

Ranked Similarity Search of Scientific Datasets:  
An Information Retrieval Approach

by

Veronika Margaret Megler

A dissertation submitted in partial fulfillment of the  
requirements for the degree of

Doctor of Philosophy  
in  
Computer Science

Dissertation Committee:  
David Maier, Chair  
Alon Y. Halevy  
Lois M.L. Delcambre  
Jiunn-Der Duh

Portland State University  
2014

© 2014 Veronika Margaret Megler

## **Abstract**

In the past decade, the amount of scientific data collected and generated by scientists has grown dramatically. This growth has intensified an existing problem: in large archives consisting of datasets stored in many files, formats and locations, how can scientists find data relevant to their research interests? We approach this problem in a new way: by adapting Information Retrieval techniques, developed for searching text documents, into the world of (primarily numeric) scientific data. We propose an approach that uses a blend of automated and curated methods to extract metadata from large repositories of scientific data. We then perform searches over this metadata, returning results ranked by similarity to the search criteria. We present a model of this approach, and describe a specific implementation thereof performed at an ocean-observatory data archive and now running in production. Our prototype implements scanners that extract metadata from datasets that contain different kinds of environmental observations, and a search engine with a candidate similarity measure for comparing a set of search terms to the extracted metadata. We evaluate the utility of the prototype by performing two user studies; these studies show that the approach resonates with users, and that our proposed similarity measure performs well when analyzed using standard Information Retrieval evaluation methods. We performed performance tests to explore how continued archive growth will affect our goal of interactive response, developed and applied techniques that mitigate the effects of that growth, and show that the techniques are effective. Lastly, we describe some of the research needed to extend this initial work into a true “Google for data”.

## **Dedication**

To my father:

Dr. Vendelin R. Megler

July 26 1921 – July 26 2011

May he rest in peace

I know we both wish he were here to celebrate this accomplishment with me

*“Let us not follow where the path may lead.*

*Let us go instead where there is no path,*

*And leave a trail.”*

*Japanese Proverb*

## **Acknowledgements**

I would like to express the deepest appreciation to my advisor, Professor David Maier. He patiently and tirelessly worked through a number of extremely challenging organizational issues that surrounded my studies, and supported me while I dealt with an unusually large number of “major life events”. I also appreciate the good humor with which he handled working with someone he sometimes claimed managed him more than he managed her.

I also thank my committee members for their interest in my work, their advice and support, and apologize for the length of my proposal and dissertation.

I have had much support from within Portland State University. Professor Kristin Tufte was a stalwart friend and supporter, allowing me to vent when needed and providing perspective and insight. Ben Sanabria, Justin Corn and particularly Basem Elazzabi helped with various parts of Data Near Here prototyping, testing and implementation, allowing me to handle far more file formats and special cases than otherwise would have been possible in the same time. Scott Brittell was my go-to person for technical support, while the members of the Computer Action Team were patient with my non-traditional technical-development platform.

Professor Alistair Moffat, from my alma mater Melbourne University in Australia, generously contributed his time and support in performing a very detailed review of the user-study results. His support gave me faith that my interpretation of the strongly positive results was valid.

The scientists at the Center for Coastal Margin Observation and Prediction (CMOP) at OHSU brought the problem to our attention, and were generous with their time (and unstinting in their advice), and the CMOP cyber-team was invaluable in getting Data Near Here (DNH) deployed and integrated with other data tools.

This work was supported by NSF grant OCE-0424602 and by an Intel-MCECS Graduate Scholarship.

## Table of Contents

Abstract .....	i
Dedication .....	ii
Acknowledgements .....	iii
List of Tables .....	xi
List of Figures .....	xii
1      Introduction .....	1
1.1     Searching for Observed Data .....	3
1.2     An Information Retrieval Approach .....	6
1.3     Contributions .....	9
1.4     Terminology .....	14
2      Background and Overview .....	18
2.1     Motivation .....	18
2.2     Information Retrieval Concepts and Scientific Data Search.....	19
2.3     Our Study Archive .....	21
2.4     Current Approaches to Searching for Scientific Data.....	24
2.5     Archives, Portals and Gateways.....	29
2.5.1     Data Access and Sharing .....	29
2.5.2     Common Characteristics of Scientific Archives.....	33
2.6     Adapting Web Information Retrieval Approaches.....	35

2.6.1	A Notion of Dataset Similarity .....	39
2.6.2	Hierarchies of Scale .....	40
2.7	Related Work.....	44
3	Model.....	52
3.1	Feature Extraction .....	53
3.2	Similarity Scoring .....	57
3.3	Adaptability: Across Dataset and Search Granularities .....	63
3.4	Discussion .....	67
4	Dataset Similarity and Metadata Extraction .....	70
4.1	Dataset Similarity.....	70
4.1.1	Data Distance as Similarity.....	71
4.1.2	Dataset Similarity: The Intuition .....	74
4.1.3	Estimating Dataset Similarity .....	78
4.1.4	Geometric and Geospatial Similarity.....	83
4.1.5	Algorithm for Dataset Scoring .....	87
4.1.6	Related Work: Dataset Similarity .....	92
4.2	Making Metadata.....	96
4.2.1	The Metadata-Creation Challenge .....	97
4.2.2	Representing A Dataset Collection: Dataset Summaries.....	99
4.2.3	Hierarchical Metadata.....	101
4.2.4	Using Hierarchical Metadata .....	103

