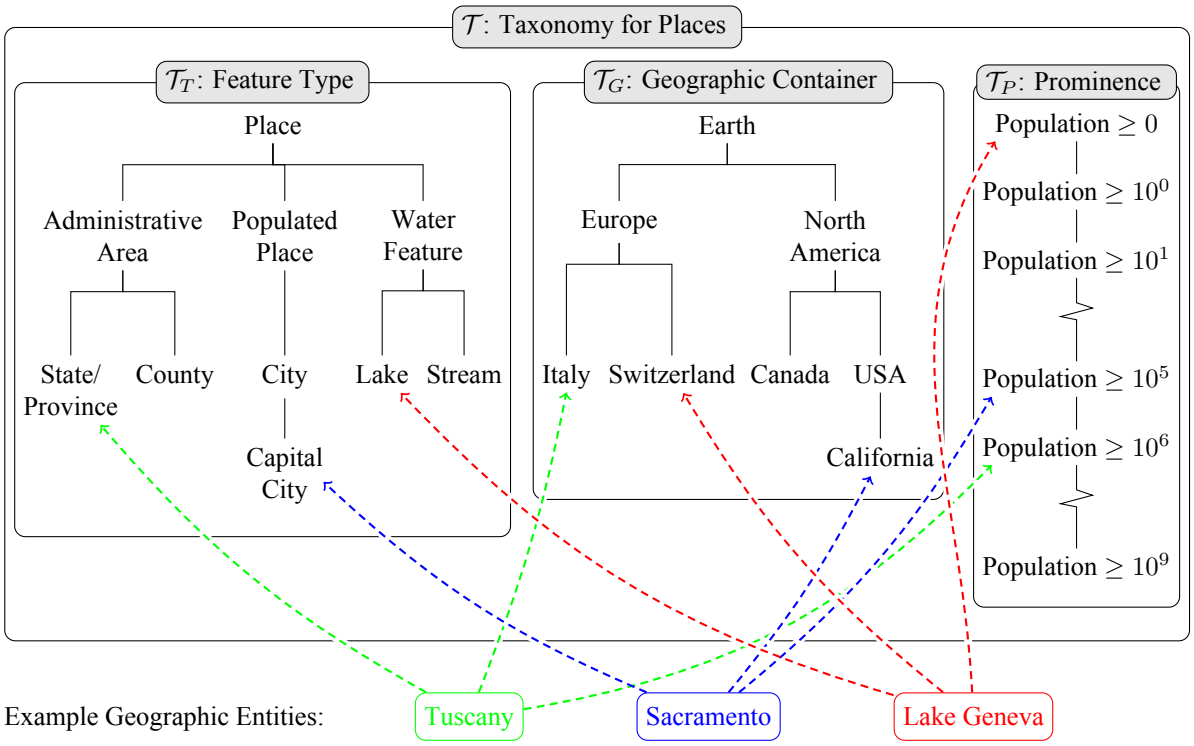


Figure 3.3: Simplified fragments of $\mathcal{T}$, the taxonomy for geographic entities that is generated from gazetteer data. The taxonomy is divided into three dimensions, $\mathcal{T}_T$, $\mathcal{T}_G$, and $\mathcal{T}_P$, which describe the feature type, geographic container, and prominence for geographic entities, respectively. Every geographic entity in the gazetteer belongs to a category $c \in \mathcal{T}_T \times \mathcal{T}_G \times \mathcal{T}_P$. Three example entities are displayed, along with their corresponding locations in each hierarchy.

3.2 Geotagging Data Tables

We view a toponym d as a character string with one or more possible geographic interpretations. The set of possible interpretations, $Geo(d) = \{g \in \mathcal{G} \mid g \text{ is a geographic interpretation of } d\}$, is determined by the entities of the gazetteer $\mathcal{G}$. For example, the interpretations of the string "Washington" include the city of Washington, D.C., the American state of Washington, the city of Washington, England, along with dozens of other, less prominent interpretations. The exact collection of interpretations depends on how strictly names are matched, such as whether gazetteer entries for "Mount Washington" or "Washington County" are included as interpretations for the string "Washington".

3.2.1 Data Extraction

In order to improve the quality and consistency of the dataset upon which this method is applied, input tables are preprocessed using the CRF-based row classifier, as described in Chapter 2. This preprocessing phase makes it possible to handle complex spreadsheets and HTML tables that contain structures such as multiple header rows, sub-total rows, and notes/footnotes or non-relational rows, which might otherwise cause errors in the geotagging process. The input to the preprocessor is a table in either spreadsheet format (.xls) or an HTML file that includes TABLE elements. For relational tables that contain at least one data row, the preprocessor outputs a list of row classes that describe the functions of the rows within each table, which are then used to isolate the cells that contain data values from the other components of the table. The remainder of our method is applied to the data cells only.

3.2.2 Taxonomy for Geographic Entities

The properties that are provided by the gazetteer are used to generate a taxonomy $\mathcal{T}$ for describing geographic entities, which is partially depicted in Figure 3.3. The taxonomy is defined by hierarchies that represent properties along three orthogonal dimensions.

- **Feature Type**. The feature type describes the class of objects that an entity belongs to, such as "Capital City" or "Park" or "Stream" or "County". Feature types belong to a type hierarchy $\mathcal{T}_T$.

- **Geographic Container**. The geographic container is an administrative region in which the entity appears. Geographic containers belong to a geographic container hierarchy $\mathcal{T}_G$, in which counties or minor regions are contained by states or provinces, which in turn are contained by their countries. For example, a category could describe entities that are in South Africa, or in Shanghai, China.

- **Prominence**. For our purposes, an entity's prominence is derived from its population. Our formulation uses the $log_{10}(pop)$ as the prominence for a place with population pop. We view the prominence hierarchy $\mathcal{T}_P$ as having multiple levels, but no branches.

The tree structure of these hierarchies is not directly available from the GeoNames database. However, it can mostly be generated from the attributes of individual places. The place type hierarchy consists of levels for the GeoNames' *feature class* and *feature code* attributes, along with an additional level for groups of similar types. The geographic container hierarchy is derived from the administrative containers listed in GeoNames and an added level representing continents as the parents of country containers. We note that many geographic categorizations are handled by the geographic container dimension, but properly recognizing proximity relationships that cross borders is not directly supported in the current approach, unless the most likely category includes the common ancestor of the locations in the container hierarchy. The difficulty arises because proximity relationships lack the regular structure of a proper containment hierarchy, meaning that the inclusion of such information into our place taxonomy would lead to an intractable number of possible categories (e.g., *"cities within 200 miles of Berlin"* where Berlin could be replaced with any location on the globe and the distance threshold could also vary). However an extension to our method that utilizes proximity relationships should be considered in future work.

We define taxonomy $\mathcal{T}$ as the Cartesian product $\mathcal{T}_T \times \mathcal{T}_G \times \mathcal{T}_P$ and an element $c \in \mathcal{T}$ is called a *category*, which has three components, one for each dimension of $\mathcal{T}$. Each entity $g \in \mathcal{G}$ has a specific category. For example, "Franklin County" is a county in Ohio, USA with population $>$ 1,000,000. The category for this entity, denoted $Cat(g)$, is $\langle$County, Ohio, Population $\geq 10^6\rangle$ with the English description *"counties in Ohio, USA, with population $\geq$ 1,000,000"*. In addition, this entity could *satisfy* many other, less-restrictive categories, such as c' = *"places in USA with population $\geq$ 10,000"*. The Boolean function $Sat(g, c)$ is defined to be true if and only if entity g satisfies category c in this way.

3.2.3 Features

The following measures are useful for estimating how well a list of toponyms is described by a specific category.

Coverage. The *coverage* of a category c over a set of column values D is defined as the fraction of values in the column with interpretations that satisfy the category.

$$Cov(D, c) = |\{d \in D \mid \exists g \in Geo(d) : Sat(g, c)\}| \,/\, |D| \tag{3.1}$$

For example, for D = [Washington, New York, Miami] and $c = \langle$City, United States, Population $\geq 10^6\rangle$, we have $Cov(D, c) = 1.0$ because all entries in D are names of large cities in the United States. For $c' = \langle$State/District, United States, Population $\geq 10^6\rangle$, $Cov(D, c') \approx 0.67$ because there is no state of Miami in the United States, so no interpretation of Miami satisfies the category.

Ambiguity. One way to differentiate between categories for describing a set of place names D is to estimate how specifically each category describes D. For example, the category $\langle$Place, Earth, Population $\geq 0\rangle$ is satisfied by any valid set of place names. However, this is not a very specific category, and results in a lot of ambiguity when trying to resolve

place names within the category. To encapsulate this concept quantitatively requires defining the *ambiguity* of a category c over a set of string values D as the average number of interpretations for each string value that satisfy the category. That is, the ambiguity is equal to the total number of possible combinations of interpretations, normalized over $|D|$ (i.e., the geometric mean).

$$Amb(D, c) = \left(\prod_{d \in D} |\{g \mid g \in Geo(d), Sat(g, c)\}| \right)^{1/|D|} \tag{3.2}$$

As an example, the cities of Conway, Lockhart, Oakland, and Oak Ridge are suburbs of Orlando with populations greater than 1,000. However, there exist larger cities that share those names across the United States. In particular, there are three cities named Conway, along with two each named Lockhart, Oakland, and Oak Ridge in the United States with populations greater than 10,000. Thus there are $3 \cdot 2^3 = 24$ possible combinations of interpretations for these place names in this category, resulting in an ambiguity value of $24^{1/4} \approx 2.21$. By contrast, for the suburb interpretations, there is exactly one city of population greater than 1,000 with each of those names in Orange County, Florida, so that category has an ambiguity value of 1. This matches our expectation that the intended categories for a toponym lists have low ambiguity, all else being equal. However, rather than enforce the direction of the correlation, we leave it to a Bayesian model.

The algorithm for computing the coverage and ambiguity of place categories for a given list of toponyms is shown in Algorithm 1. It takes a list of toponyms D as input, and returns a list of candidate place categories C, augmented with the coverage and ambiguity measures defined above. The algorithm begins by initializing the return set C (line 1) and ColumnCounts lists (line 2), which will accumulate a sequence of values for each category. The purpose of these values is to count the number of interpretations of each toponym that satisfy the category. The algorithm proceeds by iterating through each string value in D and deter-

Algorithm 1: FINDCATEGORIES(D). Given a list of toponyms, return a set of corresponding place categories, with coverage and ambiguity values.

input : List of toponyms D
output : Set of categories C

$C \leftarrow \emptyset$
Initialize ColumnCounts$_c \leftarrow$ empty list *for all* $c \in \mathcal{T}$
for each $d \in D$ **do**
  Initialize CellCounts$_c \leftarrow 0$ *for all* $c \in \mathcal{T}$
  $I \leftarrow \{i \mid i$ is a geographic interpretation for $d\}$
  for $i \in I$ **do**
    $(c_T, c_G, c_P) \leftarrow$ GETSPECIFICCATEGORY(i)
    $p_T \leftarrow$ GETANCESTORS$(c_T) \cup \{c_T\}$
    $p_G \leftarrow$ GETANCESTORS$(c_G) \cup \{c_G\}$
    $p_P \leftarrow$ GETANCESTORS$(c_P) \cup \{c_P\}$
    for $c \in p_T \times p_G \times p_P$ **do**
      CellCounts$_c \leftarrow$ CellCounts$_c + 1$
  for $c \in \mathcal{T}$ *where CellCounts*$_c > 0$ **do**
    Append CellCounts$_c$ *to* ColumnCounts$_c$
for $c \in \mathcal{T}$ *where ColumnCounts*$_c$ *is not empty* **do**
  c.cov $\leftarrow$ |ColumnCounts$_c$|$/|T|$
  c.amb $\leftarrow (\prod$ ColumnCounts$_c)^{1/|T|}$
  $C \leftarrow C \cup \{c\}$
return C

Location	Date	Sales
Rome	...	...
Athens	...	...
Dublin	...	...

Category	Coverage	Ambiguity	Normalized Likelihood
country capitals with population $\geq$ 100,000 in Europe	1.00	1.00	70.13%
county seats with population $\geq$ 10,000 in Georgia, USA	1.00	1.00	15.07%
administrative regions with population $\geq$ 100,000 in Europe	1.00	1.26	13.88%
populated places with population $\geq$ 100 in Pennsylvania, USA	1.00	1.00	0.60%
populated places in Ohio, USA	1.00	2.15	0.05%
places in Missouri, USA	1.00	1.00	0.04%
farms in Limpopo, South Africa	1.00	2.47	0.04%
administrative regions with population $\geq$ 1,000,000 in Europe	0.67	1.41	0.03%
third-order administrative divisions with population $\geq$ 100,000 in Europe	0.67	1.00	0.03%
…	…	…	…

Figure 3.4: A sample table (top) and the resulting ranked list of column categories (bottom). The set of possible categories, along with their coverage and ambiguity values, is computed using the FindCategories algorithm. Coverage values below 1.0 indicate that not all toponyms have geographic interpretations that are described by the category. Conversely, ambiguity values greater than 1.0 indicate that multiple geographic interpretations exist within the category for at least one of the toponyms. The likelihood values are the Bayesian classification results.

mining how many interpretations of each string value satisfy the possible place categories. For each interpretation of a string value, the function GETSPECIFICCATEGORY(i) returns the most specific satisfying category based on the gazetteer attributes and the components of that category are stored separately (line 7). Each component represents a node in one of the hierarchies of the taxonomy $\mathcal{T}$, and the function GETANCESTORS(n) returns all ancestors of a given node within the taxonomy (lines 8 to 10). The collection of ancestors within each dimension are combined using the Cartesian product to get the set of all categories that are satisfied by the interpretation being examined (line 11). This step makes use of the important property of our taxonomy that anything that satisfies a node n in one of the dimension hierarchies will also satisfy PARENT(n). Once the full list of satisfying categories is computed, the number of interpretations is counted (line 12) and the counts are accumulated (lines 13 and 14). Finally, the accumulated counts are used to compute the coverage and ambiguity of each category with respect to the values in D (lines 15 to 18) and the resulting set of categories is returned (line 9).

3.2.4 List Categorization

Having computed $Cov(D, c)$ and $Amb(D, c)$ for every possible place category, we proceed by identifying which of those categories is most probably the correct one. A Bayesian classifier determines the estimated likelihood that each category c is the intended category for a set of string values D. Namely, for each $c \in \mathcal{T}$, we compute an estimate of $p(C_D = c \mid c, Amb(D, C), Cov(D, c))$, where C_D represents the intended category for the toponyms in D. In practice, many categories are not satisfied by any interpretation of any of the string values, so the space of possible categories is a small portion of the full category space. The example toponym list [Rome, Athens, Dublin] has at least one interpretation in 2,141 categories, which appears to be a large number for a three toponym list, but is dwarfed by $|\mathcal{T}|$, the total number of possible categories. To estimate each category's likelihood, the components of the category $c = \langle c_T, c_G, c_P \rangle$ are treated separately so we have

$l_c(D) = p(C_D = c \mid c_T, c_G, c_P, Amb(D,c), Cov(D,c))$. We want to separate the influence of each term on the final likelihood estimate. However, the effect of each term is very dependent on the coverage value. So, we apply a assumption of independence between all conditions except for $Cov(D,c)$. This allows us to estimate the joint probability as the product of four factors, $p(C_D = c|c_T, Cov(D,c)) \cdot p(C_D = c|c_G, Cov(D,c)) \cdot p(C_D = c|c_P, Cov(D,c)) \cdot p(C_D = c|Amb(D,c), Cov(D,c))$. We can employ the chain rule of probabilities to show that $p(A|B,C) = p(C|A,B)p(A|B)/p(C|B)$, which we apply to the individual factors of the likelihood estimate to transform them into a form that we can approximate using relative frequencies found in a training dataset.

Some transformations are performed to increase the generality of the training instances. First, the depth within $\mathcal{T}_G$, rather than the node itself, is used to match a category candidate with categories in the training data, which avoids geographic bias in our model. Second, the values of $Amb(D,c)$ are discretized in order to emphasize categories that are completely unambiguous (i.e., when $Amb(D,c) = 1.0$). Finally, the likelihood of a category coverage value, given one of the category components or the ambiguity value, is modeled as a truncated normal distribution over the [0, 1] interval, whose mean and standard deviation are computed from training data [54]. For our system, training data comprised a randomly selected set of 20 toponym lists along with their proper categories. The actual number of training instances is actually much larger than this, as the 20 lists each have interpretations in a large number of categories (between 280 and 6,097 categories per toponym list), where all but one serve as negative training examples.

As shown in Figure 3.4, the result of this process is a collection of potential categories with corresponding likelihood values. The final steps of our algorithm are to select the most likely category c based on the computed likelihood values and to resolve each toponym $d \in D$ by selecting the most prominent interpretation $g \in Geo(d)$ such that $Sat(g,c)$.

3.3 Evaluation

To evaluate the effectiveness of our table geotagging algorithm, we developed a system to process tables from the Web. In this section, we describe our dataset, analyze its spatio-textual composition, evaluate the effectiveness of our method for categorization, and compare its accuracy to that of alternative methods for table geotagging.

3.3.1 Dataset

Our experimental table dataset was selected from our large corpus of spreadsheet documents (in Microsoft Excel format) and HTML tables. Non-relational tables were discarded by the preprocessing phase, as described in Section 3.2.1. From the remaining tables, we sampled 20,000 spreadsheets and 20,000 HTML tables. To avoid biasing our dataset towards large documents with many tables (i.e., spreadsheets with multiple worksheets or HTML documents with many TABLE elements), at most one table was selected from each document. As an initial filter, we identified columns that contain text values that match place names in GeoNames. To do so, we discarded any column containing fewer than three toponyms that matched GeoNames entities, within the first 100 values in the column. We then applied our algorithm to the remaining columns in order to identify those that can be described by a category from our taxonomy. This resulted in a collection of 12,861 columns from 8,422 tables. The automated processing of these columns by our algorithm results in a large array of categories, which we describe here.

The distribution of the place types (from $\mathcal{T}_T$) of columns in our dataset is shown in Figure 3.5. As expected, the most common place types found in the columns of our table corpus are populated places (i.e., cities) and administrative areas (i.e., countries, states, provinces, counties, etc.). Other, less common place types we observed include schools; airports; country, state/province, and region capitals; continents; hospitals; and rivers and streams.

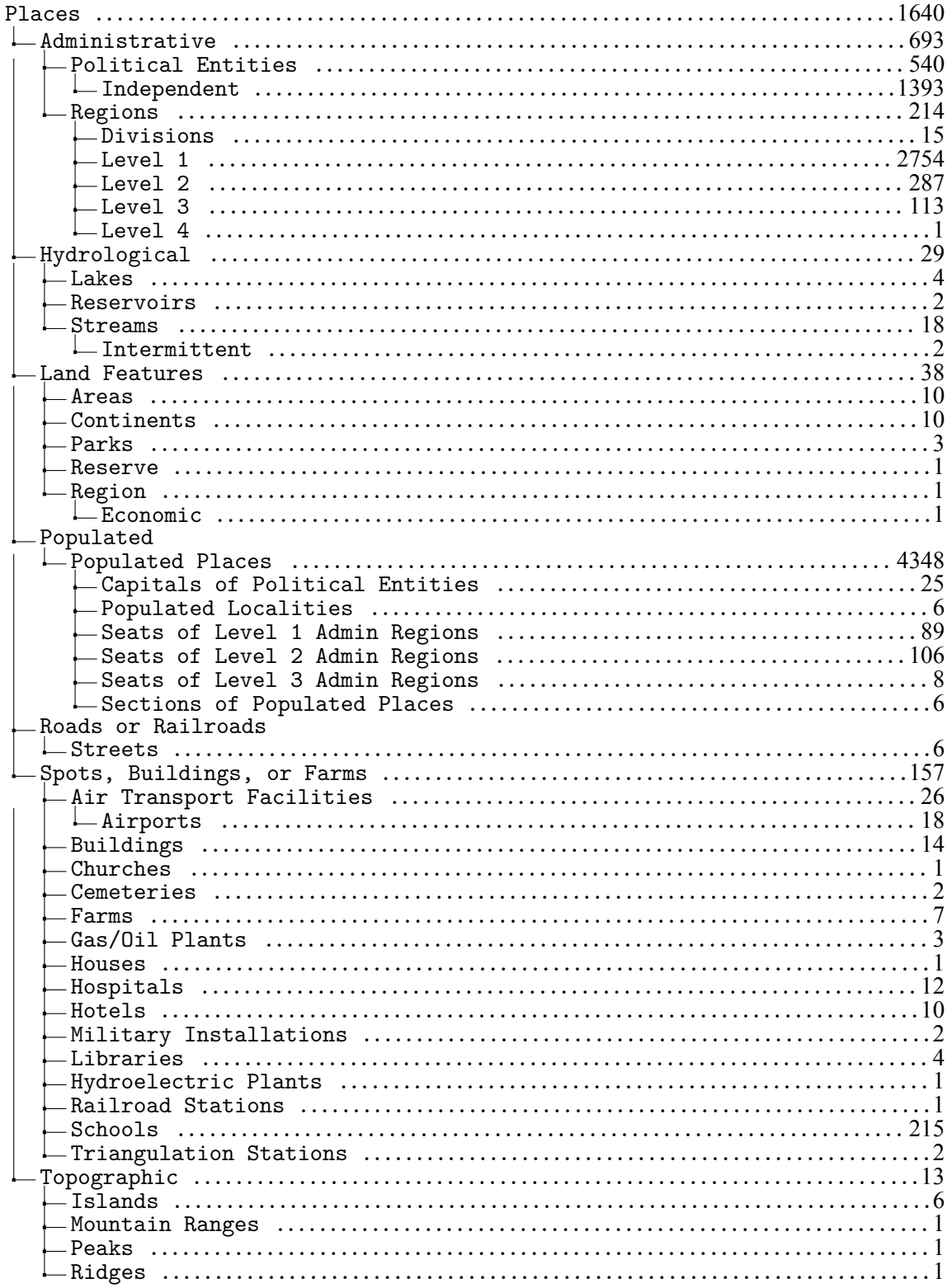


Figure 3.5: Place type distribution over nodes in $\mathcal{T}_T$ in the categorized table dataset. Values on the right indicate the number of columns that included the corresponding place type as part of their assigned category.

Table 3.1: Dataset characteristics.

	Spreadsheets	HTML
Documents	20,000	20,000
Data column count	234,776	108,795
Data row count	11,984,929	992,309
Data cell count	122,291,632	5,053,901
Geographic tables	5,117	3,305
Geographic columns	7,072	5,789

Many columns were classified with non-leaf types, which occurs when places in the column have a variety of specific types. For example, in American Baseball, there are some teams that represent states (e.g., Texas and Colorado, which are identified as administrative regions in GeoNames) and others that represent cities (e.g., New York and Chicago, which are identified as populated places), so a column containing these values would be categorized using the root node of the place type hierarchy. The root node represents generic places and was part of the assigned category for 1,640 columns according to our classifier. Another example of a generic place column is from a spreadsheet in our corpus that describes the itinerary of a trip to China, containing cell values such as “Great Wall of China”, “Beijing”, “Yangtze River”, and “Shanghai”.

Many place types are not found or are found only rarely in our table dataset. In most cases, these place types are relatively rare in Web tables, but in others the places are commonly represented in shorthand that is not found in GeoNames and consequently they are not identified by our algorithm. For example, some tables in our dataset contain information about National Parks, but use the park name without the qualifier, for example, “Yosemite” instead of “Yosemite National Park”, and the GeoNames gazetteer does not contain this as an alternate name.

A large number of distinct nodes in the geographic container hierarchy $\mathcal{T}_G$ were present in the categories assigned to columns in our dataset. In total, 361 different geographic con-

tainers were chosen. The full distribution is too large display in full, but we note that 39.7% of all geographic containers were "Earth", 9.8% were at the continent level, 41.6% were at the country level, 7.4% were at the state/province level, and 1.5% were at the country/region level. One hundred seventeen countries were part of at least one geographic container, so the dataset was geographically diverse.

The final component of our taxonomy, the $\mathcal{T}_P$ prominence hierarchy, contains only 10 nodes (since the population of the world is currently less than 10 billion and the prominence hierarchy nodes are keyed to powers of 10), with the distribution shown in Figure 3.6.

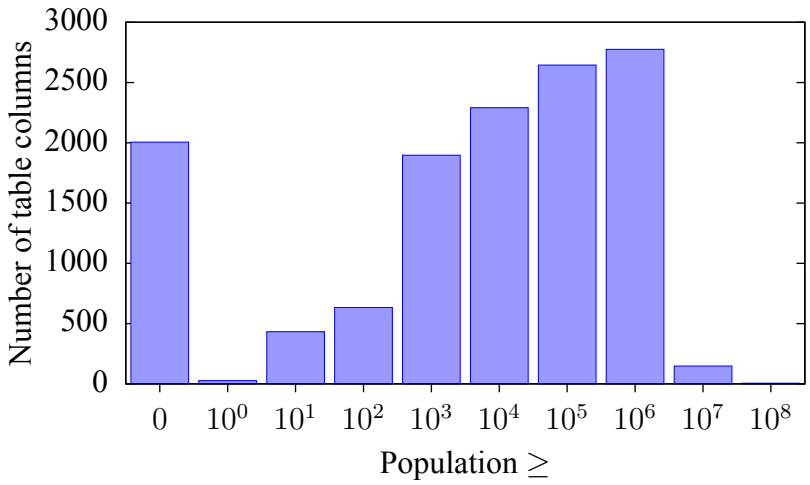


Figure 3.6: Prominence distribution over nodes in $\mathcal{T}_P$ in the categorized table dataset.

3.3.2 Category Accuracy

Next, we sampled from the full dataset to obtain a smaller dataset that could be hand-annotated by human judges and evaluated for correctness. To ensure that a wide variety of geographic columns were evaluated, columns with a variety of place types were chosen randomly. In total, 200 columns were chosen for evaluating the category classifications, balanced over different areas of the place type dimension $\mathcal{T}_T$. Fifty columns each were chosen from the following groups:

- **ADM**: Administrative Features (or a descendant),

- **POP**: Populated Places (or a descendant),
- **GP**: Generic Places (i.e., the root of $\mathcal{T}_T$),
- **OTH**: Other places types (e.g., schools, airports, etc.).

For each column/category, a human judge determined whether the category that was returned by our algorithm matched the values in the column. Results are shown in Figure 3.7. Overall, 148 of the 200 columns were correctly categorized, with 2 mis-categorized, and 50 that were wrongly chosen as geographic columns. Columns that were categorized as Administrative or Populated Places achieved the best accuracy rates, with 49 (98%) and 45 (90%) of the columns given the proper categories, respectively.

Since the four groups make up different portions of the full table dataset, we can extrapolate the overall accuracy rate by incorporating the relative prevalence of each group throughout the larger dataset. We measured the fraction of the full dataset that was categorized into each of the four groups, and found that 46.7% of column categories are administrative (ADM), 35.7% are populated places (POP), 12.8% are generic places (GP), and 4.8% are other place types (OTH). Weighting the accuracy results by these proportions lets us estimate an overall accuracy rate of 88.9%.

Most errors were due to non-geographic columns being assigned a place category, which suggests that we can improve upon our toponym recognition phase to filter out words that are ambiguous. The most common non-geographic words that were interpreted as toponyms were proper names of people (15 columns). However, the results for columns that did contain toponyms were promising, with a total of only two incorrect categories out of 148 columns.

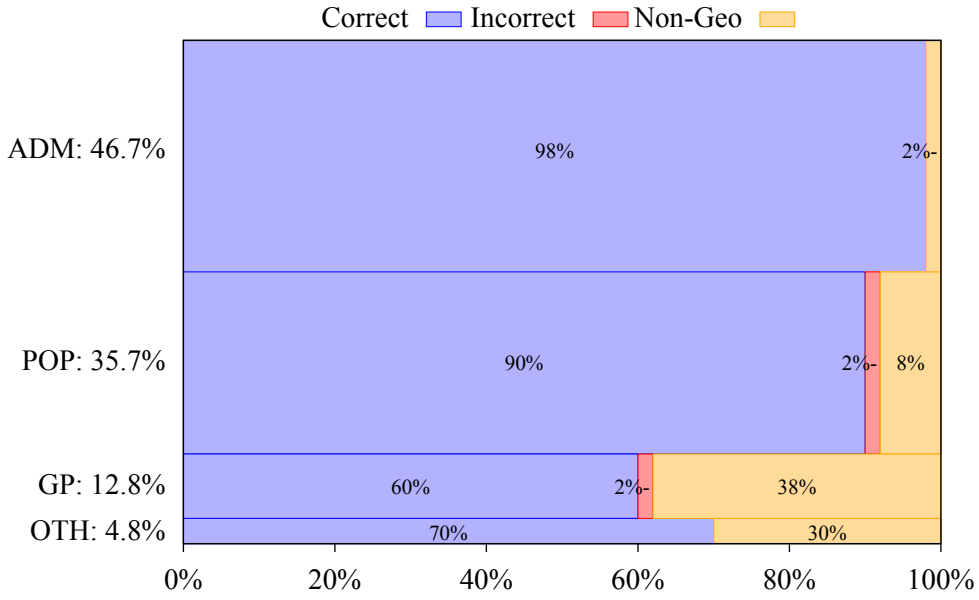


Figure 3.7: Accuracy of our algorithm for categorizing columns of toponyms. Bars are scaled horizontally to reflect the proportion of results within each group and scaled vertically to reflect the prevalence of each group within the full dataset.

3.3.3 Toponym Resolution Accuracy

Our final experiment measures our toponym resolution accuracy. From the 200 categorized columns analyzed in Section 3.3.2, we selected the 148 true geographic columns. From each column, one cell value was picked at random for inspection. A human judge was presented with the other values in the column for context and was asked to choose the most likely interpretation out of all available interpretations for the string that were present in the gazetteer. If none of the interpretations were valid, but the string was indeed a toponym, the string was marked as an "unmatched toponym". Or if the string was not a toponym, the string was marked as a "non-toponym".

The results were compared to the toponym resolution output of three algorithms. The first, Prom, considers only the prominence of the possible interpretations when resolving each toponym. The second, 2D, is a combination of three classifiers that each only uses two of the dimensions in our taxonomy $\mathcal{T}$. Each classifier is trained separately with a subset

of our feature set. To arrive at a resolution for toponyms, we choose the category with the highest likelihood value out of the three and pick the most prominent interpretation within that category. Finally, the third method, 3D, is our full method, which considers features from all three dimensions of the place taxonomy. The results of this experiment are shown in Table 3.2.

Table 3.2: Toponym Resolution Results.

Method	Accuracy
PROM	101/148 (0.682)
2D	130/148 (0.878)
3D	144/148 (0.973)

As expected, considering more dimensions improves the toponym resolution accuracy. The PROM method manages to resolve over two-thirds of the toponyms correctly, which is possible due to the large number of country, state, and metropolis occurrences in our dataset. The 2D variant improves upon this to achieve nearly 90% accuracy, since the addition of other attributes allows this method to recognize coherent types and geographically contained columns. Our full algorithm increases the accuracy rate further, where in all but 4 of the cases, the interpretation selected matched the interpretation that was assigned by our algorithm. This represents a 97.3% accuracy rate (144/148) for the toponym resolution task on geographic columns. In both columns that were assigned incorrect categories, the assigned interpretation did not match the ground truth. And in two other cases, the category was correct, but the toponym was still ambiguous within the category and a less prominent interpretation was the correct one (whereas out method chooses the more prominent interpretation in the face of ambiguity within a category). This result demonstrates the value of using our full taxonomy of hierarchical place categories for toponym resolution.

3.4 Demonstration Interface

We developed a browser-based DHTML interface for demonstrating the combined hierarchical place categories method for geotagging columns of toponyms. Users can submit place lists or tables and explore the likely categories returned by the classifier and their associated toponym interpretations. A sample session is shown in Figure 3.8, in which the toponym list is on the left and the corresponding output of the category classifier is shown to the right. The category listing includes the plain-English description of each category, along with the category coverage and ambiguity values over the input strings and the normalized likelihood that the category was the intended category for the input. In this case the results show that the most likely category for the toponyms in the input list is "populated places with population $\geq$ 1,000 in Florida, United States". This describes a consistent set of interpretations for the input list *Orlando*, *Conway*, *Lockhart*, *Oakland*, and *Oak Ridge*. However, alternative interpretations are possible, such as the second most likely category returned, "cities with population $\geq$ 10,000 in the United States".

3.5 Summary

We introduced and studied the utility of combined hierarchical place categories for identifying and resolving toponyms in structured datasets. Lists and table columns containing spatio-textual references can be difficult to geotag correctly because standard contextual clues, such as geographic containers, are sometimes omitted when the table author expects the interpretation of the references to be clear from context. However, making use of the context that is present in a list or column of similar places has not been thoroughly studied before. Here, we take the approach that the common thread of a list of toponyms can have varying specificity over multiple dimensions, namely the place type of the locations, their geographic container, and their prominence. To address this, we showed how to construct a list of possible categories that can be used to describe the list of toponyms, along with sev-

eral measures of each category's applicability. A Bayesian classifier is used to identify the most likely category based on observations made from a training data set. Our experimental analysis shows that the algorithm is effective at categorizing and resolving toponym lists that come from a large dataset of tables from the Web.

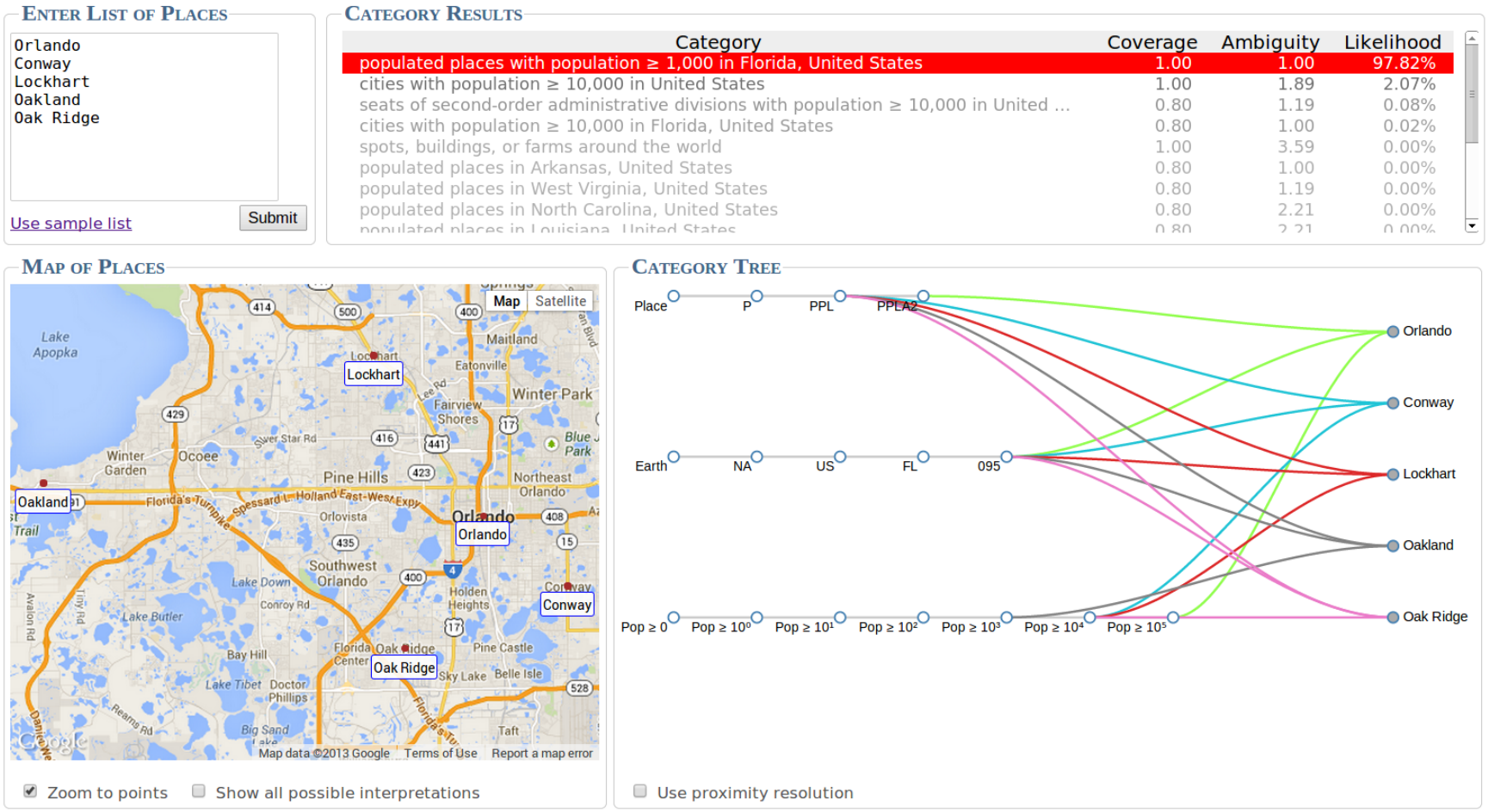


Figure 3.8: Screenshot of demonstration system for geotagging using combined hierarchical place categories. (1) User enters lists of similar places in the top left text box. (2) The system returns a ranked list of the most likely categories for describing the list, shown in the top right. (3) User explores interpretations within a selected category's constraints, using both a tree visualization of the categories in the bottom right and a map interface in the bottom left. Node labels in the category tree are abbreviated to avoid overlap and full labels are displayed when the user hovers the mouse pointer over a node. Additional disambiguation and interface options are available using the lower checkboxes.

Chapter 4

Itinerary Recognition

Itineraries come in many formats and presentation styles, making it challenging to identify them and distinguish them from other data tables that contain listings of place names. Such a decision is necessary because clearly there are many geographic datasets online that include place names, but which do not intend those place names to be interpreted as a series of stops in an itinerary. For example, tables containing demographic datasets or listings of customer addresses fall into this category. For our purposes, the term *itinerary* describes a table containing places which are intended to be visited in the listed order, while a *non-itinerary* is a table which does not have this property. We formalize the problem of identifying itineraries as follows:

Definition 1. Let L be the set of all valid latitude / longitude locations. Then, given an ordered collection $I = l_1, l_2, \ldots, l_n$ of locations $l_i \in L$, the **itinerary decision problem** (IDP) is to determine whether I represents an itinerary.

Unfortunately, the problem is difficult to solve accurately, even for humans, so the expected confidence in an algorithm's solutions must be tempered. However, as we show in Section 4.1, there are reasonably effective means of addressing this problem, even when the only availably indicators are lists of geographic coordinates.

We can re-formulate the problem to include additional context along with each location,

given that our source documents are tables and spreadsheets, not simple lists of geographic coordinates.

Definition 2. Let T be a table containing an ordered set of relations $r_1, r_2, \ldots, r_n$, where each relation r_i has an associated location l_i. The **table itinerary decision problem** (TIDP) is to determine whether T represents an itinerary.

Both the location-only IDP and context-inclusive TIDP can be addressed with statistical and machine learning methods, by incorporating several indicators that have a correlation to the outcomes of the decision problems. The following types of features are included in our implementation.

- Efficiency of stop ordering (applies to IDP and TIDP). In general, travel itineraries are designed with some constraints on the time and effort required to travel between all the stops, which results in nearby stops being visited consecutively. In place listings where spatial relationships are not taken into account, the expected length of an itinerary visiting each place in order will be distributed according to the total travel length required to visit those places in a random order.
- Returning to the start (IDP and TIDP). Itineraries are frequently "round-trips" where the starting and ending locations are the same.
- Ordering columns (TIDP only). Itinerary tables frequently contain an ordering columns such as the date that the corresponding location will be visited, or an ordinal number representing which day within the trip the location will be visited.
- Presence of travel terminology (TIDP only). Some words and phrases are commonly found in itineraries (e.g., the text "at sea" appears often in itineraries for cruise ships) and can serve as indicators of the subject of the document.

The diagram in Figure 4.1 shows the processing pipeline for our itinerary extractor. Documents are initially taken from a Web crawl and all tables are extracted to an abstract table

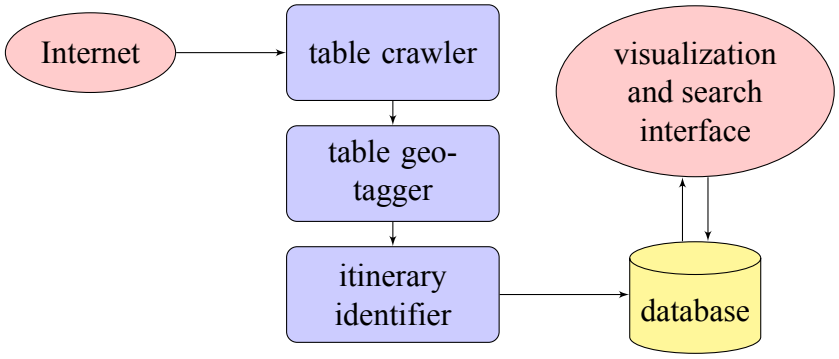


Figure 4.1: An itinerary processing pipeline.

format. The table geotagger identifies place references in table rows and assigns geographic interpretations to them. Our primary focus is the next phase, the itinerary identifier, where we classify geographic tables as either itineraries or non-itineraries. An itinerary search system could use the results of this phase to enable browsing and searching over a large database of itineraries, allowing users to visualize and compare itinerary options, like those shown in Figure 4.2. Targeting tables for itinerary retrieval has the added benefit that metadata for each stop (such as the date of the stop, any activities performed there, and lodging or transportation information), if present, is easy to associate with the stop since it is likely to appear in the same table or spreadsheet row as the stop's location name.

4.1 Itinerary Recognition

4.1.1 Importing and Geotagging Tables

We employ the table extraction method from Chapter 2 to identify the data and header sections of candidate tables. Next, the Combined Hierarchical Place Categories method of Chapter 3 is used to identify geographic tables and assign geographic interpretations to rows containing toponyms. We modify the geotagging method slightly for this application, using a different "tie-breaker" procedure in situations where multiple interpretations of a toponym exist within a list's assigned category, in order to emphasize the geographical coherence of itineraries. Instead of selecting the most highly populated interpretation, we select the

Figure 4.2: Three sample Italian vacation itineraries found on the Web.

interpretation that is nearest to the geographic centroid of the other toponyms' interpretations. In cases where multiple toponyms have ambiguous interpretations within a category, we use a greedy approach that iteratively selects interpretations closest to the geographic centroid of all already-selected interpretations.

4.1.2 Identifying Itineraries

The primary concern of this research is identifying itineraries from among the vast array of geographic tables and spreadsheets. This identification step is necessary because, while the output of the table geotagger is a collection of geographic tables along with interpretations of their place references, the vast majority of these tables are not intended to be itineraries — rather, they are tables that include entities with geographic attributes, not a travel path. Examples of non-itinerary geographic tables are demographic tables, sports team

standings, or listings of people that include a column containing each person's hometown. Many itineraries share common characteristics with non-itineraries, but the characteristics, when viewed as a whole, allow us to discern itineraries from non-itineraries in many cases.

Figure 4.3 shows fragments of several representative tables that were found by our table crawler and determined to include geographic columns. In this example, the tables share similarities in terms of column headings, data types of nearby columns, and place name formatting. In this case, the table on the left is not intended as an itinerary, which becomes more evident when viewing the plotted locations from each table in Figure 4.5. The fact that the left and right tables share several textual similarities (such as the "Date" and "Location" column headers and the comma-based place name formatting), but only one is an itinerary, suggests that rule-based methods or methods that rely on column header text or data types of nearby columns will have difficulties making the determination. Further, the difference between the mapped visualizations of the tables led us to believe that spatial analysis of the tables was an important component in accurately addressing the TIDP.

Using these observations, we developed several heuristics to act as indicators for a machine learning classifier. The most useful indicators are based on the observation that itineraries tend to be fairly efficient at visiting stops, in comparison to an ordering of the stops that is not based on their spatial proximity. This is due to the fact that trip planners take costs of transportation and travel time into account. In particular, the tendency to prefer a shorter ordering of stops is measurable by comparing the route length of the original route to that of an alternate route that visits the same stops, but in a different order. Instead of comparing with a globally optimal route, which is intractable to compute for even relatively short itineraries (since the TSP is NP-hard) and does not model true travel itineraries, we use an interchange procedure that underlies the commonly-used 2-opt method for generating approximate solutions for the traveling salesman and other optimization problems [28]. Figure 4.4a shows an example, where an alternate permutation of the location list could reverse

Date	Location	Delivery #
12/16/04	Oestrich-Winkel, DE	20031
03/17/05	Lavera, FR	20053
03/17/05	Lavera, FR	20054
04/27/05	Marl, DE	20065
05/25/05	Beringen, BE	20104
06/23/05	Schwechat-Mannswörth, AT	20112
09/08/05	Dordrecht, NL	20131
11/21/06	Litvinov, CZ	20142
11/10/05	Pasir Gudang, Johor, MY	20152
11/10/05	Pasir Gudang, Johor, MY	20153
12/14/05	Antwerpen, BE	20177
11/16/05	Tehran, IR	20179
12/19/05	Brüssel, BE	20183
01/19/06	Torre Boldone (BG), IT	20186
01/19/06	Torre Boldone (BG), IT	20187
...	...	...

Day	Dest	Activities
1	Vienna	Hotel check-in
2	Vienna	City tour
3	Vienna	Transfer to Budapest
	Budapest	City tour
4	Mohacs	Pecs excursion
	Villany	Wine tasting
5	Vukovar	Yugoslav Civil War tour
	Novi Sad	Walking tour
6	Belgrade	City tour
7	Iron Gates	Full day cruising
8	Vidin	Belogradchik excursion
9	Giurgiu	Palace of Parliament
10	Rousse	Disembarkation
	Plovdiv	Walking tour
11	Erdine	Lunch stop
12	Istanbul	City tour
13	Istanbul	Tour Topkapi Palace
14	Istanbul	Return flight home

Date	ETA	Location	Notes
9/19/07	8:00	Splendora FBC	Depart
	10:11	Nacogdoches, TX	Gas Stop
	12:09	Marshall, TX	Gas Stop & Lunch
	14:51	Texarkana, AR	
	15:22	Hope, AR	Gas Stop
	15:57	Gum Springs, AR	
	16:23	Arkadelphia, AR	Stop
9/20/07	7:30	Arkadelphia, AR	Depart
	7:39	Caddo Valley	Gas
	11:16	Dardanelle, AR	Gas Stop
	13:06	Jasper, AR	Lunch
	14:26	Dogpatch USA	Scenic/Photos
	14:42	Harrison, AR	Gas Stop & Lunch
	16:33	Francis, AR	
	16:49	Eureka Springs, AR	Stop & Gas
9/21/07	9:00	Eureka Springs, AR	Depart
	10:48	Ozark, AR	
	11:17	Van Buren, AR	Gas & Lunch
	12:53	Fort Smith, AR	
	12:55	Entering Oklahoma	
	15:10	Sunset Corner, OK	
	16:04	Entering Arkansas	
...	...	...	...

Figure 4.3: Portions of tables containing possible itineraries.

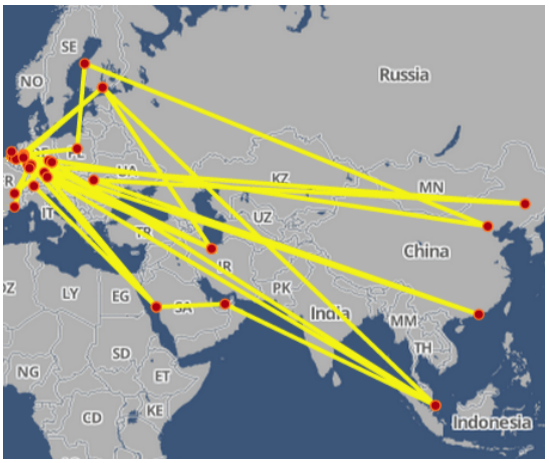
(a) path $abcd$ (b) path $acbd$

Figure 4.4: Itineraries generally follow efficient routes. For this example, we expect that an itinerary visiting locations a, b, c, and d is more likely to visit them in the order $abcd$ (shown in (a)) than the order $acbd$ (shown in (b)). Conversely, tables containing places that are ordered efficiently are more likely to be itineraries than tables containing inefficient place orderings.

the order of stops b and c. As shown in Figure 4.4b, this results in a longer total route length than the original, so is less likely (though still possible) to be chosen as part of an itinerary. We call location lists with many pairs of points whose reversal results in longer path lengths *locally efficient*, meaning that the listing could not be made into a shorter route by simply re-arranging neighboring stops. Similarly, location lists with many sequences of points whose reversal results in longer path lengths are said to be *generally efficient*.

The edge interchange procedure is the basis for two efficiency measures that we use as features in our itinerary identification algorithm. The first, ϵ_1, measures efficiency at the local level — essentially counting how many consecutive pairs of stops are in the order that results in the shortest path. The second, ϵ_2, measures stop order efficiency over longer sequences of locations — counting subsections of the full stop list that could be more efficiently reconnected to the remainder.