4.2.5	Experiences in Creating Metadata .....	105
4.2.6	Related Work: Metadata Extraction.....	108
4.3	Evaluation.....	110
5	Prototype Design and Architecture .....	111
5.1	High-Level Architecture .....	111
5.2	Current Implementation .....	115
5.2.1	Implementation Architecture .....	116
5.2.2	Data Model.....	123
5.2.3	Making Metadata .....	126
5.2.4	Catalog Contents .....	128
5.2.5	Search Interface Components .....	132
5.2.6	Plans and Extensions: Data Access.....	137
5.2.7	Extensions: Variable and Unit Names .....	138
5.2.8	Extensions: Moving from Prototype to Product .....	139
5.3	Architectural Evaluation .....	140
5.3.1	Investigation and Analysis.....	142
5.3.2	Testing.....	144
5.3.3	Results Summary .....	152
6	User Evaluation.....	155
6.1	Measuring Relevance .....	156
6.2	User Study 1: Testing the Similarity Measure .....	158

6.2.1	Methods.....	159
6.2.2	Results.....	160
6.2.3	Discussion .....	164
6.3	User Study 2: Fidelity .....	166
6.3.1	Methods.....	167
6.3.2	Results.....	171
6.4	Related Work.....	181
6.5	Discussion of the User Studies.....	182
7	Performance and Scalability .....	186
7.1	Background to Performance Evaluation .....	187
7.1.1	Metadata Catalog versus Index.....	189
7.1.2	Top-k Evaluation Techniques .....	191
7.1.3	Filter-Restart and Cutoff Scores .....	196
7.2	Basic Algorithm .....	200
7.3	Adding a Filter .....	203
7.4	Relaxation.....	207
7.4.1	Naïve Relaxation.....	209
7.4.2	Adaptive Relaxation.....	210
7.4.3	Contraction.....	212
7.5	Performance Tests: Methods and Data.....	213
7.5.1	Test Collections .....	216

7.5.2	Test Hierarchies .....	217
7.5.3	Search Suites .....	220
7.6	Relaxation Performance Test Results .....	221
7.6.1	Time Search Suite Results .....	221
7.6.2	Space Search Suite Results .....	226
7.6.3	Space-Time Search Suite Results .....	229
7.7	Hierarchy Performance Test Results.....	231
7.7.1	Time Search Suite Results .....	232
7.7.2	Space Search Suite Results .....	234
7.7.3	Space-Time Search Suite Results .....	240
7.8	Other Informal Performance Tests.....	242
7.9	Discussion .....	244
7.9.1	Effect of Filter-Restart Techniques.....	245
7.9.2	Effect of Hierarchies .....	246
7.9.3	Scaling Beyond a Single Server.....	250
7.9.4	Current Deployment.....	252
8	Future Research and Conclusions.....	253
8.1	Feature Extraction .....	253
8.2	The Metadata Mess .....	255
8.3	The Variability of Variable Names .....	258
8.4	Increasing Search Sophistication .....	263

8.5	Scalability.....	265
8.6	Towards Universal Data Search.....	266
8.7	Conclusion.....	269
	References.....	272
	Appendix A: DNH PostgreSQL Database Indexes .....	281
	Appendix B: Variable List .....	282

## **List of Tables**

Table 3.1. Model Component Dependencies .....	68
Table 5.1. Characterization of Data Near Here Metadata.....	130
Table 5.2 Characterization of Existing Metadata entries by Category .....	130
Table 5.3. Extract Process: Current Implementations .....	131
Table 6.1. Count of Test Collection Catalog Entries .....	168
Table 6.2. Responses to User Satisfaction Questions .....	170
Table 6.3. Precision and Mean Reciprocal Rank (MRR) .....	174
Table 6.4. Comparison of Average RBP Ranges.....	181
Table 7.1. Summary of Test Hierarchy Structures. ....	219
Table 7.2. Time Search Suite Response Times, No Filter vs. Filter.....	222
Table 7.3. Time Search Suite, N vs. AD.....	223
Table 7.4. Time Search Suite, AD Only vs. CN, for searches affected by CN only .....	224
Table 7.5. Space Search Suite, N vs. AD.....	227
Table 7.6. Space Search Suite, AD vs. CN, only for searches affected by CN .....	228
Table 7.7. Space-Time Search Suite, N vs. AD.....	229
Table 7.8. Space-Time Search Suite: AD vs. CN, only for searches affected by CN ....	230
Table 7.9. Time Search Suite: Hierarchy Comparisons.....	232
Table 7.10. Space Search Suite: Response Times by Hierarchy .....	233
Table 7.11. Space-Time Search Suite: Response Times by Hierarchy.....	241
Table 7.12. Hierarchy Comparisons Across Search Terms .....	248

## List of Figures

Figure 1.1. High-level web-search architecture .....	8
Figure 1.2. User interface for “Data Near Here” .....	12
Figure 2.1. Heterogeneity of data formats and data access tools.....	24
Figure 2.2. A centralized gateway architecture in the climate-change community.....	30
Figure 2.3. High-level dataset search architecture.....	37
Figure 2.4. Ideal versus approximated dataset similarity scoring.....	38
Figure 2.5. Example of a dataset hierarchy.....	41
Figure 3.1. Example of a dataset summary .....	54
Figure 3.2. High-level depiction of creating and scoring a dataset summary.....	56
Figure 3.3. Scoring a dataset summary .....	60
Figure 3.4. A set of datasets partitioned and assembled into a hierarchy,.....	65
Figure 3.5. Modifiable components in the architecture .....	67
Figure 4.1. Example of qualitative geospatial and temporal ranking .....	75
Figure 4.2. Graphic depicting a portion of the candidate distance measure .....	80
Figure 4.3 Formulae.....	82
Figure 4.4. Adapting the distance measure .....	84
Figure 4.5. Dataset distance calculation for rectangular search region .....	86
Figure 4.6. Simplified pseudo-code for dataset summary ranked search algorithm.....	88
Figure 4.7. Spatial metadata entries for a mobile station.....	102
Figure 4.8. Scoring example for intermittent data .....	104

Figure 5.1. High-level dataset search architecture .....	111
Figure 5.2. System context and component diagram .....	113
Figure 5.3. “Data Near Here” implementation architecture .....	117
Figure 5.4. Search-execution sequence diagram.....	122
Figure 5.5. Current data model. ....	125
Figure 5.6. Production counts of datasets by hierarchy level and total observations ....	129
Figure 5.7. User interface for Data Near Here.....	133
Figure 5.8. Variable details page for “oxygen” .....	134
Figure 5.9. Dataset details page for a middle hierarchy levels of a cruise .....	136
Figure 5.10. Steps of ATAM .....	142
Figure 5.11. Architecture of the data indirection ABAS .....	143
Figure 6.1. Four examples of spatial dataset summary comparisons .....	160
Figure 6.2. Level of respondent agreement .....	161
Figure 6.3. Percent agreement with distance measure .....	163
Figure 6.4. Second user-study process.....	169
Figure 6.5. Summary results for user-satisfaction survey questions .....	173
Figure 6.6. Proportion of datasets by rating.....	175
Figure 6.7. Condensed discounted cumulative gain .....	178
Figure 7.1. Pseudo-code for basic algorithm .....	202
Figure 7.2. Pseudo-code changes for basic algorithm with filter .....	205
Figure 7.3. Pseudo-code changes for naive relaxation .....	209

Figure 7.4. Pseudo-code changes (from naïve relaxation) for adaptive relaxation .....	210
Figure 7.5. Pseudo-code changes for contraction .....	212
Figure 7.6. Time search suite response times, $e$ hierarchies .....	235
Figure 7.7. Time search suite response times .....	236
Figure 7.8. Space search suite response times .....	237
Figure 7.9. Space-time search suite .....	238

## 1 Introduction

Scientists are frustrated by the limited capabilities currently available to them for locating data. Researchers can spend inordinate time just locating and selecting suitable data sets, before even starting the data analyses that might lead to scientific insights. When a search requires many selections or scanning a large archive, scientists may desist, at cost to their research. Ahrens et al. note, “when datastreams aren’t optimally exploited, scientific discovery is delayed or missed” [5]. The problem of finding scientific data, and dealing with the heterogeneity of multiple data formats, has been widely recognized [27, 33]. Large archives of data only have value commensurate with the use and reuse that can be made of their contents; and data cannot be used if it cannot be found [49, 125, 142]. Growth in data must be accompanied by improvements in tools that help scientists easily find the data they need [48]. Despite much progress in providing data access through portals and gateways, the problem of how a scientist finds the data she feels is worth accessing has not been solved. This problem was highlighted at a National Research Council workshop [142]. In our work with one oceanography observatory and research center, the Center for Coastal Margin Observation and Prediction (CMOP) [147], the scientists brought this issue of finding relevant data to our attention as one of their highest priority problems with CMOP’s scientific data archive – despite having access to state-of-the-art data access, analysis and visualization tools [148].

What scientists desire is a “Google for data.” At the high level, our vision is this: to provide scientists with an interactive search engine that can quickly find data relevant to

their research interests; and further, to let them quickly assess whether the data located is interesting enough to warrant exploring in greater detail. This dissertation describes our first steps towards this vision.

We contribute a novel approach to the problem of finding relevant data: ranked similarity search of scientific datasets. We begin by limiting the space we address to a large subset of scientific data: data made available for public download by scientific archives. The data provided by each individual archive tends to be somewhat limited in nature, in terms of the subject areas addressed, the types of data made available, and the formats used. Each archive has some staff responsible for ensuring data is accessible and has some amount of consistency or quality control. The data formats, types and contents vary widely from archive to archive.

Within this subset, we focus our experimental work on observational data, whether sensor-derived, measured in the laboratory, or generated by scientific models. This subset of scientific data already consists of many petabytes of data, is growing rapidly and is diverse, thus representing at a smaller scale many of the issues of the entire field. Given the large quantity of numeric observational data, and that a significant proportion of other scientific data is numeric, we experiment primarily on numeric observational data. We believe, however, that the work described in this dissertation can be readily extended to other fields or kinds of numeric data, and even to non-numeric data.

## **1.1 Searching for Observed Data**

Scientific research into natural systems has benefited from a rapid increase in the number and types of deployed observational sensors. Some research institutions act as observatories; they manage collections of sensors focused on a single research topic, collecting, storing and making their data available on the Internet. The stored data for the observatory forms a data archive that grows in age, size, and complexity. With billions of historical observations now stored in diverse databases and in thousands of datasets of different formats and in different navigational structures, scientists have difficulty locating data relevant to their research even within a single archive.

A decade ago, this explosion was described as the “Data Deluge” [61], and continued exponential growth in data volumes was predicted [81]. These predictions have now been realized and exceeded, and have led to an increased focus on the issues of “big data.” New big-data proposals are announced daily, and Gartner warns of big-data chaos if new management techniques are not developed to access the variety, velocity and volume of data now available [42]. “Big data” collections, such as observatory archives, represent a large, continuing investment of funds and people. We expect the value of such sources to grow as their holdings increase. Yet there is real danger that each expansion of an archive makes each individual dataset within it more difficult to locate, thus compromising that value.