To formalize our concepts of efficiency, we define a preliminary indicator function. For an ordered set of locations $L = l_1 l_2 \ldots l_n$, indexes $i, j \leq n$, and $d(l_i, l_j) =$ great circle



(a)

(b)

(c)

Figure 4.5: Visualizations of the tables from Figure 4.3 as itineraries. While the column headers and cell types of the left and right tables are similar, the topology that results from treating each table as an itinerary makes it clear that (a) is unlikely to be an itinerary, while (b) and (c) are both likely to represent itineraries. In fact, the table visualized in (a) came from a listing of shipments for a company that is certainly not intended as an itinerary. The table visualized in (b) contains the schedule for a river cruise through eastern Europe and the table visualized in (c) is the schedule for a motorcycle club's ride through several states in the U.S.A., which are both itineraries.

distance between l_i and l_j, let

$$\delta_{i,j}(L) = \begin{cases} 1 & \text{if } (d(l_i, l_{i+1}) + d(l_j, l_{j+1})) \leq \\ & \quad (d(l_i, l_j) + d(l_{i+1}, l_{j+1})) \\ 0 & \text{otherwise.} \end{cases} \tag{4.1}$$

The $\delta_{i,j}$ value indicates whether the combined lengths of the edge from l_i to l_{i+1} and the edge from l_j to l_{j+1} is shorter than (or equal to) the combined lengths of edges with swapped endpoints, l_i to l_j and l_{i+1} to l_{j+1}. Equivalently, this indicates whether a permutation of the location list that reverses the order of locations $l_{i+1} \ldots l_j$ has a shorter overall path length than the initial permutation. If not, we say that the subpath from l_i to l_j is *reasonably ordered*, because a trivial reversal of the subpath order does not shorten the overall path length. We use this to define two efficiency measures based on the fraction of subpaths that are reasonably ordered. For itinerary recognition, these definitions do not consider subpaths that include the start and end points, because for many trips these points are fixed.

- **Local efficiency** is the fraction of consecutive stop pairs whose reversal would lead to a longer total route distance. That is, for locations $L = l_1 l_2 \ldots l_n$,

$$\epsilon_1(L) = \frac{1}{n-3} \sum_{i=1}^{n-3} \delta_{i,i+2}(L). \tag{4.2}$$

- **General efficiency** is the fraction of all unique, non-consecutive edge pairs that would result in a longer total route if their endpoints were swapped. For locations $L = l_1 l_2 \ldots l_n$,

$$\epsilon_2(L) = \frac{1}{\binom{n-2}{2}} \sum_{i=1}^{n-3} \sum_{j=i+2}^{n-1} \delta_{i,j}(L). \tag{4.3}$$

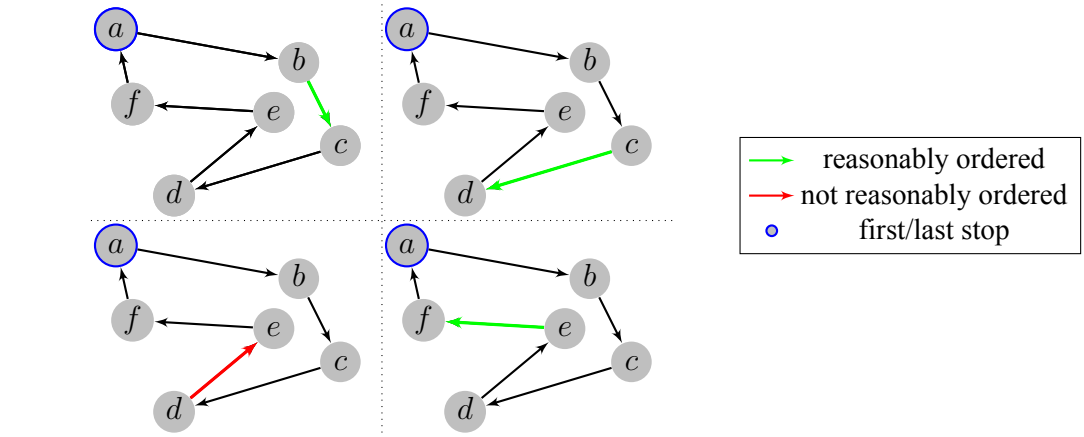
Each efficiency formula counts the number of valid swaps that result in a longer total path length, which is then normalized by the total number of valid swaps. For exam-

ple, assume we have a table containing five locations ($L = l_1 l_2 \ldots l_5$). Then $\epsilon_1(L) = \frac{1}{2}(\delta_{1,3}(L) + \delta_{2,4}(L))$ and $\epsilon_2(L) = \frac{1}{3}(\delta_{1,3}(L) + \delta_{2,4}(L) + \delta_{1,4}(L))$. An example with more stops is demonstrated in Figure 4.6. Here, there are seven stops (this is a round-trip route, so a is counted twice). Of the four subpaths with length two that do not connect to the first/last stop, three are reasonably ordered, resulting in a local efficiency value $\epsilon_1 = 0.75$. General efficiency considers subpaths of length two or greater, of which there are ten, with eight reasonably ordered, resulting in $\epsilon_2 = 0.8$. Although ϵ_1 and ϵ_2 take on similar values for this path, this is not always the case.

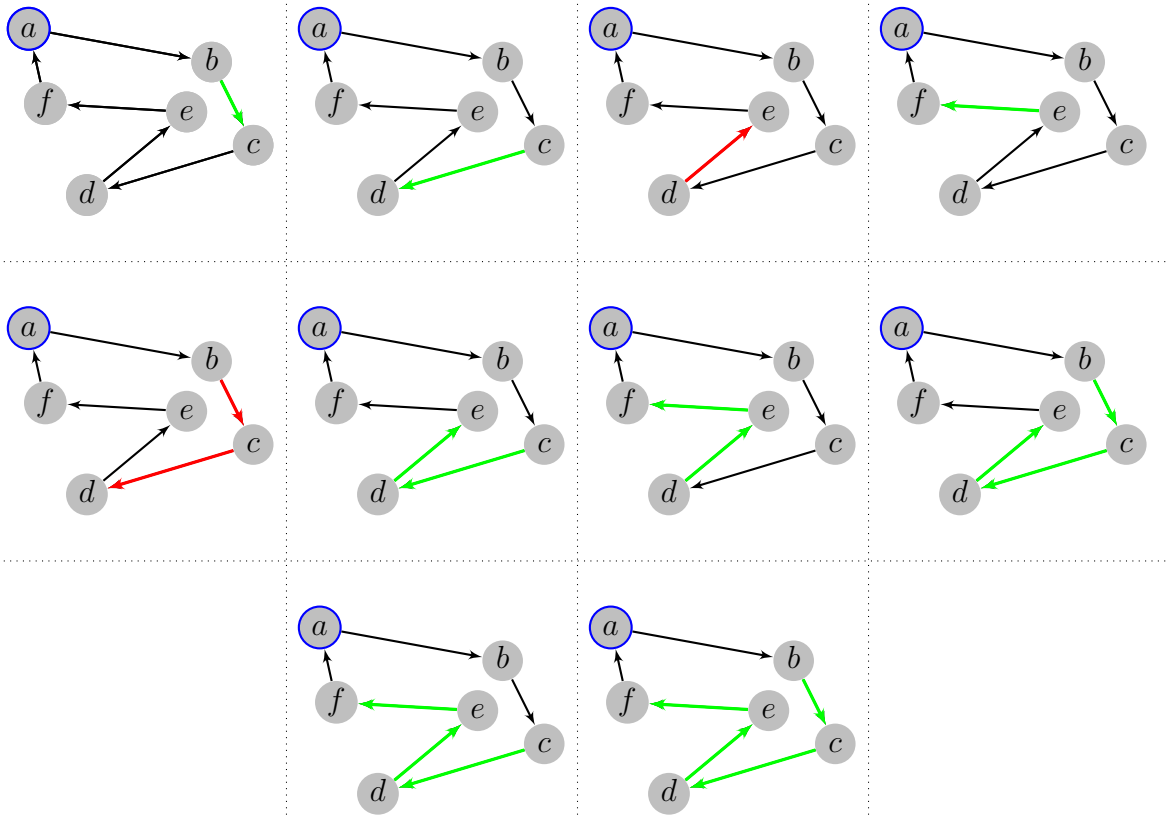
Using terminology from TSP literature [68], if reversing any sub-sequence of L results in a longer path length, we say that L is *2-optimal*. From the definition of ϵ_2, we can say that L is 2-optimal if and only if $\epsilon_2(L) = 1.0$. In general, the goal of the efficiency measures is to quantify the presence of efficient stop ordering. Consequently, we expect itineraries to exhibit high efficiency values (although values less than one are expected, given that many travel itineraries do not follow optimal paths), while non-itineraries will tend to have moderate efficiency values clustered near 0.5.

In addition to efficiency measures, we also use several other features when deciding whether or not a given table represents an itinerary. These include the following ordering features and text features.

- $f_r(t) = 1$ *iff the primary location column of the table includes the same location in the first and last positions*. We call this a *round trip* table and expect that round trip tables will be more common in itineraries than non-itineraries.
- $f_{od}(t) =$ *# of ordered date/time columns found in the table*. Since itineraries are temporal objects, itineraries in tables commonly include a date/time column.
- $f_{on}(t) =$ *# of ordered numeric columns found in the table*. While ordered numeric columns are a component of some itinerary tables (such as the center table in Fig-



(a) Local efficiency



(b) General efficiency

Figure 4.6: Subpaths examined for efficiency measures of example path. In (a), three of four subpaths of length two are reasonably ordered, so $\epsilon_1 = 0.75$. In (b), eight of ten subpaths of length greater than or equal to two are reasonably ordered, so $\epsilon_2 = 0.8$.

ure 4.3), they are also common in non-itineraries. We expect this feature to have a smaller effect on accuracy than the others.

- $f_a(t) =$ *# of text columns found in the table that are sorted alphabetically*. Unlike the previous two ordering features, we expect that tables containing alphabetically sorted columns are unlikely to be itineraries, since it is rare for a table to be arranged both spatially and alphabetically.

- $\vec{f}_t(t) =$ *a term vector of words commonly found in itineraries*. Currently, we use a list of 40 words and phrases that we found to have the highest difference in their TF/IDF values in itineraries versus non-itineraries. Such terms include "itinerary", "trip", "travel", "airport", "hotel", "cruise", month names, and others.

To account for the loose constraints inherent in manually generated tables, columns are treated as ordered or alphabetic if at least 90% of the values in the column are greater than or equal to the preceding value or at most one value is out of order in columns with fewer than 10 values.

We construct a feature vector $\vec{f}(t)$ for each table t using the features listed above, and then apply a binary classifier to compute Pr(t is an itinerary$|\vec{f}(t)$). Given that our collection of features includes a variety of feature types (fractional, binary, integer, and term vector), this is not a clear fit for any one specific machine learning classification model, so we examine three: (i) a Naive Bayes classifier [33], (ii) a decision tree [17, 89], and (iii) a support vector machine [26]. We preprocess each feature based on the expected input format for each specific classification model, giving binary features to the Naive Bayes classifier, raw numeric values to the decision tree, and standardized (mean- and variance-adjusted) values to the support vector machine.

4.2 Evaluation

4.2.1 Dataset

The tables for our evaluation were taken from a two million page Web crawl that targeted Microsoft Excel spreadsheets and HTML pages containing tables. Similar to the corpus creation procedure in previous chapters, we seeded the crawl with search results for queries of the form "⟨*data term*⟩ ⟨*geo term*⟩ ⟨*random term*⟩ ⟨*filetype*⟩". Each term was randomly selected from a hand-selected set of values or omitted, as our goal was to use a range of queries to uncover a wide variety of documents. The data term was randomly chosen from a list of terms that are often found in documents containing tables (such as "table", "stats", etc.). The geo term was randomly chosen from a large collection of place names found in the GeoNames gazetteer. The random term was a letter, number, or both, chosen at random, which is used to induce a variety of results for a static combination of the other terms. And the filetype component was set to "filetype:xls" or "filetype:xlsx" to search for Excel spreadsheets, or omitted to search for HTML documents containing tables. Statistics for the full table corpus are shown in Table 4.1. A table extraction module removed tables that were not found to be data tables (also known as "true" or "relational" tables), resulting in 662 thousand documents. Since some HTML documents contain multiple tables, and spreadsheets can likely contain multiple worksheets, the aggregate number of data tables in our corpus was 2.1 million. After running our geotag module to locate toponyms in the tables and assign interpretations to them, we obtained a set of 130 thousand documents containing 235 thousand tables. The geographic tables contained many more rows on average, as there were around 53 cells per column, compared to 28 cells per column in the full dataset.

The evaluation was performed using a subset of the full corpus where each table was manually annotated as either an itinerary or non-itinerary. For a table to qualify as an itinerary, there must be implied travel along the edges between consecutive pairs of places. This defi-

nition results in several tables being called itineraries that would not be considered itineraries for the purposes of a sightseeing trip, but which have the implied-edge property, such as a listing of exits along a section of highway or stops made by a regional train. For our purposes, these are all types of itineraries.

In all, we annotated 300 tables as either itineraries or non-itineraries. The first 200 were selected at random from our full dataset, of which only 3 were true itineraries. The next 100 were chosen from tables with a large number of stops ($n \geq 10$) and a high efficiency value ($\epsilon_1 \geq 0.8$) to ensure an adequate number of itineraries were included in the evaluation corpus (the number would otherwise be low due to the sparsity problem mentioned earlier). Later in this chapter, we account for the non-random sampling by scaling measurements based on the relative frequency of similar efficiency values within the full dataset. Of the 300 annotated tables, 60 were classified as itineraries, and 240 were classified as non-itineraries. The itineraries had a mean number of stops of 29 and a median of 22, while non-itineraries had a mean of 27 and a median of 14.

4.2.2 Itinerary Detection

Our evaluation of itinerary detection involved analyzing (i) the discriminatory power of the efficiency measures, (ii) the overall accuracy of our itinerary detector, and (iii) the contribution of individual features to classification accuracy.

The observed probability density functions of the efficiency measures are shown in Figure 4.7. The curves are smoothed using kernel density estimation [90] to reveal trends (and to avoid uninformative peaks at common fractional values such as 0.5, 0.75, $0.666\ldots$, etc.). Figure 4.7a shows the estimated distribution of ϵ_1 values for itineraries and non-itineraries in our training set. The estimates were calculated by scaling each ϵ observation by the relative frequency of similar efficiency values within the full dataset.

Evaluation of each classifier on the table itinerary decision problem was performed

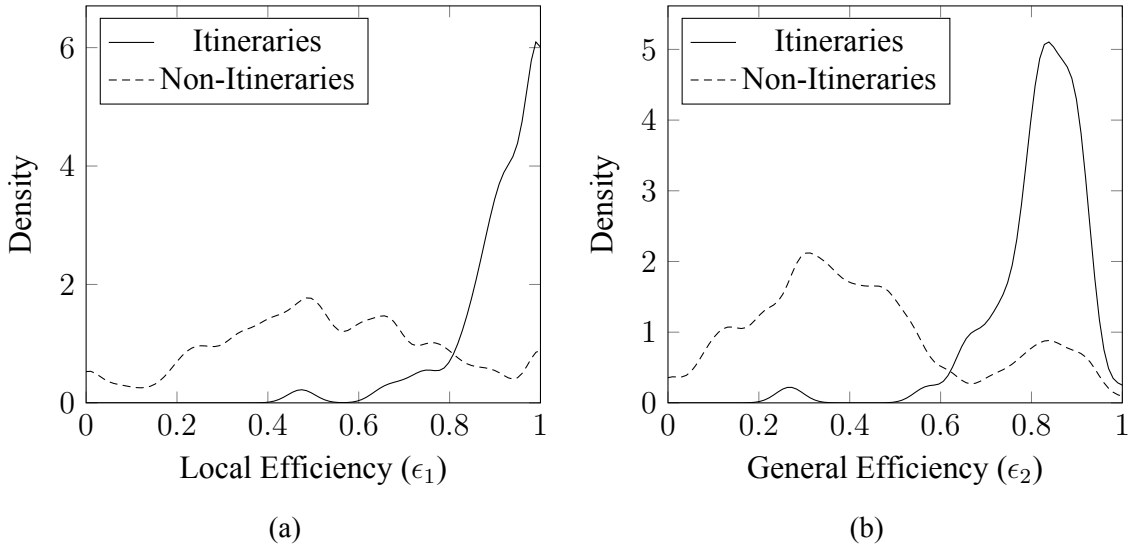


Figure 4.7: Density of the (a) ϵ_1 and (b) ϵ_2 measures. As shown, itineraries are much more likely to obtain high ϵ_1 values (> 0.8) than non-itineraries. The vastly different curves suggest that the local efficiency measure is a useful feature for distinguishing between itineraries and non-itineraries. Similar to the distributions for local efficiency (ϵ_1) values, itineraries are much more likely to have high ϵ_2 values than non-itineraries.

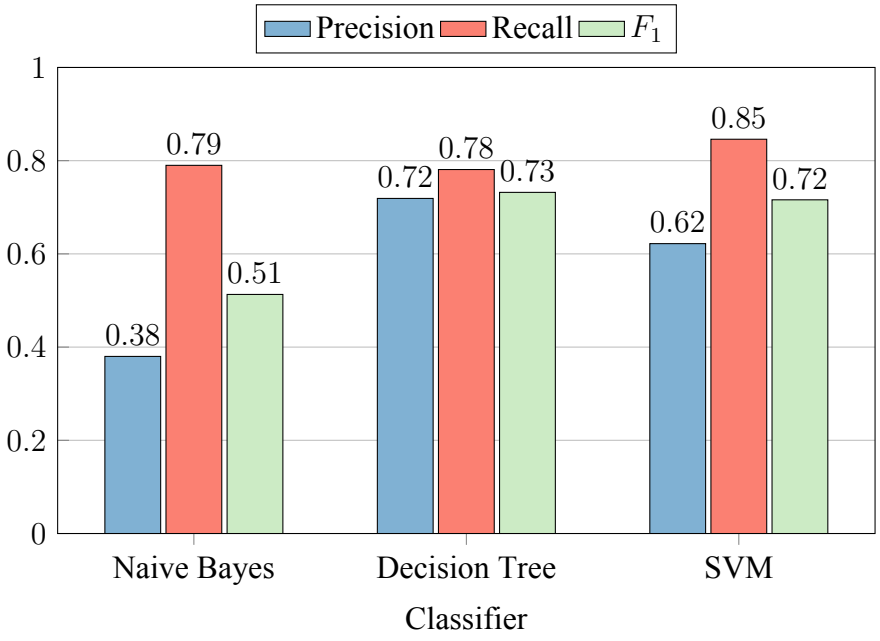


Figure 4.8: Precision, recall, and F_1 scores for each of the candidate classifiers on the itinerary identification task. The decision tree classifier achieves the highest F_1 score, followed by the SVM and Naive Bayes classifier.

Table 4.1: Dataset characteristics

Full Dataset	
Documents	2,000,000
containing data tables	662,511
Data tables	2,128,032
Columns	10,142,785
Cells	280,170,694
After removing non-geographic tables	
Documents	130,294
Data Tables	235,433
Columns	1,527,890
Cells	80,432,927

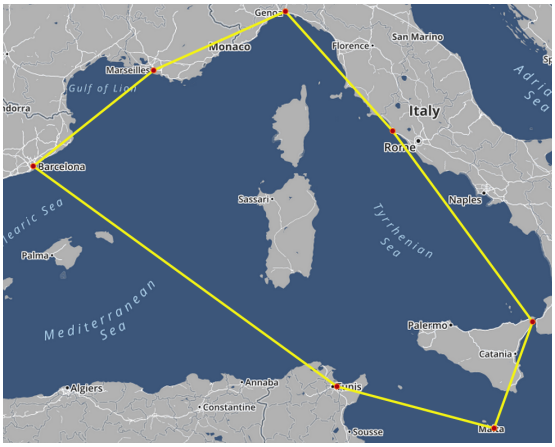
using five-fold cross validation against the annotated data set. For each classifier, we computed the average precision (P), recall (R), and F_1 score. Using T_P as the number of true positives (true itineraries correctly classified as itineraries), F_P as the number of false negatives (non-itineraries incorrectly classified as itineraries), and F_N as the number of false negatives (true itineraries incorrectly classified as non-itineraries), then $P = T_P/(T_P + F_P)$, $R = T_P/(T_P + F_N)$, and $F_1 = 2PR/(P + R)$. The results are displayed in Figure 4.8. The decision tree classifier achieves the best F_1 score of 0.73, perhaps due to strong interdependence between the features, which decision trees can exploit. This is followed closely by the SVM with an F_1 score of 0.72. The Naive Bayes classifier achieves the worst F_1 score based on a very low precision score.

Figure 4.9 shows two examples of correctly classified itinerary tables that appear commonly in our corpus—cruises and road trips. Other types of itineraries that appeared included tour dates and venues for bands and highway exit locations. As an example of the limitations of our method, two tables that were incorrectly classified by all three classifiers are shown in Figure 4.10. The first is a table of Dewey Decimal class numbers for books that focus on individual U.S. states. Interestingly, although this system for organizing books in a library pre-dates computers or computerized search systems, its choice of ordering leads to

a path used by many computer-based geographic indices: a space-filling curve. This efficient path leads to high values of ϵ_1 and ϵ_2, which cause the classifiers to deem the table an itinerary, incorrectly. Similarly, the second table is a listing of coastal Italian regions and various related statistics (only the coastline column is included in the figure). The ordering of regions is clearly influenced by their spatial location, but like the Dewey Decimal table, there is no implied edge between consecutive locations in the table, and it is therefore not an itinerary. The existence of tables such as these, which can be described as spatially-arranged non-itineraries, explains much of the classification error observed in our evaluation. This suggests that other spatial features or non-spatial features may be required to successfully detect and classify them as non-itineraries.

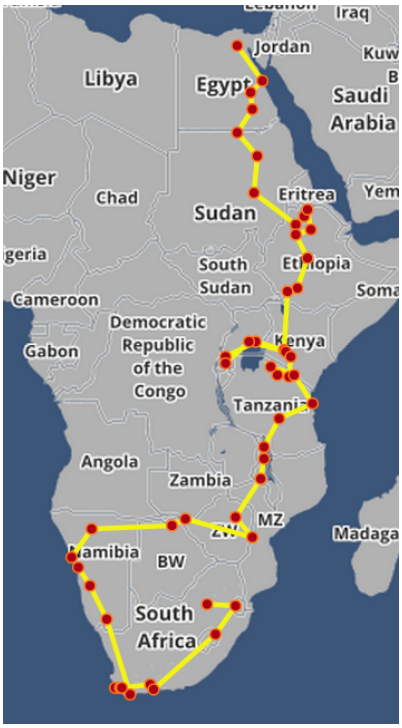
Next, we analyzed the contribution of individual features and combinations of features to the accuracy of the decision tree classifier (for the rest of this section, we use the decision tree classifier, as it was the top performer in classification accuracy). We ran the classification test repeatedly, while holding out individual features, and compared the results of each test to the results when all features were included and tabulated the results in Table 4.2. As expected, the classifier performed no better when features were removed, with the biggest change coming when we withheld the both efficiency measures. ϵ_1. The F_1 score in this case fell from 0.73 to 0.44, a drop of 0.29, which we call the marginal contribution of ϵ_1 and ϵ_2 to the F_1 score. This is a substantial difference in the F_1 score and suggests that the efficiency measures are quite discriminative, in ways that the other features are not. Somewhat surprisingly, the local efficiency score, ϵ_1 causes a much larger drop than the general efficiency score, ϵ_2, when withheld individually. We see two potential explanations for this. First, from a statistical perspective, the nature of the general efficiency measure may be less informative than the local efficiency measure, as the fraction of itineraries with $\epsilon_2 > 0.8$ is relatively lower, while the fraction of non-itineraries with $\epsilon_2 > 0.8$ is relatively higher. Second, from a data analysis perspective, it may be that the nature of itineraries leads to more

Day	Port	Arrival	Departure
Nov 30, 1999	Valletta, Malta	---	6:00 p.m.
Dec 01, 1999	Messina, Italy	7:00 a.m.	2:00 p.m.
Dec 02, 1999	Civitavecchia, Italy	8:00 a.m.	7:00 p.m.
Dec 03, 1999	Genoa, Italy	9:00 a.m.	6:00 p.m.
Dec 04, 1999	Marseilles, France	8:00 a.m.	6:00 p.m.
Dec 05, 1999	Barcellona, Spain	7:00 a.m.	2:00 p.m.
Dec 06, 1999	La Goulette, Tunisia	8:00 a.m.	6:00 p.m.
Dec 07, 1999	Valletta, Malta	9:00 a.m.	---



(a) Mediterranean cruise.