As data archive sizes grow, traditional methods scientists have used to find data begin to fail. The archive needs a way to help a scientist identify and locate data of interest in its

collection. Each archive now has some data access system – often a website or portal – that a scientist can navigate to find datasets of interest. Some data access systems rely on manual navigation of catalogs (e.g., a THREDDS catalog hierarchy); the scientist is expected to choose the correct option at each step within the catalog hierarchy that will eventually lead to her desired data. Some data access systems rely on Boolean queries for specific words in metadata contributed by the scientist who archived the data. More sophisticated systems allow text search over words in metadata, with the scientist entering words representing her interest and the system returning entries judged most relevant, even if not exactly matching all the search criteria, based on the contributed metadata text.

Much scientific data is numeric in nature. In these cases, unlike with text search of the actual document as we expect in web search, the data access system searches textual metadata contributed by the data provider for the user's terms, treating the metadata (not the actual data) as a text document. Such searches are successful only if the contributed metadata contains words that match those for which the scientist searches, and if she can represent the search in that way.

For geospatial data, one of the better served areas of scientific search due to its importance to many research fields, metadata may include a geospatial extent on the contained data, and search systems rely on only geographic comparisons such as *contains* or *intersects*.

A naïve approach to searching the content of numeric datasets might entail indexing each contained number as though it were a term in a document, and then allowing a scientist to list the numbers she is looking for. However, the searches posed by scientists are often different from searches targeted at text documents. Rather than looking for specific words, scientists are often looking for datasets containing particular variables, such as salinity or nitrogen, that manifest a desired range of data values or that were observed at some location and time. For example, a scientist may be looking for “water temperature between 5 and 10C.” Many datasets may contain the same-named variables but may contain very different values for these variables; reporting only that a dataset contains a specific variable or number (“dataset\_1 contains a ‘5’, a ‘10’, a ‘C’ and the word ‘between’”), as might be determined by a text search, has little utility for the scientist. In fact, a matching dataset may not contain a ‘5’ or a ‘10’ at all, or the data may be in Fahrenheit. This same issue is found even in non-scientific fields, such as in a components portal [4] where users want to retrieve specification documents without exact word-matches between the search terms and the attribute names and associated numeric values in the documents.

The scientist’s desired search results are sets of data, rather than text documents [103]. Once interesting datasets are identified, the datasets are generally further analyzed or visualized using a variety of scientific tools, rather than simply browsed, as is more common for text documents. The scientists have powerful analysis and visualization tools available to them [63, 107, 125]; however, these tools must be told the dataset and data

ranges to analyze or visualize. Visualizing a dataset of observations may confirm that a given dataset does or does not contain the desired data; but individually visualizing each dataset is not practical as the number of datasets increases. Thus, a key part of the scientist's needs is in linking search results to visualization, data download or analysis using appropriate tools [48, 124]. Further, since visualization and analysis tasks are generally nonlinear with respect to the amount of data analyzed [48], it is often advantageous to limit the amount of amount of data analyzed to the most relevant subset within a larger dataset.

If a scientist fails to find an exact match for her interests, or even if she locates one dataset of interest, additional potentially relevant datasets (similar to already-located datasets) are rarely found using current methods, much less are these similar datasets ranked by their relevance or similarity.

Metadata collection, curation and maintenance are themselves acknowledged and ongoing problems. Creating metadata to describe and categorize data is a labor-intensive and oft-neglected part of research projects [27, 93], and relying on scientists to contribute metadata describing their datasets is considered a prescription for failure [14, 81]. Search systems that rely on the success of this process have limited practicality and utility.

## **1.2 An Information Retrieval Approach**

Similarities in the problem description between the scientists' need to identify relevant datasets, and the long history of Information Retrieval (IR) in defining relevance [115,

116] and searching for relevant text documents in large collections [83], sparked our curiosity.

The Internet has seen explosive growth over the past few decades, along with challenges in locating sought-for information as the volume of web pages increased. Tools to address these challenges have been developed and have matured. Initially, web pages were only available to someone who knew the pages' URLs. Then, some users created themed directories of pages (such as the first version of Angie's List or Yahoo!); other users navigated these lists and hierarchies to find the pages relevant to them. Simple search capabilities were then added to ease this task. We now have large-scale search engines that index these directories and the pages they catalog. In response to a user search, the search engine identifies and returns – based on the terms the user has searched for – a list of possibly relevant pages, along with a snippet from each. The user can further examine pages from this list to find the ones suited to his information need. The search engine acts as a filter, presenting a smaller list of pages than the user would otherwise have to browse; it also orders the list by some notion of relevance (and in some cases even multiple alternative estimates of relevance [3, 34, 128]) hoping to present the best pages near the top of the list. These web-search techniques now allow users to easily find relevant documents despite the enormous growth in total pages available on the Internet.

The use of search services such as Google and Microsoft Bing have changed users' expectations of search engines [146]. Users are accustomed to receiving a ranked list of

search results, with “close” matches to their search along with “not so close” matches.

Further, they expect that the search engine will respond based on the contents of items rather than just on externally defined metadata such as document title or specified keywords.

We wondered: Can we move beyond the existing word-matching and simple geometric comparisons used in contemporary data-archive search systems, to estimating the relevance of a dataset to a scientist’s information need? Further, what are the similarities (and differences) between web search and scientists’ search for data? Can we adapt techniques from web search to help scientists find relevant datasets; and if so, which techniques are applicable, and how far can we push the correspondence?