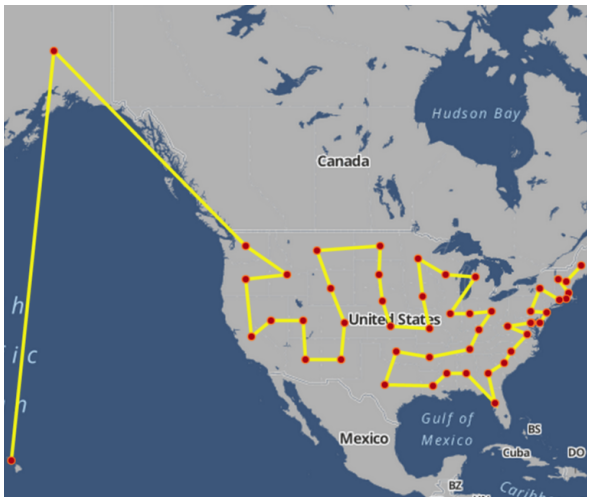
	Destination	Arrival
1	Cairo, Egypt	November 18, 2012
2	Safaga, Egypt	November 29, 2012
3	Luxor, Egypt	December 1, 2012
4	Aswan, Egypt	December 4, 2012
5	Wadi Halfa, Sudan	December 11, 2012
6	Abu Hamed, Sudan	December 14, 2012
7	Khartoum, Sudan	December 15, 2012
8	Gondar, Ethiopia	December 18, 2012
9	Simien Mountains, Ethiopia	December 20, 2012
10	Aksum, Ethiopia	December 23, 2012
11	Lalibela, Ethiopia	December 25, 2012
12	Bahir Dar, Ethiopia	December 27, 2012
13	Addis Ababa, Ethiopia	December 30, 2012
14	Arba Minch, Ethiopia	January 2, 2013
15	Jinka, Ethiopia	January 3, 2013
16	Nakuru, Kenya	January 8, 2013
17	Jinja, Uganda	January 11, 2013
18	Kabale, Uganda	January 13, 2013
19	Kigali, Rwanda	January 15, 2013
20	Volcans National Park, Rwanda	January 15, 2013
21	Kampala, Uganda	January 19, 2013
22	Lake Naivasha, Kenya	January 22, 2013
23	Nairobi, Kenya	January 24, 2013
24	Arusha, Tanzania	January 26, 2013
25	Ngorongoro Crater, Tanzania	January 28, 2013
26	Serengeti, Tanzania	January 29, 2013
27	Mount Kilimanjaro, Tanzania	January 30, 2013



(b) Road trip through Africa (table is truncated).

Figure 4.9: Two correctly classified itinerary tables.

Dewey Decimal Number	State
974.1	Maine
974.2	New Hampshire
974.3	Vermont
974.4	Massachusetts
974.5	Rhode Island
...	...



(a) Dewey Decimal classes for states in the U.S.A.

Region	Coastline
Imperia	62.7 km
Savona	80.5 km
Genova	109.2 km
Massa Carrara	13.0 km
Lucca	20.5 km
...	...

(b) Coastal Italian regions.

Figure 4.10: Two examples of misclassified tables. Both tables include lists of locations that are highly efficient by our definition, causing all three classifiers that we used in our evaluation to label them as itineraries. In (a), the Dewey Decimal system for book topic classification is shown, which orders states along a path that resembles a space filling curve. In (b), a listing of coastal Italian regions presumably follows a path with similarities to some Italian vacations, but is instead an exhaustive list of such regions and related coastal data.

Table 4.2: Feature evaluation

Feature	F_1 Without Feature	Marginal Contribution to F_1 score
ϵ_1	0.62	+0.11
ϵ_2	0.70	+0.03
ϵ_1 and ϵ_2	0.44	+0.29
f_r	0.71	+0.02
f_{od}	0.69	+0.04
f_{on}	0.72	+0.01
f_a	0.69	+0.04
$\vec{f_t}$	0.72	+0.01
non-efficiency	0.66	+0.07

local efficiency than general or global efficiency. That is, given the scheduling constraints that can shape itineraries, people may be inclined to travel efficiently for short periods, but not aim for a perfectly efficient route from start to finish. Such priorities would explain the disparate impact of these two efficiency features on our classification accuracy.

Other features all contribute to the performance of the classifier, with the ordered date column indicator f_{od} and the alphabetic column indicator f_a both contributing 0.04 to the F_1 score. The least impact is attributable to the ordered numeric column indicator, f_{on}, and the text vector, f_t, whose removal only caused a decrease of 0.01 in the F_1 score. This is somewhat surprising, since ordered numeric columns are much more prevalent in itineraries than non-itineraries. By manual inspection of the annotated table corpus, we see that 44% of true itineraries contain an ordered numeric columns, while they are found in only 15% of non-itineraries. This may be explained by the presence of temporal words in the term vector for $\vec{f_t}$, whose presence may offset the gains otherwise attributable to a numeric column. Still, the small differences in the impact of these features is overshadowed by the impact of the efficiency features.

Finally, we looked at the number of itineraries found in our full table corpus. The decision tree model classified 1,206 itineraries out of the 235,433 geographic tables in our

corpus, a total of 0.5%. This is consistent with our expectation that itineraries would be rare, but prevalent enough that a more complete crawl of the Web would result in a large quantity of itineraries to allow for map-based browsing.

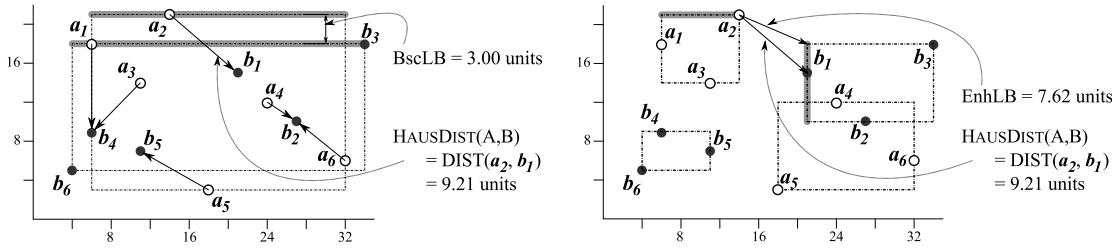
4.3 Summary

We have presented itinerary retrieval as a new area for geographic data extraction and implemented a pipeline of processing methods to evaluate our approach, which uses a machine learning classifier to decide whether a candidate table contains an itinerary. The core of our method involves computing spatial efficiency measures of the locations listed in a table, which match our notions of efficiency and were shown to have a substantial impact on the accuracy of our classifier.

Chapter 5

Queries on Extracted Point Sets

Geospatial attributes provide a rich domain of data which can support a plethora of queries. As the previous chapters have laid out, a collection of geotagged data tables from the Web can be treated as a repository of geographic point sets. There are many natural spatial queries to perform on such a repository, but here we focus on one that is particularly useful against this type of data. In particular, we look at a set of queries for finding "similar" point sets to a query point set (that could be specified geographically on a map interface or textually using place references that are then geotagged). For example, if the collection includes tables that list documented instances of diseases including locations, a user could run a query to identify k historical outbreaks with similar spatial distributions as a current outbreak. The query uses the Hausdorff distance measure, along with a variant called the modified Hausdorff distance, to describe the similarity (or more precisely, the dissimilarity) between two point sets. To avoid computing the Hausdorff distance for all point sets in the repository, we can compute an *optimistic estimate* (i.e., lower bound value) of the actual Hausdorff distance, and rule out point sets whose optimistic estimates are worse than other known values. As we show, the way in which the optimistic estimate is computed and its resulting accuracy can have a substantial effect on the execution time of queries. Consequently, we propose a method that produces a tighter estimate than a method based on the commonly-used minimum bounding rectangles (MBRs) and use this to develop a similarity search algorithm that



(a) BscLB is 32.6% of HausDist (b) EnhLB is 82.7% of HausDist

Figure 5.1: Illustrations of BscLB, EnhLB, and HausDist from A to B ($\{\boldsymbol{a_1}, \ldots, \boldsymbol{a_6}\}$ to $\{\boldsymbol{b_1}, \ldots, \boldsymbol{b_6}\}$)

efficiently finds point sets from the large collection that are similar to the query.

5.1 Background

We use the Hausdorff distance HausDist as a dissimilarity measure of a set A with respect to another set B. The Hausdorff distance $\textsc{HausDist}(A, B)$ can be regarded as the worst-case discrepancy of A with respect to B. Specifically, the distance $\textsc{HausDist}(A, B)$ is defined as the maximum value of the distance from each point $\boldsymbol{a}$ in A to its nearest point in B, i.e.,

$$\textsc{HausDist}(A, B) = \textsc{max}\{\textsc{min}\{\textsc{Dist}(\boldsymbol{a}, \boldsymbol{b}) : \boldsymbol{b} \in B\} : \boldsymbol{a} \in A\}. \tag{5.1}$$

Under this measure, a set A is considered similar to B if and only if each point in A is close to at least one point in B. We note that HausDist is asymmetric (and thus is not technically a distance function; however, we will refer to it as a distance function in the interest of brevity). We define a related symmetric relation $\textsc{SymHausDist}(A, B)$ as $\textsc{max}\{\textsc{HausDist}(A, B), \textsc{HausDist}(B, A)\}$. That is, two sets A and B are considered similar to each other if A is similar to B and B is similar to A.

A naive method to identify a point set in a collection $\mathcal{D}$ of point sets which minimizes the HausDist from Q is to compute $\textsc{HausDist}(Q, S)$ for each point set S in $\mathcal{D}$

and identify a point set that yields the smallest HausDist. One may reduce the number of HausDist computations by computing a lower bound value (an optimistic estimate) of the actual distance HausDist(Q, S) for each S to rule out sets that are obviously dissimilar to Q. Ideally, we want this lower bound to provide an estimate that is close to the actual distance HausDist(Q, S), while keeping the computation cost low with respect to that of HausDist(Q, S). Hence, the challenge of formulating a Hausdorff lower bound lies in the trade-off between the quality of the estimate and its computation cost.

Traditionally, a lower bound on HausDist(A, B) can be computed as the MaxMin distance [93] from the minimum bounding rectangle (MBR) of A to the MBR of B. Our research is based on the observation that this *basic lower bound* (BscLB) may be inaccurate when the MBR that covers A significantly overlaps the MBR that covers B. In such a case, the inaccuracy of BscLB may result in a large number of HausDist computations which can be highly undesirable when the number of point sets is large. Based on this observation, we propose a novel method which decomposes the two MBRs into sub-MBRs and computes a lower bound using sub-MBRs of the two sets. We call this new method the *enhanced lower bound* (EnhLB). Our experimental results show that EnhLB provides an estimate that is significantly closer to the actual HausDist than BscLB, which results in a greater pruning capability. Although EnhLB incurs a greater computation cost than BscLB, a significant overall performance improvement is obtained.

Figure 5.1 provides a comparison between BscLB and EnhLB (computed using the methods described in Section 5.3). In Figure 5.1a, the value of BscLB from A to B is calculated as the MaxMin distance from the MBR that encloses A to the MBR that encloses B. An edge pair that yields the BscLB value of 3 is highlighted in grey. In Figure 5.1b, the value of EnhLB from A to B is calculated using the sub-MBRs of those in Figure 5.1a. An edge pair that yields the EnhLB value of 7.62 is highlighted in grey. In this example, EnhLB is 2.54 times closer to the actual HausDist than BscLB.

It can be seen that the accuracy of EnhLB(A, B) depends on how the MBRs of A and B are decomposed into sub-MBRs as well as the number n of sub-MBRs. Hence, we formulate an algorithm to find appropriate MBRs for each point set S. Specifically, we propose an algorithm which utilizes an R-Tree index R to hierarchically organize the data points in S. The algorithm traverses R starting from the root and decomposes larger MBRs into sub-MBRs (MBRs of their children) until a desired number n of sub-MBRs is reached. We also present an empirical study to choose an appropriate value of n in Section 5.5.

In addition to HausDist, which is a measure of maximum discrepancy, we extend the concept of EnhLB to support a measure of average discrepancy called the *modified Hausdorff distance* (MHD). We then formulate an incremental search algorithm which can be applied to both HausDist and MHD. We also use this search algorithm to demonstrate the effectiveness of EnhLB in comparison to BscLB.

The contributions of our work are summarized as follows.

- An improved method (EnhLB) of calculating a HausDist lower bound that provides a greater pruning capability than the basic method (BscLB).
- An incremental search algorithm that utilizes BscLB and EnhLB and can be applied to both HausDist and MHD.
- Performance evaluations of our search algorithm in terms of the (i) total response time, (ii) I/O cost, and (iii) processing cost.

The rest of this chapter is organized as follows. Section 5.2 contains a definition of our research problem. Our lower bound computation method and corresponding search algorithm are given in Section 5.3. Section 5.4 shows how our method can be extended to support the *modified Hausdorff distance*, an outlier-resistant variant of HausDist. In Section 5.5, we report our experimental results, while Section 5.6 provides conclusions and directions of future research.

5.2 Problem Definition

We model the problem of similarity search over a collection of point sets as a HausDist minimization problem. This is similar to the formulation of the k-nearest neighbor problem for finding similar points, except that the query object and collection comprise point sets rather than individual points, and our similarity measure is the Hausdorff distance rather than a point-to-point distance measure. Formally, we define our similarity search function as follows.

Definition 1 (Similar Point Set Query)**.** The query accepts a point set Q, a collection $\mathcal{D}$ of point sets and the number k of resultant point sets. As output, the function returns a list $\mathcal{A}$ of point sets such that

(i) each element of $\mathcal{A}$ is a member of $\mathcal{D}$;

(ii) $|\mathcal{A}|$ is equal to $\text{MIN}\{k, |\mathcal{D}|\}$;

(iii) for each S in $\mathcal{A}$ and each T in $\mathcal{D} \setminus \mathcal{A}$,

$$\text{HausDist}(Q, S) \leq \text{HausDist}(Q, T);$$

(iv) for each S_i and S_j in $\mathcal{A}$ where i is less than j,

$$\text{HausDist}(Q, S_i) \leq \text{HausDist}(Q, S_j).$$

This query can be processed by separating the resultant list $\mathcal{A}$ of point sets from the rest $(\mathcal{D} \setminus \mathcal{A})$. To avoid computing HausDist for every point set in $\mathcal{D}$, we can compute an optimistic estimate for each point set S in $\mathcal{D}$. Specifically, an optimistic estimator of $\text{HausDist}(Q, S)$ is a function which returns a distance guaranteed to be less than or equal to $\text{HausDist}(Q, S)$. We use this optimistic estimate to provide the search order and to rule

out entries that clearly cannot be in the result $\mathcal{A}$. Ideally, we want this estimator to produce a value as close to HausDist(Q, S) as possible. At the same time, we also want to keep the computation cost low with respect to that of HausDist(Q, S).

To avoid computing an optimistic estimate for every point set, we can index the point sets in $\mathcal{D}$ as rectangular objects (using their respective MBRs) in a hierarchical structure like the R-Tree. In this case, an optimistic estimate of the Hausdorff distance from Q to an R-Tree node N is a value guaranteed to be smaller than the Hausdorff distance from Q to any point set in N.

The objectives of our investigation are given as follows: (i) to improve the accuracy of the existing Hausdorff estimator without introducing an excessive computation cost; and (ii) to formulate a search algorithm which utilizes this estimator. This estimator improvement and the search algorithm are described in the next section.

5.3 Point Set Similarity Search

In this section, we introduce a method which improves the accuracy in computing an optimistic estimate (a lower bound) of HausDist(A, B) using the MBRs of A and B. We observe that real-world geographic point sets have a tendency to cluster around key locations like big cities or industrialized coastal areas. Our solution is formulated based on a hypothesis that the accuracy in estimating HausDist(A, B) can be improved by using the sub-MBRs, MBRs of such clusters.

We use the R-Tree index to store all point sets in the collection $\mathcal{D}$ where each point set S is represented as a rectangular object using its MBR. We use two types of R-Trees. The first type, *primary R-Tree*, is used to store a collection of point sets where each point set is represented by its MBR. The second type, *secondary R-Tree*, is used to store points in each point set. Note that the root node of a secondary R-Tree is equivalent to its representative MBR in the primary R-Tree.

5.3.1 Incremental Search Algorithm

Our search algorithm (Algorithm 2) makes use of the two optimistic estimators BscLB and EnhLB to help search for k similar point sets with respect to a query point set Q. Specifically, we use BscLB for preliminary search ordering and EnhLB to refine the search order. Our rationale behind this practice is that BscLB, which is cheaper to compute than EnhLB, can provide a reasonable estimate of HausDist(Q, S) when Q and S are far from each other. Hence, BscLB can be used as a preliminary pruning criterion to rule out point sets with large BscLB values.

We now consider the algorithm description (Algorithm 2). The algorithm finds k point sets in $\mathcal{D}$ which minimize the HausDists from a query point set Q. An environment variable n specifies the resolution in which EnhLB is computed and is shared throughout the algorithm descriptions in this section.

The initialization steps are given by Lines 1 to 8. We create two levels of R-Trees (as described in Section 5.2) to index all point sets. The primary R-Tree is used to index all point sets and data points in each point set are in turn indexed in a secondary R-Tree. The remaining steps of initialization include (i) creating an R-Tree *QueryRT* for the query point set; (ii) initializing a priority queue *PQ*; and (iii) creating an empty list $\mathcal{A}$ to store resultant point sets.

The control loop is given by Lines 9 to 25. At the beginning of each iteration (Line 10), we retrieve the head entry (N, d, *LB-Stage*) from *PQ*, where N is the node which has the smallest estimated HausDist d. The value of *LB-Stage* identifies the nature in which the current value of d has been calculated: "0" denotes BscLB, "1" denotes EnhLB, and "2" denotes an actual HausDist. The rest of the control loop is organized into the two following cases:

- Node N contains only one secondary R-Tree *SecRT*. In this case, we check the value of

LB-Stage. If *LB-Stage* is "0", then d is currently a BscLB value. Hence, we reset d to a EnhLB value and insert the entry back into *PQ* with an *LB-Stage* of "1". If *LB-Stage* is "1", then d is currently a EnhLB value. Hence, we compute the actual HausDist and assign it to d. Then, we insert the entry back into *PQ* with an *LB-Stage* of "2". Otherwise *LB-Stage* is "2", which means that d is final and *SecRT* can be included as a query result in $\mathcal{A}$.

- Node N contains multiple children C. In this case, for each child node C, we compute an estimate d using BscLB and then we insert a priority queue entry (C, d, *LB-Stage*) into *PQ* where *LB-Stage* is set to "0".

The control loop terminates when $\mathcal{A}$ contains k objects or when *PQ* is exhausted.

5.3.2 Lower Bound Computation

In this subsection, we describe how BscLB and EnhLB used by Algorithm 2 are computed. Traditionally, a lower bound of HausDist(A, B) can be computed as the MaxMin distance [93] from the MBR which encloses A to the MBR which encloses B. This is because HausDist(A, B) can be considered as the MaxMin distance from A to B. We formally define this lower bound function as follows.

Definition 2 (Basic Hausdorff Lower bound)**.** Let M_A and M_B denote the MBRs of A and B, respectively.

$$\text{BscLB}(M_A, M_B) = \text{max}\{\text{MinDist}(f_a, M_B) : f_a \in \text{FacesOf}(M_A)\}.$$

That is, we exploit the minimum enclosing property of MBRs and assume that each MBR face touches at least one object. We then compute a lower bound value of each face using MinDist. The maximum of these lower bound values becomes the resultant estimate and is guaranteed to be smaller than or equal to the actual HausDist.

Algorithm 2: SIMSEARCH(Q, $\mathcal{D}$, k)

input : Query point set Q, Collection $\mathcal{D}$ of point sets, and Number k of results
output : k point sets with the smallest HAUSDISTS with respect to Q
environment : Number n of MBRs used to compute ENHLB

```
1  PrimRT ← Create an empty R-Tree;
2  for each Point Set S in 𝒟 do
3      SecRT ← Create an R-Tree of S;
4      Insert SecRT into PrimRT;
5  QueryRT ← Create an R-Tree of Q;
6  Priority Queue PQ ← Create an "ascending order" PQ;
7  Insert (RootOf(PrimRT), 0, 0) into PQ;
8  List 𝒜 ← Create an empty list;
9  while PQ is not empty do
10     PQ-Entry(N, d, LB-Stage) ← Dequeue(PQ);
11     if N contains one secondary R-Tree SecRT then
12         if LB-Stage is 0 then
13             d ← EnhLB(QueryRT, SecRT);
14             Insert (N, d, LB-Stage=1) into PQ;
15         else if LB-Stage is 1 then
16             d ← HausDist(QueryRT, SecRT);
17             Insert (N, d, LB-Stage=2) into PQ;
18         else
19             Insert the point set from SecRT into 𝒜;
20             if 𝒜 contains k point sets then
21                 return 𝒜;
22     else
23         for each Child C of N do
24             Distance d ← BscLB(RootOf(QueryRT), C);
25             Insert (C, d, 0) into PQ;
26 return 𝒜;
```

We now present our method for computing an optimistic estimate of HAUSDIST(A, B) using MBRs of subsets of A and B. First, we describe our algorithm Algorithm 3 to find n MBRs which cover the point set S indexed in an R-Tree R. Our algorithm utilizes the R-Tree index which organizes objects in a hierarchy of MBRs and accepts an R-Tree R of a point set and the number of MBRs to be selected from R. The objective here is to find a list of MBRs which cover the point set S. We formulate an algorithm which traverses the R-Tree R according to the areas of MBRs in the hierarchy. Specifically, we decompose the largest MBRs because smaller MBRs are likely to provide tighter estimates than large ones.

In the initialization steps, we create an empty list L to store the resultant MBRs (Line 1), then initialize a priority queue *PQ* to arrange MBR entries according to their areas in descending order (Line 2) and insert the root of R as the first entry (Line 3).

At the beginning (line 5) of each iteration of the control loop, the R-Tree node N with the largest area a is retrieved from the priority queue. If N contains points, then N is inserted into the resultant list L since there are no R-Tree nodes beneath N. Otherwise, child entries of N are inserted into *PQ*. The control loop terminates when *PQ* is exhausted or there are at least n entries in *PQ* and L. Finally, all entries in *PQ* are inserted into L, and L is returned as output.

Figure 5.2 demonstrates an example run of Algorithm 3. The figure contains a three-level R-Tree R, where the top level (Level 3) corresponds to the root node and the bottom level (Level 1) comprises nodes whose immediate children are data points. Assume that the n value is 4. At the initialization, the resultant list L is initialized to an empty list and the root node is inserted into the priority queue *PQ* which arranges MBRs in descending order according to their areas. In the first iteration of the control loop, the root node is retrieved from the head of *PQ*. Then, we expand the root node by inserting its immediate children M_1, M_2, and M_3 into *PQ*. Since the size of *PQ* is 3 and L is still empty, we need to further explore R to meet the minimum requirement of n MBRs. In the second iteration, M_3, which is the

Algorithm 3: GETCOVMBRS(R)

input : R-Tree R of data points
output : List L of MBRs
environment : Requested number n of MBRs in L

```
1  List L ← Create an empty list of nodes (MBRs);
2  Priority Queue PQ ← Create a "descending order" PQ;
3  Insert (RootOf(R), 0) into PQ;
4  while SizeOf(PQ) + SizeOf(L) < n and PQ is not empty do
5      PQ-Entry (Node N, Area a) ← Dequeue(PQ);
6      if N contains points then
7          Insert N into L;
8      else
9          for each Child C of N do
10             Area a ← AreaOf(C);
11             Insert (C, a) into PQ;
12 for each (Node N, Area a) in PQ do
13     Insert N into L;
14 return L;
```

largest MBR, is retrieved from *PQ*. Then, we expand M_3 by inserting the children M_{10}, M_{11} and M_{12} into *PQ*. At this point there are 5 MBRs in *PQ* which is greater than the n value of 4. Hence, all these MBRs are inserted into $\mathcal{A}$ and returned as query results.

Note that the actual length l of the resultant list L may not exactly match the requested number n of MBRs. Specifically, the length l of L depends on how the conditions in Line 4 are broken, i.e., whether n MBRs are obtained or *PQ* is exhausted first. If n MBRs are obtained first, l must be greater than or equal to n but smaller than $(n + b)$ where b is the branching factor of R. This is because, at each iteration we can add at most b MBRs into *PQ*. In the case where *PQ* is exhausted first, i.e., we do not have enough MBRs to satisfy the request, l is less than n.

We now present our algorithm (Algorithm 4) to compute an optimistic estimate of HAUSDIST(A, B) using MBRs of subsets of A and B. The algorithm accepts R-Trees R_A and R_B of two point sets and the number n of MBRs from each R-Tree that will be used to calculate a lower bound. Specifically, we use GETCOVMBRS (Algorithm 3) to select n MBRs

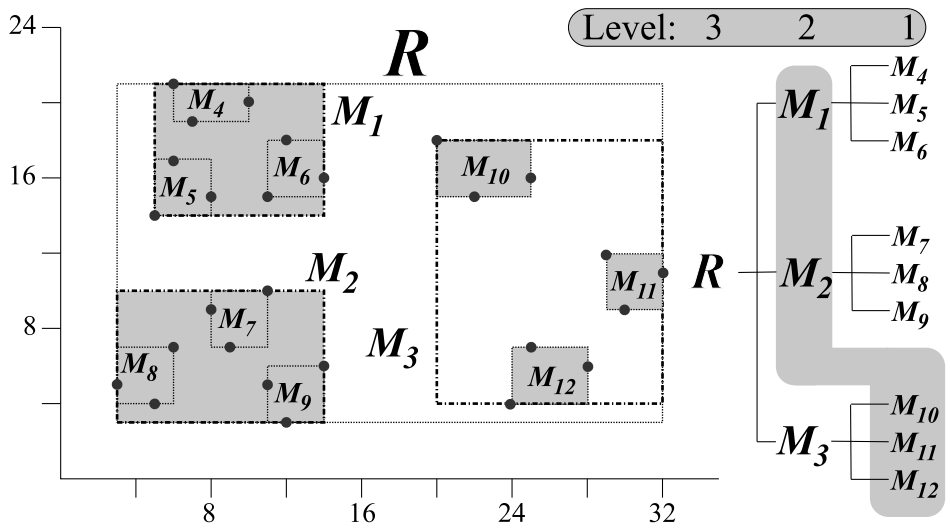


Figure 5.2: An example of GETCOVMBRS(R) with the requested number n of MBRs of 4, where selected MBRs are highlighted in gray.

from R_A and another n MBRs from R_B, and store them in lists L_A and L_B respectively (Lines 1 and 2). The resultant distance is computed as the MAXMIN distance from faces of MBRs in L_A to the MBRs in L_B (Lines 4 to 12). Specifically, for each face F_A of MBRs in L_A we compute the minimum distance from F_A to all MBRs B in L_B. The resultant distance is calculated as the maximum of these minimum distances computed in the while loop (Lines 8 to 10). Note that the while loop may not need to iterate through the entire L_B if it is found that the minimum distance of the current F_A to L_B cannot affect d_{max}. That is, the current d_{min} is less than or equal to the current d_{max}.

5.3.3 Discussion

As we have described, our similarity search algorithm (Algorithm 2) uses BSCLB for preliminary sorting and uses ENHLB to refine the estimate initially given by BSCLB. Consequently, we only compute ENHLB values for those point sets whose BSCLB values are insufficient to rule them out from the search. However, by introducing ENHLB as an intermediate step, each of the resultant point sets has to be considered three times, i.e., once for each of BSCLB, ENHLB and the actual HAUSDIST. This incurs an overhead in terms of prior-

Algorithm 4: EnhLB(R_A, R_B)

input : R-Trees R_A and R_B of two point sets, and the number n of MBRs used to compute the result
output : Optimistic Estimate of the HausDist from points in R_A to points in R_B
environment : Number n of MBRs used to compute EnhLB

```
1  MBR-List L_A ← GetCovMBRs(R_A);
2  MBR-List L_B ← GetCovMBRs(R_B);
3  Distance d_max ← 0;
4  for each MBR A in L_A do
5      for each Face F_A in A do
6          Distance d_min ← ∞;
7          B ← First MBR in L_B;
8          while d_min ≤ d_max and B is not null do
9              d_min ← min{d_min, MinDist(F_A, B)};
10             B ← Next MBR in L_B;
11         d_max ← max{d_max, d_min};
12 return d_max;
```

ity queue operations. To provide a better insight into performance evaluation, we compared this method to methods that use BscLB or EnhLB alone in our experiments (Section 5.5).

5.4 Extension: Handling Outliers

Since HausDist(A, B) is a measure of maximum discrepancy of A with respect to B, the measure can be sensitive to outliers. Specifically, if there is only one object $\boldsymbol{a}$ in A that is far away from B, then the distance from that object $\boldsymbol{a}$ to B will be used as the resultant distance. That is, the measure disregards the majority of points in A which are much closer to B. To mitigate this problem, a variant of the Hausdorff distance called the *modified Hausdorff distance (MHD)* [69] can be used to spread out the effect of outliers over the entire point set A. A formal definition of MHD can be given as

$$\text{MHD}(A, B) = \frac{\sum\{\text{min}\{\text{Dist}(\boldsymbol{a}, \boldsymbol{b}) : \boldsymbol{b} \in B\} : \boldsymbol{a} \in A\}}{|A|}.$$

In this section, we extend our concept of lower bound calculations to support the MHD measure.

Since MHD(A, B) is the average of the distances from points in A to their nearest point in B, the HausDist lower bound computed as the MaxMin distance from the MBRs of A and B is not guaranteed to be smaller than or equal to MHD(A, B). As a result, we have to use MinDist as our *MHD basic lower bound (*MHD-BscLB*)* in this case.

In a similar manner as the enhanced lower bound for the Hausdorff distance, an MHD enhanced lower bound can be computed from MBRs of subsets inside the point sets A and B. Algorithm 5 displays our MHD modification of Algorithm 4. Specifically, we can represent the point sets A in B as lists L_A and L_B of sub-MBRs, respectively (Lines 1 and 2). We can then compute a weighted sum of MinDist of MBRs in L_A to L_B based on the point count of the node corresponding to each MBRs in L_A (Lines 3 to 11). The resultant distance is the sum divided by the total number of points (Line 12).

Algorithm 5: MHD-EnhLB(R_A, R_B)

input : R-Trees R_A and R_B of two point sets
output : Optimistic Estimate of the HausDist from points in R_A to points in R_B
environment : Number n of MBRs used to compute EnhLB

```
1  MBR-List L_A ← GetCovMBRs(R_A);
2  MBR-List L_B ← GetCovMBRs(R_B);
3  Distance c_total ← 0;
4  Distance d_sum ← 0;
5  for each MBR A in L_A do
6      Distance d_min ← ∞;
7      Count c ← Number of points in A;
8      for each MBR B in L_B do
9          d_min ← min{d_min, MinDist(A, B)};
10     d_sum ← d_sum + d_min · c ;
11     c_total ← c_total + c;
12 return d_sum/c_total;
```

Figure 5.3 provides a comparison between MHD-BscLB, MHD-EnhLB and the actual HausDist from one point set to another. It can be seen that MHD-BscLB which is

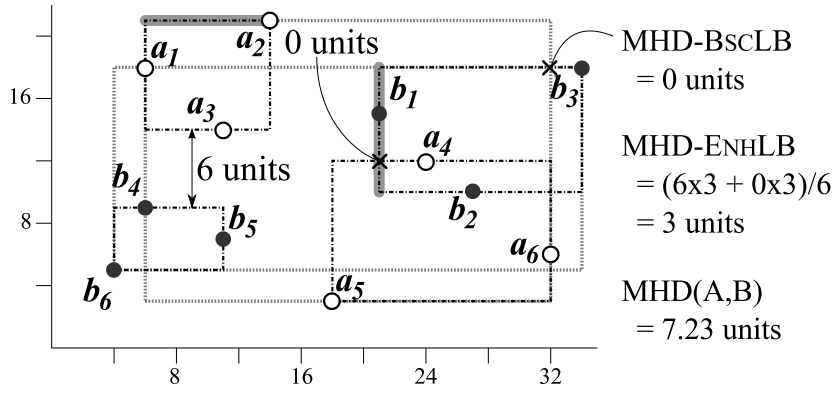
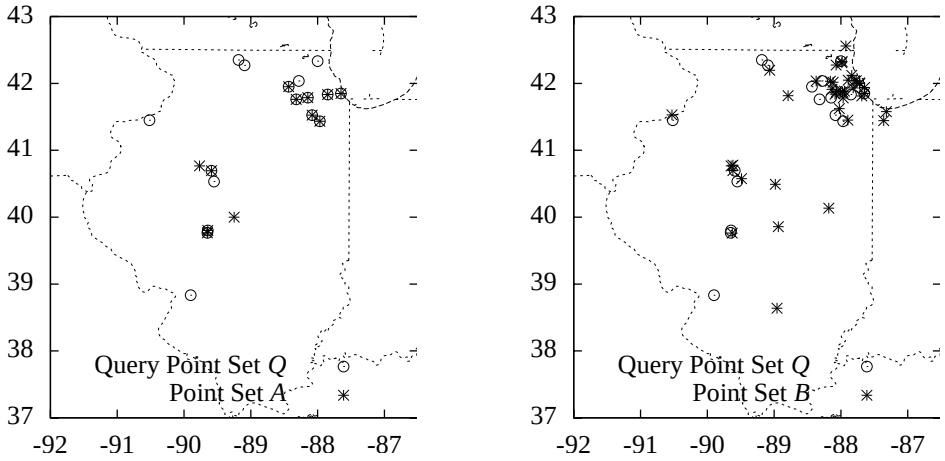


Figure 5.3: Comparison between MHD-BscLB, MHD-EnhLB and MHD from A to B ($\{\boldsymbol{a_1}, \ldots, \boldsymbol{a_6}\}$ to $\{\boldsymbol{b_1}, \ldots, \boldsymbol{b_6}\}$)

computed as MinDist yields an estimate of 0 units due to the overlap. On the other hand, we can decompose the root MBRs of A and B into sub MBRs where M_{A1} corresponds to $\{\boldsymbol{a_1},\boldsymbol{a_2},\boldsymbol{a_3}\}$, M_{A2} corresponds to $\{\boldsymbol{a_4},\boldsymbol{a_5},\boldsymbol{a_6}\}$, M_{B1} corresponds to $\{\boldsymbol{b_1},\boldsymbol{b_2},\boldsymbol{b_3}\}$, and M_{B2} corresponds to $\{\boldsymbol{b_4},\boldsymbol{b_5},\boldsymbol{b_6}\}$. MHD-EnhLB can be computed using Algorithm 5, where L_A and L_B are $\{M_{A1}, M_{A2}\}$ and $\{M_{B1}, M_{B2}\}$, respectively. In this case, MHD-EnhLB yields an estimate of 3 units which is much closer to the actual MHD of 7.23 units than MHD-BscLB.

To form a similarity search algorithm, we can modify Algorithm 2 by replacing BscLB with MHD-BscLB, EnhLB with MHD-EnhLB, and HausDist with MHD. In our experimental studies, we compare a method which uses both MHD-BscLB and MHD-EnhLB to ones which use MHD-BscLB or MHD-EnhLB alone. The difference between the similarity measures HausDist and MHD are shown as example query results in Figure 5.4. A query point set Q is shown, which comprises locations in Illinois, along with two of the top results (point sets A and B) when searching with the symmetric measures SymHausDist and SymMHD in the NA-Test dataset (described in Section 5.5). SymHausDist(Q, A) of 1.01 units is lower than SymHausDist(Q, B) of 1.32 units. For the SymMHD, the results are reversed. That is, SymMHD(Q, A) is 0.26 units which is greater than SymMHD(Q, B) of 0.22 units. We observe that B has a large collection of points near query points in the top right corner but has a few outliers near the bottom which are far away from the query points.

Using MHD means that the effects of these outliers are reduced by the averaging nature of the distance function. Hence, B is considered nearer to Q than A according to MHD.



(a) POINT SET A (b) POINT SET B

Figure 5.4: Top SIMSEARCH results for (a) HAUSDIST and (b) MHD queries.

5.5 Evaluation

In this section, we evaluate the effectiveness of our method. As shown in Table 5.1, we compare methods which use only BSCLB or ENHLB to Algorithm 2 which uses a hybrid of BSCLB and ENHLB. To emphasize this contrast, this method is referred to as HYB in this section.

The rest of this section is organized as follows. We first describe the dataset used for testing. Next, we study the impact of adjusting the number n of sub-MBRs selected for each point set on the performance of the three methods. Finally, we show the performance improvements achieved using HYB for our sample datasets.

Table 5.1: Similarity Search Methods

Search Method	Preliminary Sort	Refinement
BSC	BSCLB	-
ENH	ENHLB	-
HYB	BSCLB	ENHLB

5.5.1 Setup

Evaluating our Hausdorff search algorithm requires running the algorithm on a collection of point sets, which we created by geotagging a large number of tables extracted from spreadsheets in our table corpus. The geotagging method used for these tables was a preliminary version of our algorithm, not the full algorithm based on combined hierarchical place categories from Chapter 3.

After spreadsheet extraction and geotagging, we sampled a subset of the point sets for our evaluation in which there was a large degree of overlap between the point set MBRs and each set contained a large number of points. Specifically, we selected only point sets containing over 300 points in North America to form a test dataset that we call NA-TEST. Note that the HYB search method exhibits fast performance on the full data set as well—however, small point sets were excluded because they cause the Hausdorff distance computation to become less expensive, so differences between the methods are negligible. As shown in Table 5.2, the resulting collection contains 923 point sets, with an average point set size of 955 points.

Table 5.2: NA-TEST Point Sets

Number of Point Sets	923
Minimum Point Count	300
Maximum Point Count	5,796
Total Point Count	881,713

In our implementation, HAUSDIST(A, B) and MHD(A, B) are computed by iterating through the pairwise distances $\{\text{DIST}(\boldsymbol{a}, \boldsymbol{b}) : \boldsymbol{a} \in A, \boldsymbol{b} \in B\}$, which is sufficiently fast for these point sets, given that the average size is less than 1,000 points.

5.5.2 Accuracy of Estimators

Figure 5.5 shows the accuracy of BscLB and EnhLB as we vary the number k of returned point sets. For smaller k values, EnhLB produces estimates which are much closer to the actual HausDist (normalized as 1). As k increases (which means that HausDist(Q, S) increases), the difference between EnhLB and BscLB diminishes. As a result, when sorting point sets S with respect to Q, we can use BscLB to rule out point sets that are obviously far away from Q and use EnhLB for point sets that require further examinations before calculating the HausDist.

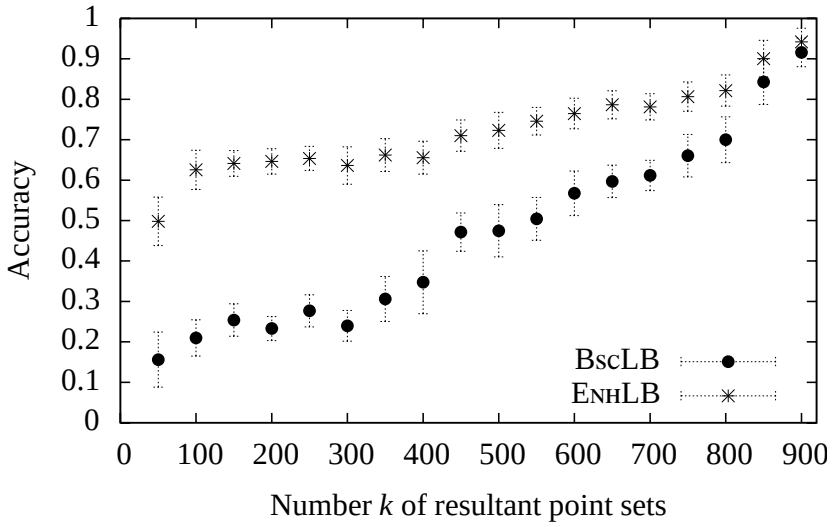


Figure 5.5: Accuracy of the two estimators BscLB and EnhLB given as the estimated distance divided by HausDist where (i) the measured value is the average μ of 100 runs, and (ii) each error bar represents one standard deviation in either direction from μ.

5.5.3 Selecting the number of MBRs

The motivating hypothesis behind the Enh and Hyb search methods is that using multiple sub-MBRs to calculate a lower bound of the Hausdorff distance is substantially more accurate than using a single MBR (i.e., the root MBR). To test this hypothesis, we now look at the performance of the SimSearch algorithm using varying numbers of sub-MBRs.

Figure 5.6 shows the performance of the SimSearch algorithm on the NA-Test dataset,

using different values of n. For this test, we randomly selected a sample of 100 point sets to serve as our collection of query point sets *QuerySets*. For each $Q \in$ *QuerySets*, we perform SimSearch with the number k of results set to 1 and the number of MBRs set to n. The average running time for each value of n is displayed. The tests were performed using the Enh and Hyb methods for $20 \leq n \leq 240$, and using the Bsc method. Since the Bsc method is equivalent to using either of the other methods with $n = 1$, the Bsc result is displayed as *num MBRs* $= 1$ in the figures. The running time for each value of n is broken into three components: (i) *LB Time*, (ii) *HausDist Time*, and (iii) *PQ Time*. These correspond to (i) time spent generating lower bound estimates, (ii) time spent computing exact Hausdorff distances, and (iii) time spent maintaining the priority queue of the collection of point sets, NA-Test.

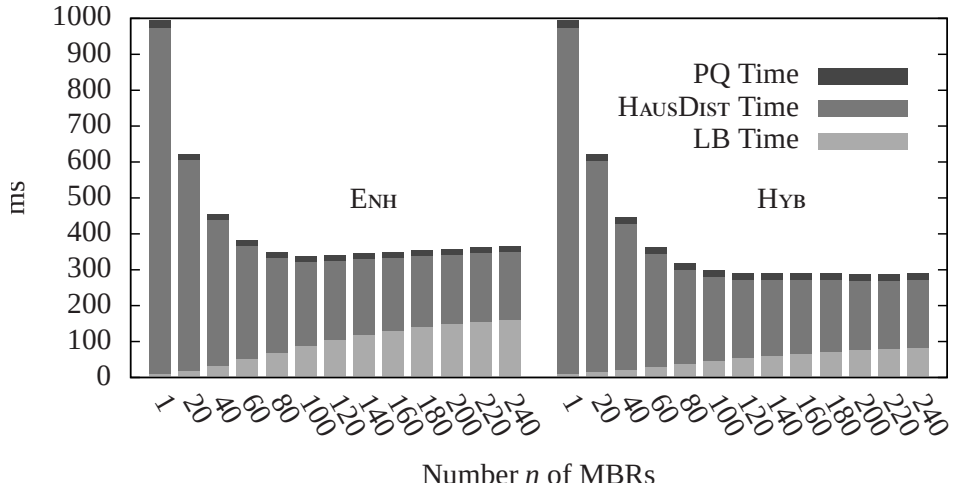


Figure 5.6: Average performance of Enh and Hyb search methods on the NA-Test dataset, for different numbers of sub-MBRs, and $k = 1$.

We can see that the worst performance, in terms of total running time, occurs when $n = 1$ (the Bsc case). The majority of this time is spent computing exact Hausdorff distances between the query set Q and other point sets in NA-Test. *PQ Time* is relatively small in this case, as it is for all other values of n, so we focus our discussion on the *LB Time* and HausDist *Time* factors. When $n = 1$, *LB Time* is also small, since only the BscLB value is being computed for candidate point sets. However, as expected, BscLB alone does not provide a particularly accurate ranking of point sets in NA-Test, so identifying the set with

the smallest Hausdorff distance to Q requires performing a large number of exact Hausdorff distance calculations.

The value of n has a positive correlation with *LB Time*. This is because calculating EnhLB requires visiting each sub-MBR in the query and candidate point sets. However, *HausDist Time* has a negative correlation with n due to the accuracy improvement. This dominates the effect of n on *LB Time*. As a result, we observe an overall decrease in the total running time as n increases.

The same experiment was performed using MHD instead of HausDist, as shown in Figure 5.7. The total computation time is generally one order-of-magnitude greater than for the HausDist search experiment on the same data set. The slower behavior is caused by the inherent differences in the distance measures, which we observed in Section 5.4. In particular, since the lower bound computations are based on MinDist instead of MaxMin, the bounds are generally much smaller than their HausDist counterparts. This means that many more candidate point sets must be considered before we identify point sets that have a MHD value that is less than the minimum remaining BscLB or EnhLB in the priority queue of SimSearch.

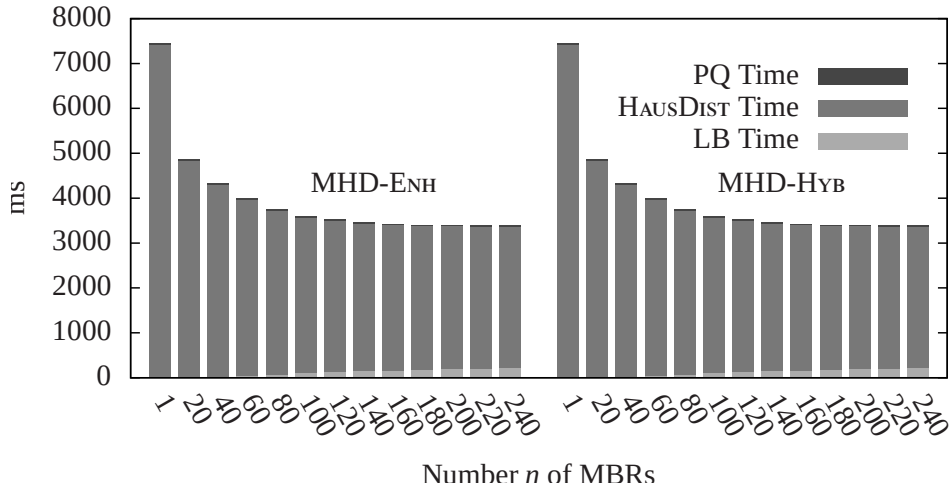


Figure 5.7: Average performance of MHD-Enh and MHD-Hyb search methods on the NA-Test dataset, for different numbers n of sub-MBRs and $k = 1$.

Despite the overall increase in running time for performing SimSearch with MHD, there is still a large increase in performance as n increases from small values. The performance of the Enh and Hyb search methods is nearly identical, which shows that using BscLB provides very little benefit in the SimSearch procedure.

In summary, the most significant outcomes of this experiment are (i) the performance of SimSearch benefits greatly from using EnhLB, and (ii) choosing an appropriate value of n does not require extreme precision. The first outcome is clear from the reduced running time for any n value greater than 1, whereas the second follows from the nearly flat behavior of the Hyb graphs for sufficiently large values of n. In this case, the flat behavior starts at the n value of 140. Hence, we choose 140 as the default value of n hereafter.

5.5.4 Performance Studies

Next, we focus on the performance improvements achieved by the Hyb method under various queries and query parameters. The performance improvements can be evaluated using multiple measures, as displayed in Figures 5.8-5.10, which show how the performance of SimSearch changes for different values of the k parameter, while fixing n to the default value of 140 MBRs. The experiment involved running SimSearch once against the NA-Test dataset for each combination of the following parameters:

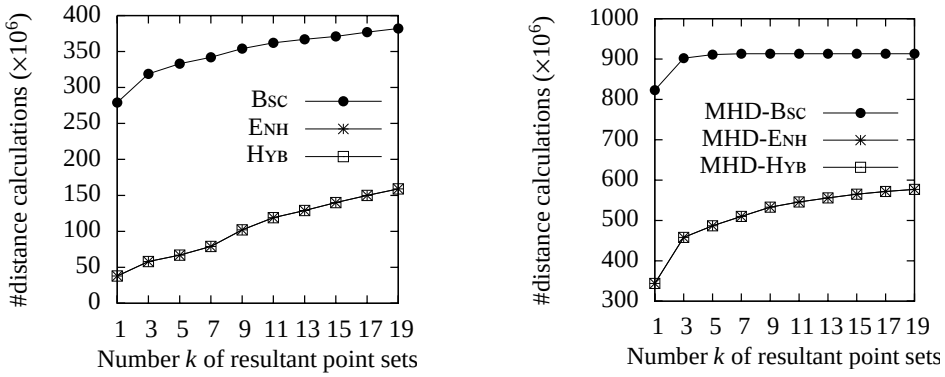
- every $Q \in$ *QuerySets* (cardinality: 100)
- every search method (Bsc, Enh, and Hyb)
- every odd value of k from 1 to 19
- both HausDist and MHD

For each combination, we recorded the following measures:

- number of point set to point set Hausdorff distance computations performed
- number of point-to-point and MBR-to-MBR distance calculations performed

- total search time

Each measure was averaged over all Q in *QuerySets*. Figure 5.8 plots the increase in the total number of Hausdorff distance computations performed using each lower bound method, for increasing values of k. The results show that the Bsc method consistently requires the greatest number of Hausdorff distance computations, while Enh and Hyb both require significantly fewer calculations. In fact, Enh and Hyb require exactly the same number of Hausdorff distance computations in each case. This result is due to the fact that both methods use the EnhLB value to order the search priority queue before computing the Hausdorff distance between point sets. The results also show that for every lower bound method, the number of distance calculations increases as k increases, since the results of searching with a larger k value will be a superset of the results with a smaller k value. An interesting observation from this experiment is that the relative performance improvement from Bsc to both Enh and Hyb decreases as k increases, which means that the largest reduction in Hausdorff distance computations occurs when k is equal to 1.