We believe we have demonstrated that we can; this dissertation describes our results from exploring these questions. We explore the feasibility and utility of applying IR techniques to search over an existing archive of scientific datasets, with some first steps towards extending the approach over multiple archives. We begin with the standard high-level

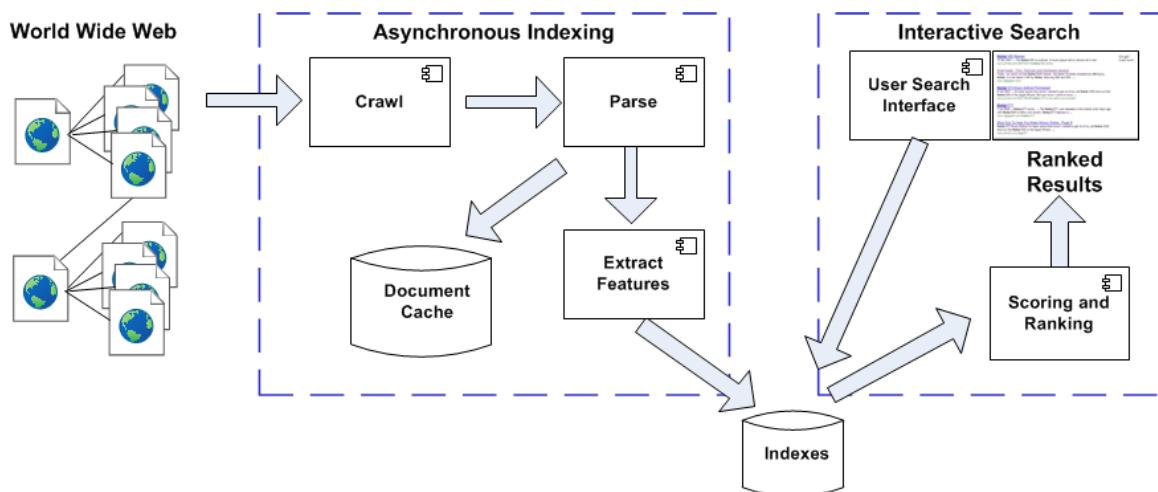


Figure 1.1. High-level web-search architecture

architecture used for web search, as shown in Figure 1.1, and adapt it to our situation.

In devising the details of our approach, we assume that scientists will continue to use tools they are familiar with to visualize or analyze data once they have discovered it. We are biased towards identifying light-weight, easy-to-use approaches that ease the discovery process; as noted in considering the success of Google Maps, “Richness and depth are trumped by speed and ease, just as cheap trumps expensive: not always, but often” [94]. Other search engines have found that providing fast response time is key to their utility [117]; we believe the same is true in searching for scientific data. We are also biased towards exploiting well-studied and optimized underlying functionality and techniques wherever possible.

### 1.3 Contributions

We assert that the IR concept of relevance is applicable to ranked retrieval of scientific datasets, and that therefore IR similarity measures and IR evaluation methods are also applicable. Without such concepts applied to scientific data, the usefulness of a scientific archive decreases as the archive grows beyond the ability of an individual scientist to navigate it.

Specifically, our contributions are:

1. We cast the problem of finding scientific data within an archive or a collection of archives as an Information Retrieval problem, similar in nature to finding relevant text on the web or in a document collection.

2. We present an approach to solving the problem, by adapting and applying Information Retrieval techniques to scientific datasets.
3. We formulate a model for evaluating the similarity of a set of datasets to a scientist’s search. We show that the model can be instantiated by a flexible, componentized software architecture.
4. We show that we can directly map these approaches into a ranked-retrieval system for datasets. We demonstrate the feasibility of the approach, model and architecture by implementing these principles in a prototype (“Data Near Here”) over the majority of CMOP’s holdings, which represent multiple types, scales and formats of data. Our implementation links search results to visualization, data download and analysis using appropriate tools, as desired by our scientists.
5. We propose a candidate similarity measure consistent with cognitive science research, and implement it in the prototype.
6. We provide evidence of system utility via two user studies. In these studies we demonstrate that the concepts of “dataset relevance” and “dataset similarity” are meaningful when applied to scientific data. The first user study examines our candidate similarity measure, while the second user study evaluates our prototype.
7. We demonstrate that IR measures (such as Mean Reciprocal Rank, Rank Biased Precision and Discounted Cumulative Gain) are applicable to dataset search. We compare our candidate similarity measure to several alternatives using these measures;

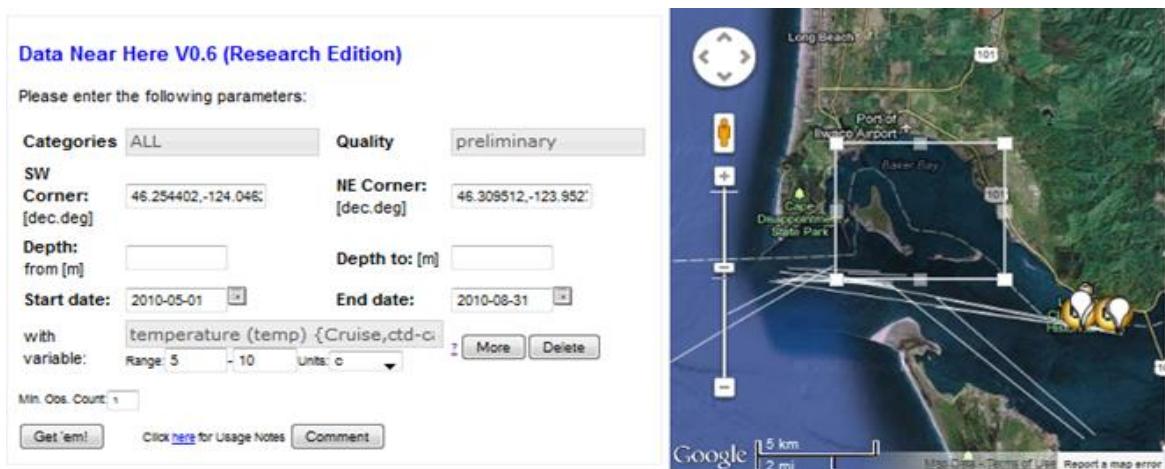
- the IR measures indicate our initial candidate similarity measure performs well in comparison.
8. We address the desire by scientists to easily identify a subset of data that matches their research interests within a larger set of data by virtually segmenting datasets to give the appearance of multiple, smaller datasets within our system. We incorporate this capability into our model and implement it in our prototype.
  9. We describe and evaluate the architecture of our system, and show it will meet currently anticipated needs.