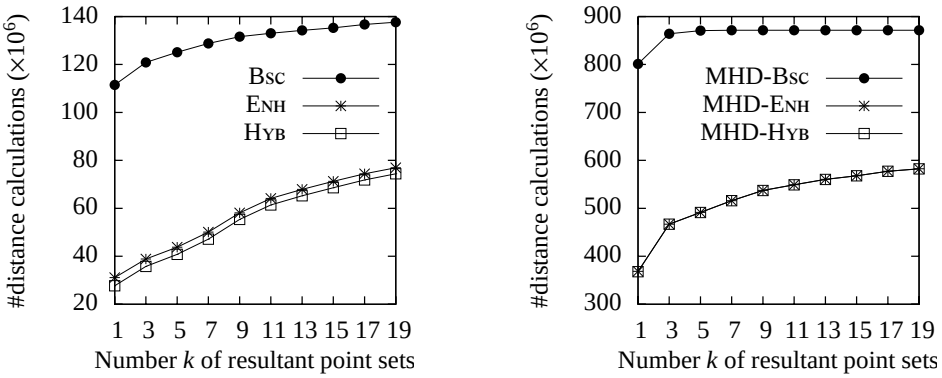


(a) HausDist (b) MHD

Figure 5.8: Average number of full Hausdorff distance computations performed during SimSearch queries using Bsc, Enh and Hyb search methods on the NA-Test dataset, for different values of k.

Figure 5.9 shows the total number of distance calculations that occur during search for increasing values of k. The number of distance calculations includes both point-to-point distance calculations performed during Hausdorff distance computations, and MBR-to-MBR

distance calculations performed during lower bound computations. In the chart, we see a similar result to Figure 5.8, except that ENH and HYB are slightly separated, representing the fact that measuring the total distance calculations also takes the lower bound computations into account. Hence, this is a more complete measure of the total computation costs than Hausdorff distance computations alone.



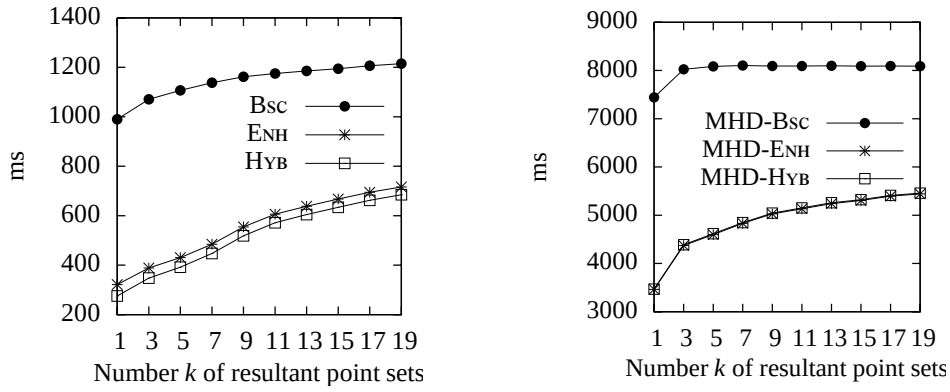
(a) HAUSDIST (b) MHD

Figure 5.9: Average number of distance calculations performed during SIMSEARCH queries using BSC, ENH and HYB search methods on the NA-TEST dataset, for different values of k.

Figure 5.10 plots the average time required to return the k point sets from *QuerySets* with the lowest Hausdorff distance from each Q. The elapsed time has a strong correlation to the number of distance calculations performed, as shown in Figure 5.9. However, here we see the average time required for each search, which ranges between 275 ms and 684 ms for the HYB method using the Hausdorff distance, and between 3468 ms and 5459 ms for the HYB method using the Modified HAUSDIST (MHD). These times were recorded on an Intel i7-2720QM @ 2.20 GHz with 8GB RAM.

5.5.5 Performance Distribution

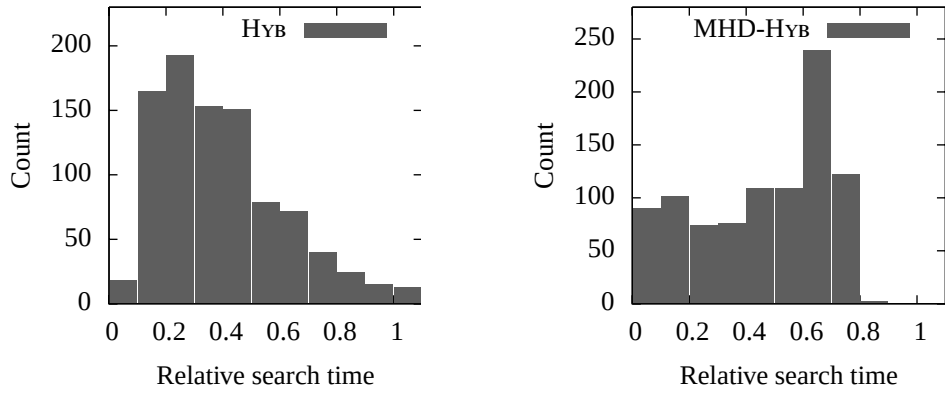
To gain a better insight into the performance of HYB relative to BSC, we show distributions of performance improvements using histograms in addition to the average performance presented previously in Sections 5.5.3 and 5.5.4. The histogram in Figure 5.11a presents the relative performance of HYB with respect to BSC for 923 query point sets. The x-axis of the



(a) HausDist (b) MHD

Figure 5.10: Average performance of SimSearch queries using Bsc, Enh and Hyb search methods on the NA-Test dataset, for different values of k.

histogram represents the relative search time, i.e., the total search time of Hyb divided by that of Bsc for each query point set. For example, a relative performance value of 0.2 means that Bsc takes 5 times as long as Hyb to process the same query. The x values are organized into 11 bins, where the leftmost bin represents a relative performance range of $[0.0, 0.1)$ and the rightmost bin represents a relative performance range of $[1.0, 1.1)$. The y-axis represents the count for each bin. We set k and n to the default values of 1 point set and 140 MBRs, respectively. The same setup also applies to Figure 5.11b which presents the relative performance of MHD-Hyb with respect to MHD-Bsc.



(a) HausDist (b) MHD

Figure 5.11: Histogram of performance improvements for different query point sets. Performance improvement is measured as elapsed search time using the Hyb method, as a fraction of elapsed search time using Bsc, so smaller values represent larger speedups. For all tests, $n = 140$ and $k = 1$.

The distribution of relative search times for Hausdorff distance searches is shown in Figure 5.11a. The vast majority of HYB searches take only 0.1 to 0.5 of the original BSC time, corresponding to a 50% to 90% reduction in search time, which is a significant improvement. 73.7% of queries result in HYB search times that are at least 50% less than the corresponding BSC search time. A small fraction of query point sets (1.4%) suffer an increased search time, which occurs when using the ENHLB does not result in a better ordering of point sets in the priority queue.

Using the HYB method for modified Hausdorff distance searching (MHD-HYB) also achieves significant speedup factors, although the distribution is different, as shown in Figure 5.11b. In particular, all queries are sped up using HYB, and a larger fraction achieve speedup factors higher than 90% (i.e., a search time that is less than one tenth of the original BSC search time). However, only 48.8% of queries result in HYB search times that are at least 50% less than the corresponding BSC search time.

5.6 Summary

This chapter presented a new approach for similarity search over a large collection of point sets, where similarity is measured using the Hausdorff distance. Our method constructs an ordering of the collection of point sets using a new lower bound estimation technique called ENHLB, which allows us to rule out dissimilar point sets without computing the full Hausdorff distance between them and our query set. We also applied this technique to searches using the modified Hausdorff distance, an outlier-resistant variant. On a dataset of geotagged spreadsheets from the Web, similarity search times improved substantially using our method, for both distance measures.

Chapter 6

Data Table Visualizations

As we have observed, the set of Web-accessible geographic tables can be treated as a heterogeneous collection of geographic feature sets. Tables vary in terms of the scale of the features they describe, the proximity of their locations, the presence (or absence) of implied travel between locations, and in other ways, and this heterogeneity requires a similarly broad variety of visualizations for analyzing and interacting with them. In this chapter, we look at three specific visualization and interactivity challenges that stem from the extraction of information from geographic data tables and explore three corresponding techniques to address them. Section 6.1 presents an itinerary layout algorithm that is useful for exposing itineraries such as those identified in Chapter 4 to user inspection. In Section 6.2, we look at an interface for performing Hausdorff and modified Hausdorff distance queries on extracted point sets, which allows the identification of tables with specified geographic distributions. Section 6.3 describes a strategy for supporting visual comparisons between locations in a table, rather than spatial comparisons, while Section 6.4 concludes the chapter.

6.1 Automated Tabular Itinerary Layout

In its basic form, the itinerary visualization problem is to produce a visual encoding of a graph $G = (V, E)$ where V represents the waypoints and E represents edges between consecutive waypoints. Each waypoint $w \in V$ is represented as a tuple (x, y, i, name) where

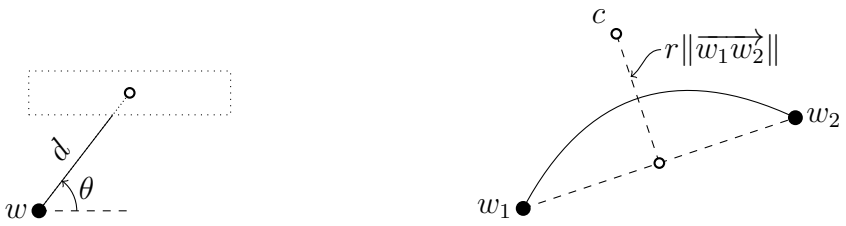


Figure 6.1: Geometric effects of layout parameters (a) d, θ, and (b) r. Label placement is determined by parameters d and θ, while link curvature is determined by parameter r.

x and y are the Cartesian coordinates describing the location of the waypoint (usually in projected or screen coordinates), i is the index of the waypoint in the current itinerary, and $name$ is a string value containing the place name (or other description) for the waypoint that should be included in the visualization. Note that we can treat all point locations as Cartesian coordinates. In case of coordinates encoded as latitude/longitude pairs, we convert to Cartesian coordinates using a suitable geographic projection (e.g., the Mercator).

An interesting aspect of itinerary visualization is that, since edges connecting consecutive waypoints are not meant to track a precise route, the intentional imprecision can be made clear to viewers. One way to achieve this is with simple curves that mimic artistic renderings of itineraries. So, rather than straight edges used in many types of graph visualization, we allow curved edges, where the curvature of the edges is determined by layout parameters.

As shown in Figure 6.1, the degrees of freedom in our itinerary layout algorithm involve the positioning of the text label associated with each waypoint and the curvature of paths representing journeys between consecutive pairs of waypoints. Thus, we have two parameters (θ and d) for each label and one parameter (r) for each edge, resulting in a total of $3n - 1$ parameters for an itinerary consisting of n waypoints.

1. *Label position*. The positions of text labels are determined based on a direction (θ) and distance (d) from the waypoint location. The label is positioned such that the centroid of the label is along the line extending in direction θ from the waypoint, but so that the

nearest point on the label's bounding box is at distance d from the waypoint.

2. *Curvature*. Each segment between consecutive waypoints is drawn as a quadratic Bézier curve. The control point of the curve is placed along the perpendicular bisector of the straight line segment connecting the waypoints. For consecutive waypoints w_1 and w_2 and curvature parameter r, control point c is positioned on the perpendicular bisector of $\overrightarrow{w_1 w_2}$ at a distance of $r\|\overrightarrow{w_1 w_2}\|$ from the segment's midpoint. For positive (negative) values of r, the control point is positioned to the left (right) while traveling from w_1 to w_2.

Other potentially usable parameters are discussed in Section 6.4. However, segment curvature and label position address the most common adjustments that we noticed in a collection of itineraries found on the web that were created by human cartographers. Additionally, the size of the layout search space is exponentially related to the number of available parameters, so we prefer to keep that number small.

The quality of a layout is measured based on the presence or absence of several factors. We observe that the following factors detract from the suitability of an itinerary layout.

1. Labels outside the visible map area
2. Edges overlapping non-incident waypoints
3. Labels overlapping other labels
4. Edges overlapping other edges
5. Labels overlapping edges
6. Labels overlapping waypoints
7. Small angles between incident edges
8. Deviation of segment curvature ratios from target

9. Distance of labels from corresponding waypoints

Our goal is to identify a layout that minimizes the presence of these factors by varying the available parameters, so it can be treated as an optimization problem. The layout constraints lead to numerous local minima in the objective function, which may act as attractors for a greedy approach. We avoid these by developing a layout method based on simulated annealing.

6.1.1 Method

In the simulated annealing context, the value of the objective function for a specific candidate is called its *energy*. We compute the energy of a candidate layout as a weighted sum of negative factors. Formally, we measure energy $e = \sum_{i=1}^{9} \alpha_i f_i$, where each f_i corresponds to one of the undesirable factors listed above (e.g., the number of edge-edge intersections) and α_i represents the corresponding weight. Weights were chosen by evaluating user preferences on a small set of sample itineraries and recording an order of weights assigning the highest penalty to the least desirable layouts (factors are listed in order, starting with most undesirable). Lower energy values indicate more desirable layouts.

The most computationally expensive components of the energy computation procedure involve detecting intersections between Bézier curves that represent edges and other edges, as well as the intersection of these curves with waypoint nodes and text labels. One method is interval subdivision, which evaluates the curve equation at several points and uses the convex hull properties of Bézier curves to test the bounding box for intersections. Other methods for detecting intersections between Bézier curves include curve implicitization and clipping [94]. However, these methods introduce complexity into the layout algorithm for only modest speed improvements over interval subdivision when the Bézier curves are of low degree (such as the quadratic curves used in our method).

We use the standard simulated annealing formulation, whereby the progression of the

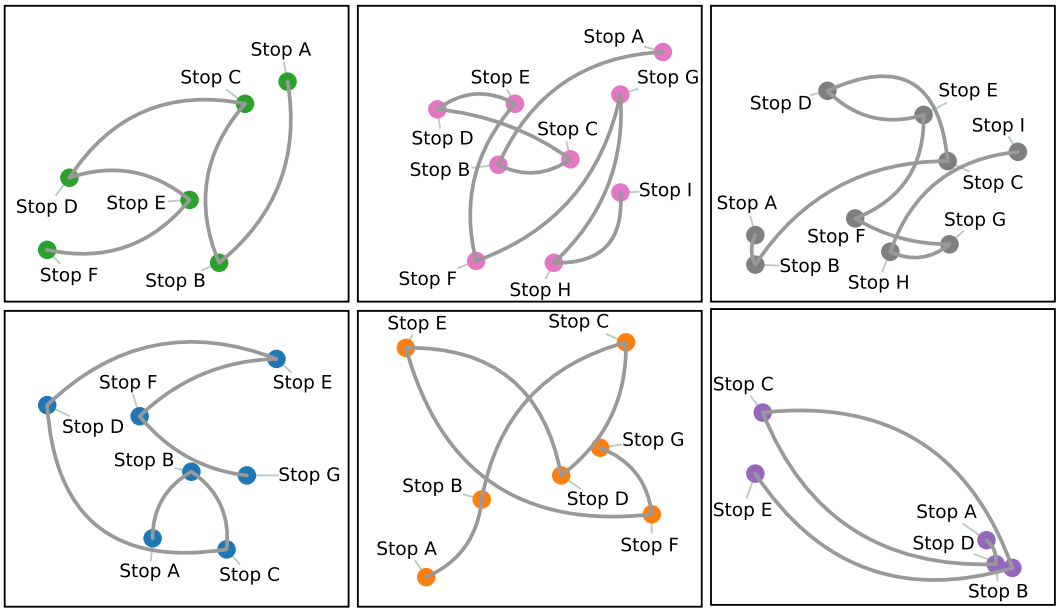


Figure 6.2: Six sample itinerary layouts. Each itinerary visits a collection of between 5 and 10 randomly-located waypoints. The layout parameters for each itinerary are computed independently.

algorithm is controlled by a *temperature* variable. A high initial temperature is iteratively reduced, simulating the cooling process that takes place in physical annealing scenarios. As the temperature decreases, parameter changes that result in higher energy layouts are less likely to be chosen as the next state of the system. The acceptance probability for a particular parameter change is based on the Metropolis criterion, a standard simulated annealing acceptance test [55]. The process terminates when the temperature falls below a specified threshold.

The simulated annealing algorithm is shown in Algorithm 6. It takes the input graph, represented as a set of waypoints P and a set of edges E, and augments the input graph with layout parameters (for curvature and label position). The algorithm begins by initializing the temperature variable t and energy variable e (line 1). For several iterations, the temperature decreases by ratio t_{decay} (line 8) until the temperature falls below the t_{accept} threshold (line 2). In each iteration, a candidate state is chosen (line 3) by picking parameter values for the layout using the CANDIDATE function, which applies a stochastic update to a single layout parameter.

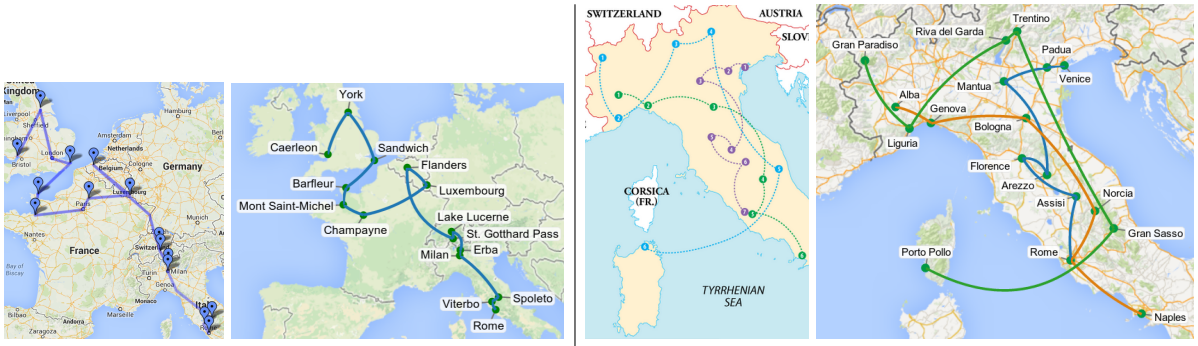
The candidate state's energy is computed by computeLayoutEnergy (line 4). Candidate layouts that reduce the energy are always accepted, while those that increase the energy are accepted according to the Metropolis criterion (line 5). When a candidate layout is accepted, its parameters and energy are copied to be used for subsequent comparisons (lines 6 and 7). Finally, the resulting layout is returned (lines 9).

Algorithm 6: FindLayout(P, E). Augment a collection of waypoints and edges with layout parameter values.

```
   input  : List of waypoints P, list of edges E
   output : Lists P, E, augmented with layout parameters
1  t ← t_0; e ← ∞;
2  while t > t_accept do
3  |  P_C, E_C ← candidate(P, E);
4  |  e' ← computeLayoutEnergy(P_C, E_C);
5  |  if e' < e OR exp((e − e')/t) < random() then
6  |  |  P, E ← P_C, E_C;
7  |  |  e ← e';
8  |  t ← t × t_decay;
9  return (P, E);
```

The number of iterations taken by simulated annealing here is $log_{t_{decay}}(t_{accept}/t_0)$. Since we expect that each layout parameter requires a consistent number of stochastic updates to arrive at an acceptable value, we update the t_{decay} value in order to maintain a consistent ratio of iterations to layout parameters. In particular, we set $t_{accept} = \phi^{1/n}$, where ϕ is the desired t_{decay} value for an itinerary with a single waypoint and n is the number of waypoints.

Figure 6.2 shows the results of our algorithm on six randomly-generated itineraries. The visualizations minimize the least desirable layout factors, with no occluded labels and minimal overlap of labels, edges, and waypoints. Minor layout issues are visible, such as the placement of labels for Stop B and Stop D in the bottom right diagram. Here, a better layout would involve switching the positions of the two labels. However, swapping them would have required several fortuitous parameter changes by the candidate function in order to climb out of a local minimum in the energy function, which did not occur.



(a) (b)

Figure 6.3: Two itineraries taken from image search results and their reproductions using our itinerary layout method. The reproductions are shown to the right of the originals. The map in (a) was created by a blogger to display her European itinerary. The order of stops and precise stop locations are difficult to discern in the original, but our automatically generated version addresses both of these issues by adding labels and using curves for edges. The map in (b) shows three suggested itineraries in northern Italy. Our method supports laying out multiple disconnected itineraries, and the simulated annealing algorithm settles on a layout that avoids label overlap even in a somewhat dense itinerary diagram such as this.

6.1.2 Demonstration System

A demonstration system illustrates the effectiveness of our itinerary layout algorithm. It consists of three primary components: (1) a geotagging module, (2) a mapping module, and (3) an itinerary visualization module. The geotagging module uses the method outlined in Chapter 3 to return geographic coordinates for place names that have geographic interpretations. The mapping module renders the map base layer, upon which the itinerary visualization is displayed. The system currently allows for Google Maps (with the standard Mercator projection) or one of several static map projections. A challenge for this module is picking appropriate bounds for the geographic window. To avoid a specialized solution for each projection, we use a generic algorithm that projects each waypoint into screen coordinates, computes a bounding box in projected coordinates, then scales and transforms the active region in the projection to fill the available space for the map. The resultant screen locations are used as inputs to the next module, which generates the itinerary layout. The visualization module takes the projection waypoint coordinates, along with the waypoint names and

edge topology, and generates layout parameters d, θ, and r for the corresponding waypoints and edges using the algorithm in Section 6.1.1. The system uses the actual screen sizes of waypoint label text to do accurate overlap tests. Once the simulated annealing algorithm's iterations are done, the resultant layout parameters are used to render the itinerary on the map.

The system accepts direct input of itineraries, by allowing users to enter names of waypoints, but any itinerary gathering technique could be substituted, such as supplying the extracted itineraries from Chapter 4 or supplying structured itineraries based on travel site content. Figure 6.3 shows two itineraries taken from the internet along with reproduced visualizations generated by our system.

6.2 Point Set Similarity Search

In this section, we present GeoXLS, an implementation of the search query algorithms described in Chapter 5 that enables users to submit a set of locations as a query object Q and to find data tables containing locations similar to those in Q. The results are ranked according to their similarity to Q, using one of several user-selected similarity measures related to the Hausdorff distance. The data tables that were indexed for this system include spreadsheets that were crawled from the Web and describe a wide variety of geographic features, including lists of universities, airports, and national parks. We use this collection to demonstrate how GeoXLS can be used to "complete the set" by identifying sets containing similar locations.

To demonstrate the versatility of GeoXLS, we also show that it can be applied to two other types of geotagged web documents in addition to spreadsheets and HTML tables. First, we apply GeoXLS to a collection of disease outbreak data. Users can specify a set Q of locations and find disease outbreaks that contain locations similar to those in Q. Second, we use a collection of geotagged news articles where each article may contain multiple locations. Using this dataset, users may issue a query like "Find events that are related to these n geographic locations." For example, to understand interactions between the United States and

China related to military activities in Libya, a user can place points in Washington, Beijing, and Tripoli as a search query, which returns multiple relevant articles.

6.2.1 System Components

As shown in Figure 6.4, GeoXLS consists of the following components: (i) document geotagger, (ii) RDBMS, (iii) Hausdorff spatial index, and (iv) web application.

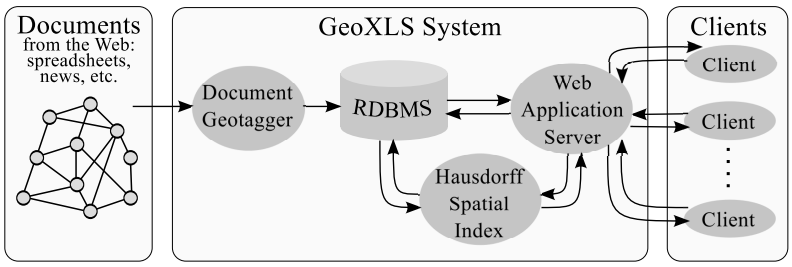


Figure 6.4: GeoXLS system architecture.

Before users can search for web documents, those documents must be processed and stored. This is accomplished by the *document geotagger* module, which performs different actions depending on the data source (e.g., spreadsheets, news articles, disease outbreaks, etc.). Table geotagging has been discussed extensively in Chapter 3. We have developed similar submodules for geotagging news articles and disease outbreak documents based on the geotagging components of the NewsStand [60, 92, 102] and related STEWARD [65] systems.

The results of the document geotagging process are stored in a relational database management system (*RDBMS*) which maintains information about each geotagged document and its associated points. For example, in the case of data tables this includes the original source URI and title of each document, as well as the row number, place name, and lat/long for each associated point. This enables efficient ad hoc access to each web document's attributes without requiring that the original documents be read during every GeoXLS query.

In addition to the RDBMS, we developed a Python and C++ module to perform Haus-

dorff searches, which effectively serves as a Hausdorff spatial index over our database of point sets. That is, the Hausdorff spatial index component is used to answer questions of the following general form for a specified query point set q (note that we express the query using SQL syntax, but the Hausdorff index is not actually integrated into the RDBMS):

```
1:       SELECT pt_set
2:         FROM collection
3:        WHERE NumPoints(pt_set) < maxPoints
4:     ORDER BY HausDist(q, pt_set)
5:        LIMIT k;
```

The search system accepts the following parameters:

- q. Query points can either be specified as a set of lat/long values or by using another point set that already exists in the index.
- $k \geq 1$, specifies the number of search results to return.
- *maxPoints* ≥ 1, the maximum point set size to consider for the search. Some documents have tens of thousands of points with wide geographic coverage. Without this parameter, these documents can cause search results to be full of large, mostly irrelevant point sets.

The distance function (Line 4) and the order of its arguments are determined by two additional parameters: *Hausdorff Type* and *Direction*. The following table shows how different combinations result in different settings.

Direction	Hausdorff Type	
	HausDist	MHD
FromQuery	`HausDist(q, pt_set)`	`MHD(q, pt_set)`
ToQuery	`HausDist(pt_set, q)`	`MHD(pt_set, q)`
Symmetric	`SymHausDist(q, pt_set)`	`SymMHD(q, pt_set)`

The Hausdorff spatial index is reconstructed periodically to incorporate additions to

each document collection, and is kept in memory to support efficient query response times.

The final component, which connects the RDBMS and Hausdorff spatial index to the end-user, is the GeoXLS web application. The application is composed of a back-end server written in Python, and a client-side web application written in JavaScript. The main function of the web application server is to generate the HTML pages that the client interacts with and handle asynchronous requests for data from the client. The current implementation of the server handles two types of asynchronous HTTP requests: *search query requests*, and *document information requests*. Search query requests are submitted with the array of parameters listed earlier in this section, and then passed to the Hausdorff index to execute the query. The server takes the resulting list of document identifiers and adds relevant information (such as the URI of the document or other metadata). The combined results are used to generate an HTML listing of search results. Document information requests are submitted when a user selects a document in the search result listing. To handle these requests, the server queries the RDBMS for a list of points associated with that document, and returns that list along with any point-specific attributes.

The client-side web application serves as the user interface for GeoXLS. It provides HTML input elements for user-specified search parameters, communicates with the web application server to perform search queries, and provides a Google Maps-based interface for entering query points and browsing the point sets of search result documents. A search query session typically consists of the following user actions:

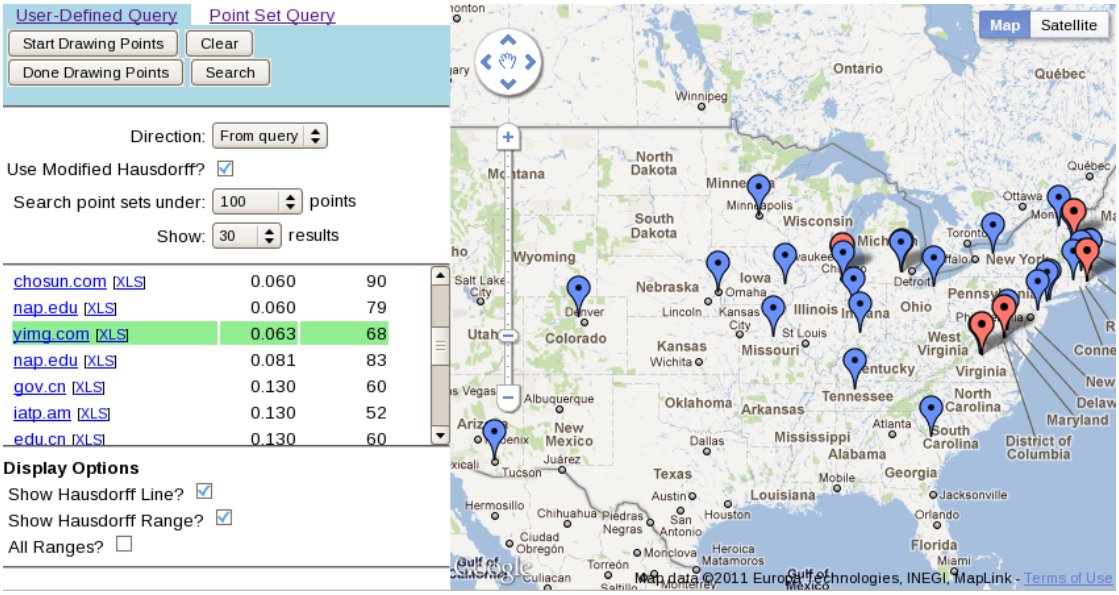
- *Specify query points.* This is done by (i) clicking "Start Drawing Points"; (ii) clicking appropriate locations in the map pane to add those to an array of query points; and then (iii) clicking "Done Drawing Points".
- *Select query parameters.* Users can specify the direction for the Hausdorff search, whether to use MHD or HausDist, the maximum point set size to consider in the search, and the number of results to display.

- *Execute search.* After users click the "Search" button, the search parameters are sent to the server. Results are displayed in tabular format.
- *Select display options.* When browsing search results, additional map overlays can assist users in interpreting the results. In particular, the Hausdorff distance can be displayed as: (i) a line between the points that determine the Hausdorff distance, or (ii) a circle around one or all of the markers in the appropriate point set.
- *Browse search results.* When users select documents from the search results, the application makes document information requests to the server, which returns the associated locations and their attributes. These are displayed on the map as blue markers, and additional overlays are rendered depending on the currently selected display options.

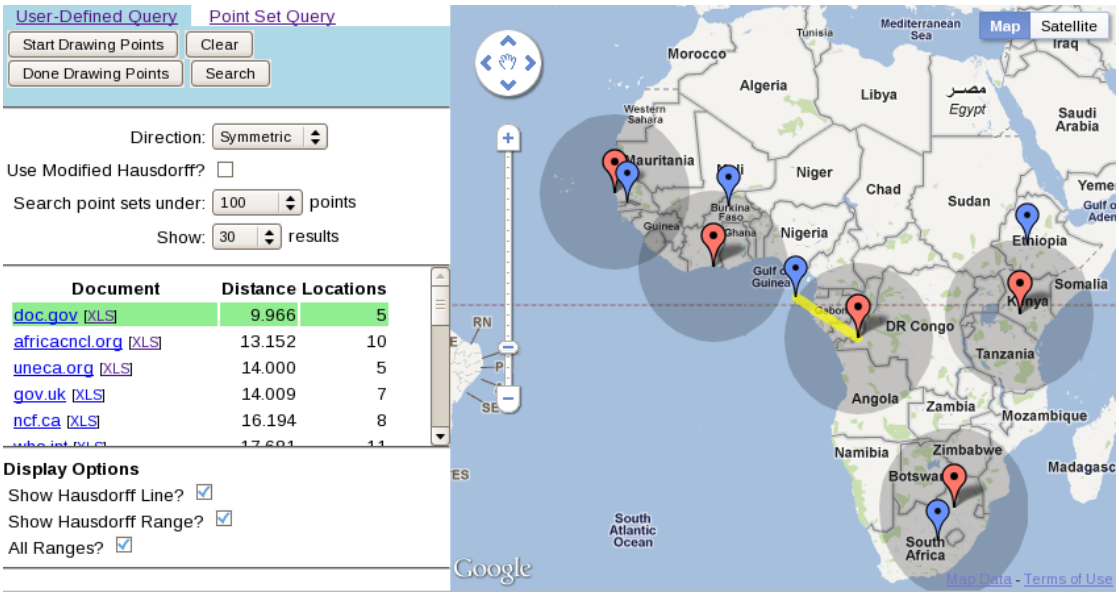
6.2.2 Query Examples

A screenshot of GeoXLS is shown in Figure 6.5a. The user has specified the locations of six universities in the U.S. as query points and executed a search using the modified Hausdorff distance. Based on the values of the selected search parameters, the search result listing contains 30 data tables with 100 or fewer points, ordered by each document's modified Hausdorff distance from the query point set. When executing a FROMQUERY search, GeoXLS returns point sets *containing but not restricted to* locations similar to the query points. Since documents mentioning these six universities are likely to include other universities, the search results contain documents with locations of various universities including those near the query set. Note that this containment relationship is reversed for TOQUERY and the containment relationship is bidirectional for SYMMETRIC.

Figure 6.5b shows another example of search results. The chosen query locations are large cities in Africa, possibly where the user's organization has international headquarters. The search results consist of many data tables that focus on Africa, potentially published by other groups that the user's organization can partner with. The line and circle overlays illus-



(a)



(b)

Figure 6.5: GeoXLS search results where the query points represent (a) the locations of six universities in the U.S. and (b) the locations of five cities in Africa. The query points are shown as red markers, the selected search result is highlighted in green, and the point set representing the selected search result is shown using blue markers. The Hausdorff distance is illustrated as a line and circles around the query points in (b).

trate the geometric properties of this measure. The yellow line connects the two points that determine the symmetric Hausdorff distance in this example, and the grey circles around the query points use the Hausdorff distance as their radii. Due to the definition of the Hausdorff distance, this means that each circle contains at least one point in the result set (blue marker).

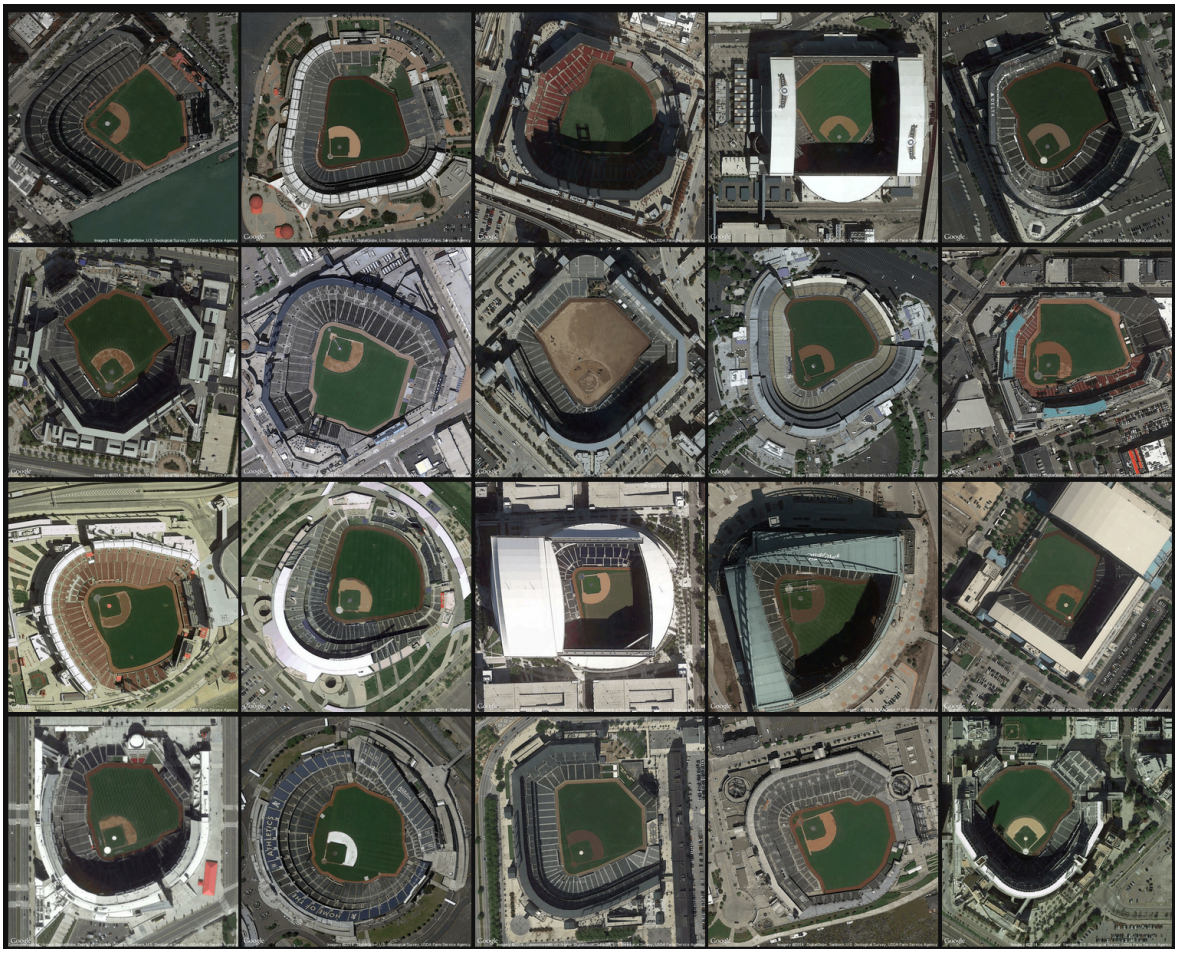
6.3 Parallel Detail Views

A third visualization technique that can be applied to geographic tables involves showing a collection of individual, detailed views of each location associated with a table row. We call this the *parallel detail view*. In this visualization, we are exploiting the expected coherence between objects that appear within a table, in order to show a detailed view of each individual location. Unlike full map visualizations that show the spatial relationships between geographic objects, the parallel detail view enables a visual comparison of the objects. Specifically, given a set of geographic coordinates, the parallel detail view supplies a detailed satellite view of each location. The current system displays views using the Google Static Maps API, however with the proliferation of imagery such as Bing Bird's Eye imagery or Google Street View, multiple options could be supplied.

The parallel detail view requires very precise geotagging and would not be appropriate for larger geographic features, such as cities or administrative regions. However, for landmarks or features with a distinctive satellite appearance, it can be useful to see a collection of such objects using detailed satellite imagery of each in tandem. We note that the choice of zoom level can affect the clarity of the visualization, as geographic features of different types may be too large or too small to inspect visually unless they are appropriately sized with the zoom level of the map. We find that few locational features are visually recognizable at scales smaller than 1:300,000 (roughly corresponding to Google Maps API zoom level 12 for moderate latitudes on a monitor with standard pixel density). The interface defaults to a zoom level of 14 and allows manual adjustment by the user. In the future, the selection of

	A	B	C	D	E	F	G
1	**Stadium**	**Seating capacity**	**Team**	**Playing surface**	**Opened**	**Distance to center field**	**Ballpark typology**
2	*AT&T Park*	41,503	San Francisco Giants	Grass	2000	399 feet (122 m)	Retro Classic
3	*Angel Stadium of Anaheim*	45,483	Los Angeles Angels of Anaheim	Grass	1966	396 feet (121 m)	Modern Retro Modern
4	*Busch Stadium*	43,975	St. Louis Cardinals	Grass	2006	400 feet (120 m)	Retro Classic
5	*Chase Field*	48,633	Arizona Diamondbacks	Grass	1998	407 feet (124 m)	Retro Modern
6	*Citi Field*	41,922	New York Mets	Grass	2009	408 feet (124 m)	Retro Classic
7	*Citizens Bank Park*	43,651	Philadelphia Phillies	Grass	2004	401 feet (122 m)	Retro Classic
8	*Comerica Park*	41,681	Detroit Tigers	Grass	2000	420 feet (130 m)	Retro Classic
9	*Coors Field*	50,398	Colorado Rockies	Grass	1995	415 feet (126 m)	Retro Classic
10	*Dodger Stadium*	56,000	Los Angeles Dodgers	Grass	1962	395 feet (120 m)	Modern
11	*Fenway Park*	37,499	Boston Red Sox	Grass	1912	420 feet (130 m)	Jewel Box
12	*Great American Ball Park*	42,319	Cincinnati Reds	Grass	2003	404 feet (123 m)	Retro Modern
13	*Kauffman Stadium*	37,903	Kansas City Royals	Grass	1973	410 feet (120 m)	Modern Retro Modern
14	*Marlins Park*	36,742	Miami Marlins	Grass	2012	422 feet (129 m)	Contemporary
15	*Miller Park*	41,900	Milwaukee Brewers	Grass	2001	400 feet (120 m)	Retro Modern
16	*Minute Maid Park*	40,963	Houston Astros	Grass	2000	435 feet (133 m)	Retro Modern
17	*Nationals Park*	41,418	Washington Nationals	Grass	2008	402 feet (123 m)	Retro Modern
18	*O.co Coliseum*	35,067	Oakland Athletics	Grass	1966	400 feet (120 m)	Multipurpose
19	*Oriole Park at Camden Yards*	45,971	Baltimore Orioles	Grass	1992	410 feet (120 m)	Retro Classic
20	*PNC Park*	38,362	Pittsburgh Pirates	Grass	2001	399 feet (122 m)	Retro Classic
21	*Petco Park*	42,302	San Diego Padres	Grass	2004	396 feet (121 m)	Retro Modern

(a)



(b)

Figure 6.6: (a) Spreadsheet containing data about Major League Baseball stadiums and (b) corresponding parallel detail view.

zoom level could potentially be automated using the feature types returned by the gazetteer. However, there is a wide variety of scales, even within the same feature classes. For example, baseball stadiums and public parks exist at different physical scales and would require different zoom levels of satellite imagery to perform useful visual inspections, but both are commonly listed as parks in the GeoNames gazetteer.

Figure 6.6 shows a spreadsheet containing factual information about Major League Baseball stadiums. While the tabular representation is useful for many purposes, questions such as "How many stadiums in Major League Baseball face to the East?" are easier to answer with the parallel detail view shown below. Other questions, such as "Which stadiums are located in urban centers?" could be answered by adjusting the zoom level of the satellite view.

As with the two visualizations looked at earlier in this chapter, parallel detail views have applications outside of table data, especially given the need for precise coordinates for each data point. Our implementation additionally supports Wikipedia categories as a way of specifying a location set, and associates each Wikipedia page in the category with the geographic coordinates on that page.

6.4 Summary

First, we presented an algorithm and system for generating automated itinerary visualizations. In the future, itinerary layout could be adapted to allow increased realism through options for paths along great circles (approximating flight paths) or along various land or sea transportation routes. The high-level framework that we introduced allows for the addition of such constraints. Additionally, the layout procedure itself could be adapted to include a force-directed component that performs with lower latency (although force-directed approaches are prone to settling at local minima). Further changes involving spatial distortion around waypoint locations, as in LineDrive [8], could also be used.

Next, we described GeoXLS, a novel system for searching collections of point sets. The search results are not based on proximity of a single query point or a single result point to others, but rather they are based on the overall similarity of the query and result point sets. GeoXLS supports multiple similarity measures related to the Hausdorff distance, and has been used to search several large collections of geotagged web documents.

Finally, we described a type of visualization that is well-suited to the task of visually inspecting and comparing the geographic locations found in a data table: the parallel detail view. While this view is only suitable for a subset of tables whose locations exhibit visually interesting features at a constant zoom level, the view provides a valuable way of inspecting that subset. The existing support for satellite imagery could potentially be augmented with options to view aerial or street-view imagery to take advantage of the geographic pervasiveness of geo-located visual media that is now available on the Web.

Chapter 7

Conclusions

In this dissertation, we have explored the usage of table structure to achieve better understanding of table contents. Geographic tables, in particular, were shown to be a previously untapped resource, while supporting a rich set of metadata annotations, query types, and visualizations. In this chapter, we summarize our contributions and suggest several avenues for further work.

Chapter 2 introduced our structural decomposition method for tables, based on the expectation of coherence between cells that appear in the same functional areas within tables. Several row classification methods were discussed, including a probabilistic grammar and others that used conditional random fields. The classification accuracy was improved by adopting a logarithmic binning feature encoding and was experimentally validated by showing a substantial increase in accuracy when compared with Google's WebTables technique on classifying row functions in a complete table. This method served as the basis for data table extraction in each of the following chapters.

In Chapter 3, we examined a new approach to document geotagging, the CHPC method, which took advantage of the presumed consistency between entities that appear in a single table. The method involved selecting a geographic category for a table from a place taxonomy, based on how well the category fit with potential geographic interpretations of the toponyms,

and individual toponyms are then resolved within the chosen category. This method was shown to perform with much higher accuracy than alternative methods that do not make use of the full place taxonomy.

Chapter 4 focused on a particular type of data table: the travel itinerary. We applied some domain knowledge about common itinerary traits in order to effectively classify tables as itineraries or non-itineraries. The most useful features in making this determination were related to the ordering of table rows—specifically, whether the order in which locations appeared resulted in an efficient path. Path efficiency was quantified using a technique that had previously been developed as an optimization for the traveling salesman problem, and the method was able to successfully identify many itineraries from a large corpus of data tables.

Treating geographic tables as point sets gives rise to several interesting queries. In Chapter 5, a scheme was presented for identifying point sets with a similar distribution to a query point set. The technique relied on an optimistic estimate of the Hausdorff distance between two point sets and used a branch-and-bound technique to limit the computation required for similarity queries. An evaluation of the performance of our method showed a substantial improvement over an alternative method using a weaker lower bound estimate, both for the Hausdorff distance and the outlier-resistant modified Hausdorff distance.

Finally, we developed several means of visualizing and interacting with geographic tables, which were presented in Chapter 6: (i) automated itinerary layout, (ii) similarity searching on geotagged tables, and (iii) parallel detail views for visual inspection of location sets found in tables. Each visualization method aided in the spatial or visual inspection of geographic tables by presenting a corresponding map-based view.

Several open problems persist in the area of table understanding, of which we draw attention to three of the most important. First, while our method for functional analysis of complex tables improves on prior art, there remains a need for a unified theory of table composition (and decomposition). An ideal theory would apply a hierarchical representation

of table structures that includes both high-level structures that cover many table cells and fine-grained structures within individual cells, in order to explain the rich variety of structures found in the constituent parts of a data table. As we believe that table generation mechanisms are closer to natural language sentence processes than they are to formal grammar rules, taking inspiration from NLP's grammar induction methods may be helpful.

Second, we wonder if there is a probabilistic model of column coherence that is general enough to simultaneously apply to disparate knowledge domains. The CHPC method is especially well-formulated for geographic disambiguation because geographic entities can be described and compared using a small number of attributes (i.e., their geographic containers, geographic types, and prominence). Other types of entities vary in more subtle and less clear ways, making comparisons between and categorizations of sets of entities more difficult. Existing ontologies that cover broad knowledge domains could provide a starting point for understanding the ways in which entities are related and thus support accurate disambiguation of entities in data tables.

Third, just as the spreadsheet brought data tables to light as a popular method for communicating structured data on a computer, a system for interacting with generic data tables that achieves broad appeal could spur increased interest in methods of utilizing data tables on the Web. Such a system would make it easy to search and browse data tables, support primitives that we expect of standard relational databases such as joining data from multiple sources, and allow interacting with tables of known types in a manner that we expect based on the knowledge domain (such as map-based systems for geographic data).

In spite of some unresolved challenges, as the prevalence of data tables continues to increase, there is no choice but to develop better tools for understanding, processing, and interacting with the data that they hold. I hope and expect that the techniques described here will influence the development of future research that can harness data tables as a valuable resource.

References

[1] M. D. Adelfio, S. Nutanong, and H. Samet. Searching Web Documents as Location Sets. In *Proceedings of the 19th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems (GIS'11)*. Chicago, IL, Nov. 2011, pp. 489–492.

[2] M. D. Adelfio and H. Samet. GeoWhiz: Toponym Resolution Using Common Categories. In *Proceedings of the 21th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems (SIGSPATIAL'13)*. Orlando, FL, Nov. 2013, pp. 542–545.

[3] M. D. Adelfio and H. Samet. Schema Extraction for Tabular Data on the Web. *Proceedings of the VLDB Endowment*, 6(6) Apr. 2013, pp. 421–432.

[4] M. D. Adelfio, S. Nutanong, and H. Samet. Similarity Search on a Large Collection of Point Sets. In *Proceedings of the 19th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems (GIS'11)*. Chicago, IL, Nov. 2011, pp. 132–141.

[5] M. D. Adelfio and H. Samet. Automated Tabular Itinerary Visualization. In *Proceedings of the 22nd ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems (SIGSPATIAL'14)*. Dallas, TX, Nov. 2014, pp. 593–596.

[6] M. D. Adelfio and H. Samet. Itinerary Recognition: Travelers, like Traveling Salesmen, Prefer Efficient Routes. In *Proceedings of the 8th ACM SIGSPATIAL Workshop on Geographic Information Retrieval (GIR'14)*. Dallas, TX, Nov. 2014.

[7] M. D. Adelfio and H. Samet. Structured Toponym Resolution Using Combined Hierarchical Place Categories. In *Proceedings of the 7th ACM SIGSPATIAL Workshop on Geographic Information Retrieval (GIR'13)*. Orlando, FL, Nov. 2013.

[8] M. Agrawala and C. Stolte. Rendering Effective Route Maps: Improving Usability Through Generalization. In *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH'01)*. Los Angeles, CA, Aug. 2001, pp. 241–249.

[9] H. Alt, B. Behrends, and J. Blömer. Approximate Matching of Polygonal Shapes. In *Proceedings of the 7th Annual Symposium on Computational Geometry (SoCG'91)*. North Conway, NH, June 1991, pp. 186–193.

[10] H. Alt, P. Brass, M. Godau, C. Knauer, and C. Wenk. Computing the Hausdorff Distance of Geometric Patterns and Shapes. *Discrete and Computational Geometry*, 25 2003, pp. 65–76.

[11] E. Amitay, N. Har'El, R. Sivan, and A. Soffer. Web-a-Where: Geotagging Web Content. In *Proceedings of the 27th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR'04)*. Sheffield, UK, July 2004, pp. 273–280.