10. We explore the performance of our prototype, using data of interest to our scientists. Looking ahead to continued growth in the data collections, we adapt and assess several performance-improvement techniques to support larger collections.

Given the availability of an observational archive and associated scientific research community at CMOP, we further limit our empirical work to their archive and to their immediate needs. While our experiments are undertaken within one particular scientific research discipline and archive of observational data, the same issues and problems we consider are seen in many other research disciplines [142]. We also believe our work can be extended to cover a confederation of scientific archives.

Our prototype, called “Data Near Here”, has now been deployed for use by our scientists for over six months. Figure 1.2 shows the primary search interface, which includes on a single webpage the search interface and ranked search results. The search interface is akin to the “advanced search” used by some text-search engines, but adapted to scientific

data. (Note that the search interface is naïve, and is not the subject of our research.) The search results are displayed as brief summaries of the datasets, akin to the “snippets” of a document shown in web search. Details of a dataset’s metadata and contents can be viewed by clicking on an individual snippet. Because geospatial location is so important to our user base, the spatial extent of the search is shown on the map interface (the square white box), and the spatial extent of the datasets in the search results list are also shown on the map (here, diagonal lines and markers; other datasets may be represented by



There were 24 results returned; all are listed, and 24 initially shown on map. Temp was found in 24 entries.

Display	Type	Collection	Quality	Start Time	End Time	From Depth	To Depth	temp	Observations	Data Location	Score	DNH	
<input checked="" type="checkbox"/>	1	Cruise	<a href="#">Cruise_May-June 2010_Weoma_2010-07-16_Segment 3</a>	preliminary	2010-07-16 05:16 PDT	2010-07-16 05:29 PDT	-5	-5	9.89:12.14 c	14	<a href="#">Download</a>	98	<a href="#">DNH</a>
<input checked="" type="checkbox"/>	2	Cruise	<a href="#">Cruise_April 2010_Weoma_2010-04-17_Segment 4</a>	preliminary	2010-04-17 04:06 PDT	2010-04-17 04:26 PDT	-5	-5	10.60:10.85 c	21	<a href="#">Download</a>	97	<a href="#">DNH</a>
<input checked="" type="checkbox"/>	3	Cruise	<a href="#">Cruise_April 2010_Weoma_2010-04-17_Segment 11</a>	preliminary	2010-04-17 18:52 PDT	2010-04-17 23:59 PDT	-5	-5	10.88:11.21 c	244	<a href="#">Download</a>	96	<a href="#">DNH</a>
<input type="checkbox"/>	4	Cruise	<a href="#">Cruise_April 2010_Weoma_2010-04-18_Segment 1</a>	preliminary	2010-04-18 00:00 PDT	2010-04-18 01:15 PDT	-5	-5	10.88:11.07 c	77	<a href="#">Download</a>	96	<a href="#">DNH</a>

Figure 1.2. User interface for “Data Near Here”, showing a sample search for a geographic region (shown as a rectangle on the map) and date range, with temperature data in the range 5:10C. Result datasets (or subsets) are shown as points and lines in the output pane, together with their relationship to the search region. In the ranked list of answers, no full matches for the search conditions were found; four partial matches to a search with time, space and a variable with limits are listed, and more are shown on the map.

polylines or polygons). However, our work is not limited to this kind of geospatial-temporal search.

We expand on the background on the problem and other approaches in Chapter 2; that chapter includes an overview of our solution and describes related work. We describe our underlying model for dataset similarity and search in Chapter 3.

Chapter 4 provides more tangible detail on our approach to dataset similarity; we also describe our approach to extracting metadata from our test archive of datasets. We use these approaches in our prototype. In Chapter 5 we describe the prototype, giving details of the implementation and an architectural evaluation of the implementation.

In Chapter 6 we describe our two user studies. Our first user study tests the feasibility of ranked search of scientific datasets via a relatively straightforward similarity measure we developed. This first study focuses on geospatial and temporal characteristics of observational datasets, two features that are critical in many areas of scientific research. Our second user study takes the form of an operational test by users of the search tool against around 30,000 datasets totaling more than 0.5 TB, the observational component of the repository. This data represents over a decade of environmental observations. Thus, the task we are studying in this work is both topical and real. The searchers are scientists using the repository, who formulated searches representing their own information needs. This study structure gave us a tight linkage between real users with their own information needs, and the assigned relevance ratings or “ground truth” for their search results.