[12] E. Bart. Parsing Tables by Probabilistic Modeling of Perceptual Cues. In *Proceedings of the 10th IAPR International Workshop on Document Analysis Systems (DAS'12)*. IEEE. Gold Coast, Queensland, Australia, Mar. 2012, pp. 409–414.

[13] G. D. Battista, P. Eades, R. Tamassia, and I. G. Tollis. *Graph Drawing: Algorithms for the Visualization of Graphs*. Upper Saddle River, NJ: Prentice Hall PTR, 1998.

[14] N. Beckmann, H.-P. Kriegel, R. Schneider, and B. Seeger. The R*-Tree: An Efficient and Robust Access Method for Points and Rectangles. In *Proceedings of the 1990 ACM SIGMOD International Conference on Management of Data (SIGMOD'90)*. Atlantic City, NJ, May 1990, pp. 322–331.

[15] I. Bensalem and M. K. Kholladi. Toponym Disambiguation by Arborescent Relationships. *Journal of Computer Science*, 6(6) June 2010, pp. 653–659.

[16] P. A. Bernstein, J. Madhavan, and E. Rahm. Generic Schema Matching, Ten Years Later. *Proceedings of the VLDB Endowment*, 4(11) Aug. 2011, pp. 695–701.

[17] L. Breiman, J. Friedman, C. J. Stone, and R. A. Olshen. *Classification and Regression Trees*. New York: Chapman & Hall, 1984.

[18] D. Buscaldi and P. Rosso. Map-Based vs. Knowledge-Based Toponym Disambiguation. In *Proceedings of the 5th ACM Workshop on Geographic Information Retrieval (GIR'08)*. Napa Valley, CA, Oct. 2008, pp. 19–22.

[19] M. J. Cafarella, A. Y. Halevy, and N. Khoussainova. Data Integration for the Relational Web. In *Proceedings of the 35th International Conference on Very Large Data Bases (VLDB'09)*. Lyon, France, Aug. 2009, pp. 1090–1101.

[20] M. J. Cafarella, A. Halevy, D. Z. Wang, E. Wu, and Y. Zhang. Uncovering the Relational Web. In *Proceedings of the 11th International Workshop on Web and Databases (WebDB'08)*. Vancouver, Canada, June 2008.

[21] M. J. Cafarella, A. Halevy, D. Z. Wang, E. Wu, and Y. Zhang. WebTables: Exploring the Power of Tables on the Web. In *Proceedings of the 34th International Conference on Very Large Data Bases (VLDB'08)*. Auckland, New Zealand, Aug. 2008, pp. 538–549.

[22] H.-H. Chen, S.-C. Tsai, and J.-H. Tsai. Mining Tables from Large Scale HTML Texts. In *Proceedings of the 18th International Conference on Computational Linguistics (COLING'00)*. Saarbrücken, Germany, July 2000, pp. 166–172.

[23] J. Chen, J.-Y. Pan, and S. Papadimitriou. TSum: Fast, Principled Table Summarization. In *Proceedings of the Seventh International Workshop on Data Mining for Online Advertising (ADKDD'13)*. Chicago, IL, Aug. 2013.

[24] E. F. Codd. A Relational Model of Data for Large Shared Data Banks. *Communications of the ACM*, 13(6) June 1970, pp. 377–387.

[25] A. Corral, Y. Manolopoulos, Y. Theodoridis, and M. Vassilakopoulos. Closest Pair Queries in Spatial Databases. In *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data (SIGMOD'00)*. Dallas, TX, May 2000, pp. 189–200.

[26] C. Cortes and V. Vapnik. Support-Vector Networks. *Machine Learning*, 20(3) Sept. 1995, pp. 273–297.

[27] E. Crestan and P. Pantel. Web-Scale Table Census and Classification. In *Proceedings of the Fourth ACM International Conference on Web Search and Data Mining (WSDM'11)*. Hong Kong, China, Feb. 2011, pp. 545–554.

[28] G. A. Croes. A Method for Solving Traveling-Salesman Problems. *Operations Research*, 6(6) Nov. 1958, pp. 791–812.

[29] I. F. Cruz, V. R. Ganesh, and S. I. Mirrezaei. Semantic Extraction of Geographic Data from Web Tables for Big Data Integration. In *Proceedings of the 7th Workshop on Geographic Information Retrieval (GIR'13)*. Orlando, FL, Nov. 2013, pp. 19–26.

[30] A. Das Sarma, L. Fang, N. Gupta, A. Halevy, H. Lee, F. Wu, R. Xin, and C. Yu. Finding Related Tables. In *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data (SIGMOD'12)*. Scottsdale, AZ, May 2012, pp. 817–828.

[31] M. De Choudhury, M. Feldman, S. Amer-Yahia, N. Golbandi, R. Lempel, and C. Yu. Automatic Construction of Travel Itineraries Using Social Breadcrumbs. In *Proceedings of the 21st ACM Conference on Hypertext and Hypermedia (HT'10)*. Toronto, Canada, June 2010, pp. 35–44.

[32] M. Dredze, P. P. Talukdar, and K. Crammer. Sequence Learning from Data with Multiple Labels. In *Proceedings of the 1st International Workshop on Learning from Multi-Label Data at ECML/PKDD*. Bled, Slovenia, Sept. 2009.

[33] R. O. Duda, P. E. Hart, and D. G. Stork. *Pattern Classification*. Second. New York: Wiley Interscience, 2000.

[34] H. Elmeleegy, J. Madhavan, and A. Halevy. Harvesting Relational Tables from Lists on the Web. *Proceedings of the VLDB Endowment*, 2(1) Aug. 2009, pp. 1078–1089.

[35] D. W. Embley, M. Hurst, D. P. Lopresti, and G. Nagy. Table-Processing Paradigms: A Research Survey. *International Journal on Document Analysis and Recognition*, 8(2) June 2006, pp. 66–86.

[36] M. Erwig, R. Abraham, I. Cooperstein, and S. Kollmansberger. Automatic Generation and Maintenance of Correct Spreadsheets. In *Proceedings of the 27th International Conference on Software Engineering (ICSE'05)*. St. Louis, MO, May 2005, pp. 136–145.

[37] T. M. J. Fruchterman and E. M. Reingold. Graph Drawing by Force-Directed Placement. *Software: Practice and Experience*, 21(11) Nov. 1991, pp. 1129–1164.

[38] W. Gatterbauer, P. Bohunsky, M. Herzog, B. Krüpl, and B. Pollak. Towards Domain-Independent Information Extraction from Web Tables. In *Proceedings of the 16th International Conference on World Wide Web (WWW'07)*. Banff, Canada, May 2007, pp. 71–80.

[39] *GeoNames*. http://geonames.org/.

[40] R. Gupta and S. Sarawagi. Answering Table Augmentation Queries from Unstructured Lists on the Web. *Proceedings of the VLDB Endowment*, 2(1) Aug. 2009, pp. 289–300.

[41] M. Guthe, P. Borodin, and R. Klein. Fast and Accurate Hausdorff Distance Calculation Between Meshes. *Journal of WSCG*, 13(2) 2005, pp. 41–48.

[42] A. Guttman. R-Trees: A Dynamic Index Structure for Spatial Searching. In *Proceedings of the 1984 ACM SIGMOD International Conference on Management of Data (SIGMOD'84)*. Boston, MA, June 1984, pp. 47–57.

[43] L. Han, T. Finin, C. Parr, J. Sachs, and A. Joshi. RDF123: From Spreadsheets to RDF. In *Proceedings of the 7th International Semantic Web Conference (ISWC'08)*. Karlsruhe, Germany, Oct. 2008, pp. 451–466.

[44] T. Hassan and R. Baumgartner. Table Recognition and Understanding from PDF Files. In *Proceedings of the Ninth International Conference on Document Analysis and Recognition (ICDAR'07)*. Vol. 2. Curitiba, Brazil, Sept. 2007, pp. 1143–1147.

[45] I. Herman, G. Melancon, and M. Marshall. Graph Visualization and Navigation in Information Visualization: A Survey. *IEEE Transactions on Visualization and Computer Graphics*, 6(1) Jan. 2000, pp. 24–43.

[46] G. Hignette, P. Buche, J. Dibie-Barthélemy, and O. Haemmerlé. Fuzzy Annotation of Web Data Tables Driven by a Domain Ontology. In *Proceedings of the 6th European Semantic Web Conference on The Semantic Web: Research and Applications (ESWC'09)*. Heraklion, Crete, Greece, May 2009, pp. 638–653.

[47] G. R. Hjaltason and H. Samet. Distance Browsing in Spatial Databases. *ACM Transactions on Database Systems*, 24(2) June 1999, pp. 265–318.

[48] G. R. Hjaltason and H. Samet. Incremental Distance Join Algorithms for Spatial Databases. In *Proceedings of the 1998 ACM SIGMOD International Conference on Management of Data (SIGMOD'98)*. Seattle, WA, June 1998, pp. 237–248.

[49] G. R. Hjaltason and H. Samet. Index-Driven Similarity Search in Metric Spaces. *ACM Transactions on Database Systems*, 28(4) Dec. 2003, pp. 517–580.

[50] M. F. Hurst. "The Interpretation of Tables in Texts". PhD thesis. Edinburgh, Scotland: University of Edinburgh, 2000.

[51] D. P. Huttenlocher, K. Kedem, and J. M. Kleinberg. On Dynamic Voronoi Diagrams and the Minimum Hausdorff Distance for Point Sets Under Euclidean Motion in the Plane. In *Proceedings of the 8th Annual Symposium on Computational Geometry (SoCG'92)*. Berlin, Germany, June 1992, pp. 110–119.

[52] D. Jannach, K. Shchekotykhin, and G. Friedrich. Automated Ontology Instantiation from Tabular Web Sources—The AllRight System. *Web Semantics*, 7(3) Sept. 2009, pp. 136–153.

[53] P. Jha. "Wang Notation Tool: A Layout Independent Representation of Tables". MA thesis. Rensselaer Polytechnic Institute, Department of Electrical Engineering, 2008.

[54] G. H. John and P. Langley. Estimating Continuous Distributions in Bayesian Classifiers. In *Proceedings of the Eleventh Conference on Uncertainty in Artificial Intelligence (UAI'95)*. Montreal, Canada, Aug. 1995, pp. 338–345.

[55] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by Simulated Annealing. *Science*, 220(4598) May 1983, pp. 671–680.

[56] J. D. Lafferty, A. McCallum, and F. C. N. Pereira. Conditional Random Fields: Probabilistic Models for Segmenting and Labeling Sequence Data. In *Proceedings of the 18th International Conference on Machine Learning (ICML'01)*. Williamstown, MA, June 2001, pp. 282–289.

[57] O. Lassila. The Resource Description Framework. *IEEE Intelligent Systems*, 15(6) Nov. 2000, pp. 67–69.

[58] J. L. Leidner. “Toponym Resolution in Text: Annotation, Evaluation and Applications of Spatial Grounding of Place Names”. PhD thesis. Edinburgh, Scotland: University of Edinburgh, School of Informatics, 2007.

[59] K. Lerman, L. Getoor, S. Minton, and C. Knoblock. Using the Structure of Web Sites for Automatic Segmentation of Tables. In *Proceedings of the 2004 ACM SIGMOD International Conference on Management of Data (SIGMOD’04)*. Paris, France, June 2004, pp. 119–130.

[60] M. D. Lieberman and H. Samet. Supporting Rapid Processing and Interactive Map-Based Exploration of Streaming News. In *Proceedings of the 20th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems (SIGSPATIAL’12)*. Redondo Beach, CA, Nov. 2012, pp. 179–188.

[61] M. D. Lieberman and H. Samet. Adaptive Context Features for Toponym Resolution in Streaming News. In *Proceedings of the 35th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR’12)*. Portland, OR, Aug. 2012, pp. 731–740.

[62] M. D. Lieberman and H. Samet. Multifaceted Toponym Recognition for Streaming News. In *Proceedings of the 34th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR’11)*. Beijing, China, July 2011, pp. 843–852.

[63] M. D. Lieberman, H. Samet, and J. Sankaranarayanan. Geotagging with Local Lexicons to Build Indexes for Textually-Specified Spatial Data. In *Proceedings of the 26th International Conference on Data Engineering (ICDE’10)*. Long Beach, CA, Mar. 2010, pp. 201–212.

[64] M. D. Lieberman, H. Samet, J. Sankaranarayanan, and J. Sperling. Spatio-Textual Spreadsheets: Geotagging via Spatial Coherence. In *Proceedings of the 17th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems (GIS’09)*. Seattle, WA, Nov. 2009, pp. 524–527.

[65] M. D. Lieberman, H. Samet, J. Sankaranarayanan, and J. Sperling. STEWARD: Architecture of a Spatio-Textual Search Engine. In *Proceedings of the 15th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems (GIS’07)*. Seattle, WA, Nov. 2007, pp. 186–193.

[66] M. D. Lieberman, H. Samet, and J. Sankaranayananan. Geotagging: Using Proximity, Sibling, and Prominence Clues to Understand Comma Groups. In *Proceedings of the 6th Workshop on Geographic Information Retrieval (GIR’10)*. Zurich, Switzerland, Nov. 2010, 6:1–6:8.

[67] G. Limaye, S. Sarawagi, and S. Chakrabarti. Annotating and Searching Web Tables Using Entities, Types and Relationships. *Proceedings of the VLDB*, 3(1) Sept. 2010, pp. 1338–1347.

[68] S. Lin and B. W. Kernighan. An Effective Heuristic Algorithm for the Traveling-Salesman Problem. *Operations Research*, 21(2) Mar. 1973, pp. 498–516.

[69] R. Lipikorn, A. Shimizu, and H. Kobatake. A Modified Exoskeleton and a Hausdorff Distance Matching Algorithm for Shape-Based Object Recognition. In *Proceedings of the International Conference on Imaging Science, Systems and Technology (CISST’03)*. Las Vegas, NV, June 2003, pp. 507–511.

[70] Y. Liu, K. Bai, P. Mitra, and C. L. Giles. TableSeer: Automatic Table Metadata Extraction and Searching in Digital Libraries. In *Proceedings of the 7th ACM/IEEE Joint Conference on Digital Libraries (JCDL'07)*. Vancouver, Canada, June 2007, pp. 91–100.

[71] J. Madhavan, P. Bernstein, K. Chen, A. Halevy, and P. Shenoy. Corpus-based Schema Matching. In *Proceedings of the 19th International Conference on Data Engineering (ICDE'03)*. Bangalore, India, Mar. 2003, pp. 57–68.

[72] B. Martins, H. Manguinhas, and J. Borbinha. Extracting and Exploring the Geo-Temporal Semantics of Textual Resources. In *Proceedings of the 2nd IEEE International Conference on Semantic Computing (ICSC'08)*. Santa Clara, CA, Aug. 2008, pp. 1–9.

[73] K. S. McCurley. Geospatial Mapping and Navigation of the Web. In *Proceedings of the Tenth International World Wide Web Conference (WWW'01)*. Hong Kong, May 2001, pp. 221–229.

[74] V. Mulwad, T. Finin, and A. Joshi. A Domain Independent Framework for Extracting Linked Semantic Data from Tables. In *Search Computing*. Vol. 7538. 2012, pp. 16–33.

[75] D. Nadeau, P. Turney, and S. Matwin. Unsupervised Named-Entity Recognition: Generating Gazetteers and Resolving Ambiguity. In *Canadian AI*. Quebec City, Canada, June 2006, pp. 266–277.

[76] S. Nutanong, E. H. Jacox, and H. Samet. An Incremental Hausdorff Distance Calculation Algorithm. *Proceedings of the VLDB Endowment*, 4(8) May 2011, pp. 506–517.

[77] N. Okazaki. *CRFsuite: A Fast Implementation of Conditional Random Fields (CRFs)*. http://www.chokkan.org/software/crfsuite/. 2007.

[78] S. E. Overell and S. M. Rüger. Using Co-Occurrence Models for Placename Disambiguation. *International Journal of Geographical Information Science*, 22(3) Mar. 2008, pp. 265–287.

[79] D. Papadias, Y. Tao, K. Mouratidis, and C. K. Hui. Aggregate Nearest Neighbor Queries in Spatial Databases. *ACM Transactions on Database Systems*, 30(2) June 2005, pp. 529–576.

[80] R. Pimplikar and S. Sarawagi. Answering Table Queries on the Web using Column Keywords. *Proceedings of the VLDB Endowment*, 5(10) June 2012, pp. 908–919.

[81] D. Pinto, A. McCallum, X. Wei, and W. B. Croft. Table Extraction Using Conditional Random Fields. In *Proceedings of the 26th Annual International Conference on Research and Development in Information Retrieval (SIGIR'03)*. Toronto, Canada, July 2003, pp. 235–242.

[82] H. C. Purchase. Which Aesthetic Has the Greatest Effect on Human Understanding? In *Proceedings of the 5th International Symposium on Graph Drawing (GD'97)*. Rome, Italy, Sept. 1997, pp. 248–261.

[83] R. S. Purves, P. Clough, C. B. Jones, A. Arampatzis, B. Bucher, D. Finch, G. Fu, H. Joho, A. K. Syed, S. Vaid, and B. Yang. The Design and Implementation of SPIRIT: A Spatially Aware Search Engine for Information Retrieval on the Internet. *International Journal of Geographical Information Science*, 21(7) Aug. 2007, pp. 717–745.

[84] T. Qin, R. Xiao, L. Fang, X. Xie, and L. Zhang. An Efficient Location Extraction Algorithm by Leveraging Web Contextual Information. In *Proceedings of the 18th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*. San Jose, CA, Nov. 2010, pp. 53–60.

[85] G. Quercini and C. Reynaud. Entity Discovery and Annotation in Tables. In *Proceedings of the 16th International Conference on Extending Database Technology (EDBT'13)*. Genoa, Italy, Mar. 2013, pp. 693–704.

[86] G. Quercini, H. Samet, J. Sankaranarayanan, and M. D. Lieberman. Determining the Spatial Reader Scopes of News Sources Using Local Lexicons. In *Proceedings of the 18th SIGSPATIAL International Conference on Advances in Geographic Information Systems (GIS'10)*. San Jose, CA, Nov. 2010, pp. 43–52.

[87] E. Rauch, M. Bukatin, and K. Baker. A Confidence-Based Framework for Disambiguating Geographic Terms. In *Proceedings of the HLT-NAACL 2003 Workshop on Analysis of Geographic References (GEOREF'03)*. Edmonton, Canada, May 2003.

[88] S. Reddy, K. Shilton, G. Denisov, C. Cenizal, D. Estrin, and M. Srivastava. Biketastic: Sensing and mapping for better biking. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (SIGCHI'10)*. Atlanta, GA, Apr. 2010, pp. 1817–1820.

[89] L. Rokach and O. Maimon. *Data Mining with Decision Trees: Theory and Applications*. New York: World Scientific, 2008.

[90] M. Rosenblatt. Remarks on Some Nonparametric Estimates of a Density Function. *The Annals of Mathematical Statistics*, 27(3) Sept. 1956, pp. 832–837.

[91] N. Roussopoulos, S. Kelley, and F. Vincent. Nearest Neighbor Queries. In *Proceedings of the 1995 ACM SIGMOD International Conference on Management of Data (SIGMOD'95)*. San Jose, CA, May 1995, pp. 71–79.

[92] H. Samet, J. Sankaranarayanan, M. D. Lieberman, M. D. Adelfio, B. C. Fruin, J. M. Lotkowski, D. Panozzo, J. Sperling, and B. E. Teitler. Reading News with Maps by Exploiting Spatial Synonyms. *Communications of the ACM*, 57(10) Oct. 2014, pp. 64–77.

[93] H. Samet. *Foundations of Multidimensional and Metric Data Structures*. San Francisco: Morgan-Kaufmann, 2006.

[94] T. W. Sederberg and S. R. Parry. Comparison of Three Curve Intersection Algorithms. *Computer Aided Design*, 18(1) Jan. 1986, pp. 58–63.

[95] S. Seth, R. Jandhyala, M. Krishnamoorthy, and G. Nagy. Analysis and Taxonomy of Column Header Categories for Web Tables. In *Proceedings of the 9th IAPR International Workshop on Document Analysis Systems (DAS'10)*. Boston, MA, June 2010, pp. 81–88.

[96] F. Sha and F. C. N. Pereira. Shallow Parsing with Conditional Random Fields. In *Proceedings of the Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics (HLT-NAACL'03)*. Edmonton, Canada, May 2003, pp. 213–220.

[97] H. Shin, B. Moon, and S. Lee. Adaptive Multi-Stage Distance Join Processing. In *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data (SIGMOD'00)*. Dallas, TX, May 2000, pp. 343–354.

[98] A. C. e Silva, A. M. Jorge, and L. Torgo. Design of an End-to-End Method to Extract Information from Tables. *International Journal of Document Analysis and Recognition*, 8(2-3) June 2006, pp. 144–171.

[99] J. Strötgen, M. Gertz, and P. Popov. Extraction and Exploration of Spatio-Temporal Information in Documents. In *Proceedings of the 6th Workshop on Geographic Information Retrieval (GIR'10)*. Zurich, Switzerland, Feb. 2010, pp. 16–23.

[100] M. Tang, M. Lee, and Y. J. Kim. Interactive Hausdorff Distance Computation for General Polygonal Models. *ACM Transactions on Graphics*, 28(3) Aug. 2009.

[101] C. Tao and D. W. Embley. Automatic Hidden-Web Table Interpretation, Conceptualization, and Semantic Annotation. *Data & Knowledge Engineering*, 68(7) July 2009, pp. 683–703.

[102] B. Teitler, M. D. Lieberman, D. Panozzo, J. Sankaranarayanan, H. Samet, and J. Sperling. NewsStand: A New View on News. In *Proceedings of the 16th SIGSPATIAL International Conference on Advances in Geographic Information Systems (GIS'08)*. Irvine, CA, Nov. 2008, pp. 144–153.

[103] Y. A. Tijerino, D. W. Embley, D. W. L. Y. Ding, and G. Nagy. Towards Ontology Generation from Tables. *World Wide Web*, 8(3) Sept. 2005, pp. 261–285.

[104] G. Tsoumakas and I. Katakis. Multi-Label Classification: An Overview. *International Journal of Data Warehousing and Mining*, 3(3) 2007, pp. 1–13.

[105] P. Venetis, A. Halevy, J. Madhavan, M. Paşca, W. Shen, F. Wu, G. Miao, and C. Wu. Recovering Semantics of Tables on the Web. *Proceedings of the VLDB Endowment*, 4(9) June 2011, pp. 528–538.

[106] J. Wang, H. Wang, Z. Wang, and K. Q. Zhu. Understanding Tables on the Web. In *Proceedings of the 31st International Conference on Conceptual Modeling (ER'12)*. Florence, Italy, Oct. 2012, pp. 141–155.

[107] X. Wang. "Tabular Abstraction, Editing, and Formatting". PhD thesis. University of Waterloo, Department of Computer Science, 1996.

[108] X. Wang and D. Wood. An Abstract Model for Tables. *Communications of the TEXUsers Group (TUGboat'93)*, 14(3) Oct. 1993, pp. 231–237.

[109] Y. Wang and J. Hu. A Machine Learning Based Approach for Table Detection on the Web. In *Proceedings of the 11th International Conference on World Wide Web (WWW'02)*. Honolulu, HI, May 2002, pp. 242–250.

[110] S. Winter and C. Freksa. Approaching the Notion of Place by Contrast. *Journal of Spatial Information Science*, 5(1) July 2012, pp. 31–50.

[111] J. Wood, A. Slingsby, and J. Dykes. Visualizing the Dynamics of London's Bicycle-Hire Scheme. *Cartographica: The International Journal for Geographic Information and Geovisualization*, 46(4) Nov. 2011, pp. 239–251.

[112] C. Xia, H. Lu, B. C. Ooi, and J. Hu. Gorder: An Efficient Method for KNN Join Processing. In *Proceedings of the Thirtieth International Conference on Very Large Data Bases (VLDB'04)*. Toronto, Canada, Sept. 2004, pp. 756–767.

[113] M. Yakout, K. Ganjam, K. Chakrabarti, and S. Chaudhuri. InfoGather: Entity Augmentation and Attribute Discovery by Holistic Matching with Web Tables. In *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data (SIGMOD'12)*. Scottsdale, AZ, May 2012, pp. 97–108.

[114] H. Yoon, Y. Zheng, X. Xie, and W. Woo. Smart Itinerary Recommendation Based on User-Generated GPS Trajectories. In *Ubiquitous Intelligence and Computing*. Vol. 6406. Oct. 2010, pp. 19–34.

[115] R. Zanibbi, D. Blostein, and J. R. Cordy. A Survey of Table Recognition: Models, Observations, Transformations, and Inferences. *International Journal on Document Analysis and Recognition*, 7(1) Mar. 2004, pp. 1–16.