We then consider another challenge: is it possible to provide interactive response times, as we have come to expect from document search, over a metadata collection representing a scientific data archive of current and expected sizes? To understand the performance and scalability characteristics of the concepts beyond the catalog sizes afforded by the current prototype, we estimate the potential effect of growth of the archive on the number of dataset summaries. We developed techniques to improve response time; we describe these techniques and assess the improvement they afford in Chapter 7.

We recognize that there are significant additional research challenges to generalizing this initial work. We describe a subset of these challenges in Chapter 8.

## 1.4 Terminology

While our work is currently focused on scientific data, we recognize that non-scientists may also have interest in the data; also, the principles we develop may be useful for data that is not directly scientific in nature. We therefore sometimes use the more general term *user* to refer to a person who uses our prototype or systems built using our model, in addition to using the term *scientist*.

We use the term (*scientific*) *archive* to mean an online-accessible collection of scientific data. The archive is generally under the administrative control of a single institution or agency, and preserves and provides access to data pertinent to the agency's mission. For example, CMOP has an archive of oceanographic and coastal-margin data, while

NOAA’s National Climatic Data Center has an online data archive of weather and climate data.<sup>1</sup>

Most scientific archives have a contact person responsible for responding to questions about the data, ensuring that data can be downloaded, creating and managing metadata to describe the data, etc. We refer to this person as the archive’s *curator*.

We use the term *dataset* to mean a defined set of (scientific) data with some internal structure that can be consistently determined via some access method. The set of data contains *variables* (like environmental variables such as “oxygen saturation” and “nitrate”), and those variables have one or more *values*. This definition of dataset includes such variants as a file stored in a file system containing tabular data in a series of columns, a file in a standard scientific format such as NetCDF, or a set of rows in a database table. Although we have experimented most with these forms and use them in our examples, we do not restrict our definition to these dataset types. Our ultimate goal is to match the concepts of dataset, variables and values within the minds of the scientists; we believe that groups of scientists within a single research field often have similar definitions, and thus these definitions should be recognizable and replicable.

We borrow the term *catalog* from the field of library science to describe the collection of entries we create that summarize and allow us to search over, locate and access scientific datasets. Our catalog contains data about the source data, that is, *metadata*.

---

<sup>1</sup> <http://www.ncdc.noaa.gov/cdo-web/>

To (we hope) reduce confusion, we use the term *index* in its Information Technology usage, to refer to a list of keys used to quickly access data within a data structure. We recognize that our catalog is itself a form of index, and can be implemented in a relational database in a set of tables that themselves have indexes.

Our metadata does not retain the association of specific values with the source observation; that is, within any set of tabular data we operate on columns of data, not on rows. For simplicity, we may refer to a column name as being an *attribute* or a *field* of the dataset. We may refer to the set of values contained in a column as its *footprint*, and the minimum and maximum values as its *bounds* or *range*. We sometimes refer to a dataset's collection of bounds across all columns as the dataset's footprint.

We use the term *dataset summary* to refer to the brief description we use to represent the dataset in our metadata catalog. Generically we regard it as consisting of a unique identifier for the summary, an identifier of the dataset to which it refers, and some collection of features. We store our data summaries in a catalog; thus, our data summaries are metadata.

Adapting a definition from Information Retrieval (IR) [83], a dataset is *relevant* if the scientist perceives that it contains data that satisfies her information need.

We refer to the user's representation of their information need, as presented to a search system, as his or her *search*. We consider the search as consisting of a number of conditions that describe the overall information need, and we refer to each of these

conditions as a *search term*. When a search term specifies a variable with a higher and lower value in some units, we call this interval the desired variable's *range*.

We use *query* to mean a single request to a relational database system. Thus, a single search may be implemented via a series of queries against a database.

## **2 Background and Overview**

How do scientists locate data today, and why are these methods not adequate? In this chapter, we first discuss our motivation in Section 2.1, then some similarities to and differences from the field of Information Retrieval in Section 2.2. In Section 2.3 we describe a fairly typical scientific archive we work with and use as our test bed, and in Section 2.4, current methods used in searching for data in similar archives. We identify what we believe are some common characteristics of scientific archives in Section 2.5; for our work to be generally applicable across archives, our approach must be robust with respect to these characteristics.

We then provide an overview to our work in adapting web Information Retrieval (IR) techniques to this field (Section 2.6) and summarize related work (Section 2.7).

### **2.1 Motivation**

The practice of observational science has changed dramatically in the past few decades. Scientists now research ecosystem-scale and global problems in interdisciplinary teams – driving the need for more data over more environmental variables from more locations over longer timeframes [5]. Time expended in searching for relevant data is now a significant overhead on scientist productivity.

In our work with one scientific archive, we see the effect on scientists of the dramatic growth in available data. Microbiologists, who once collected a few water samples a year and studied them intensively, now have equipment that can capture a water sample every few seconds; on science cruises, sensors measure environmental variables every few

milliseconds over hundreds of miles. The methods available for managing and exploiting the information now being collected have not kept pace. Anecdotal evidence indicates that even when a scientist has previously worked with a dataset, he or she may “misremember” exactly what time or place it was obtained. Given current access methods, such a dataset is effectively “lost”, as far as that scientist is concerned [92]. Informally, scientists have told us that they have abandoned research questions because they found it too hard to locate relevant data, even in cases where they knew the data they sought existed. This behavior is not limited to our archive or scientific field [124]; in the field of chemistry, Caruthers states “we [scientists] are starting to die from data,” and that scientists desperately need better ways to store and retrieve research data or “we are going to be more and more inefficient in the science that we do in the future.” He adds “data from experiments conducted as recently as six months ago might be suddenly deemed important, but researchers might never find those numbers” [22]. The problems scientists experience in locating relevant data threaten to undermine the value of large and growing archives.

Our research goal is to counteract these problems by making it easy for scientists to find data relevant to their research questions, despite growth in the archives that store that data.

## **2.2 Information Retrieval Concepts and Scientific Data Search**

There are strong similarities between the scientists’ search for data and the needs addressed by the field of Information Retrieval. In both cases, a user represents his

information need by a set of search criteria; ideally, he desires an exact match to his information need. In its absence (due to lack of data or to an imprecise formulation of his information need), he may consider a “near match” instead; research by D’Ulizia et al. shows that 95% of users would rather have an approximate answer or a near match than none at all [28].

The traditional text IR model focuses on three components: a document, a search, and a similarity measure that compares them. Traditional text IR treats a document as a bag of words, with each distinct word within the document regarded as a *feature*; further, a frequently used word (e.g., “the”) is seen as having less value than a less frequently used word (e.g., “deconstructionism”). In many scientific datasets, each variable name is listed once as a column heading and not repeated, while all the values are listed in a column below; units may be supplied in the heading or as metadata. Treating a dataset as a document and applying a simple bag-of-words model implies matching based on word equality. However, when treated this way, the values in a column would be disconnected from the associated variable and each (white-space separated) value treated as a word; the association between a variable, its units and values would be lost. Finding a variable “water\_temperature” and the value “5.0” somewhere in the same dataset is not the same as finding “water\_temperature of 5.0”. Similarly, a text IR search also consists of a bag of words (for example, “Paris Hilton”), and thus each search term can be matched to a document feature. Our scientists, however, do not search for specific values found in a dataset (“water\_temperature 5.93615C”), but rather express their information needs in

terms of an observational variable with values in some range (“water\_temperature between 5 and 10C”). In text IR, a similarity measure is applied to the search and each candidate document, and documents “more similar” than some cut-off value are returned. Common text similarity measures (e.g., cosine similarity using tf-idf) balance how frequently words in the search occur in a document with how frequently they occur in the document archive. For scientific data, it is not clear that a frequently occurring value makes a dataset containing it less relevant to a search. Thus, the bag-of-words similarity measure is also not a direct fit.

Thus, the bag-of-words model and the similarity measures that rely on that model, as used in traditional text IR, do not seem to directly apply. Even when we have solved the issues of accessing and reading relevant datasets (see Section 2.4), traditional IR methods must be adapted to be usable for scientific data. Nevertheless, we are curious to see how much we can learn and apply from existing IR techniques, how the techniques can be adapted, and how far applying these approaches can take us towards our goal.

Given the size of this research topic, our approach was to work with a specific archive, described below, to experiment with and test a combination of methods, and to use the results to inform further experimentation and research.

### 2.3 Our Study Archive

The Center for Coastal Margin Observation and Prediction (CMOP) is a National Science Foundation Science and Technology Center (NSF STC) based in Portland, Oregon, focused on coastal-margin and near-ocean issues. It is a multi-institution, cross-

disciplinary research partnership consisting primarily of oceanographers and marine biologists. CMOP has collected observational data from an ever-changing set of fixed and mobile sensors around the Columbia River and off the Washington and Oregon coasts for more than a decade – a rich resource for oceanographic research.

CMOP collects observations using a wide variety of fixed and mobile observation platforms; individual observations may even be manually collected, for example a water sample may be gathered by a biologist with a bucket. At another extreme is a network of fixed stations, each of which have a single geospatial location (often with multiple elevations), and may have collected a million observations spanning a decade. The mobile sensors may collect millions of observations over widely varying geographic and temporal scales: science cruises may cover hundreds of miles in the ocean over several weeks, while gliders and autonomic unmanned vehicles (AUVs) are often deployed for shorter time periods – hours or days – and a few miles, often in a river or estuary.

A single observation consists of a set of measurements of *environmental* or *biological variables* at a point in space at a point in time. Observations that are related in some way (often, collected from the same source, at a similar time, or supporting a single research project) are stored together in one or more datasets. The set of environmental or biological variables (hereafter called simply *variables*) observed changes over time; there are frequent changes in the instruments deployed as new instruments are developed and new research topics studied, leading to changes in the data structures used to store the data.

CMOP’s observational archive spans more than a decade and contains thousands of datasets, with over 0.5TB of data in aggregate. CMOP scientists analyze historical observations and run complex simulation models, producing additional terabytes of (computer-generated) observations [82]. Almost all data (both observed and modeled) is accessible for public download via CMOP’s portal or from their THREDDS server.<sup>2</sup> The datasets are heterogeneous in content, format and storage type. Multiple tools are needed to access and read the different dataset types, and no single interactive query or search capability spans all the data. Figure 2.1 gives a simplified schematic of the range of data formats and data access tools currently used at CMOP for searching for data; all of these tools are variously used by scientists to locate and identify the relevant subsets of data in the archive for their analysis. Some of these tools are also used for analysis once the relevant data has been located.

The CMOP repository is in many ways a typical scientific archive. An informal examination of other sister archives accessed by CMOP scientists showed only minor differences in the capabilities and tools provided.

Scientists want to search these collections of observations for data that matches their research criteria. The scientists at CMOP often define their information needs using varying combinations of geospatial areas, temporal ranges, environmental variables collected, and ranges of readings for specific variables. For example, one microbiologist may be looking for “any temperature readings near the Astoria Bridge in August 2011” in

---

<sup>2</sup> Data can be accessed via CMOP’s website, <http://www.stccmop.org>. Some data is not available to the public until quality assurance is completed.

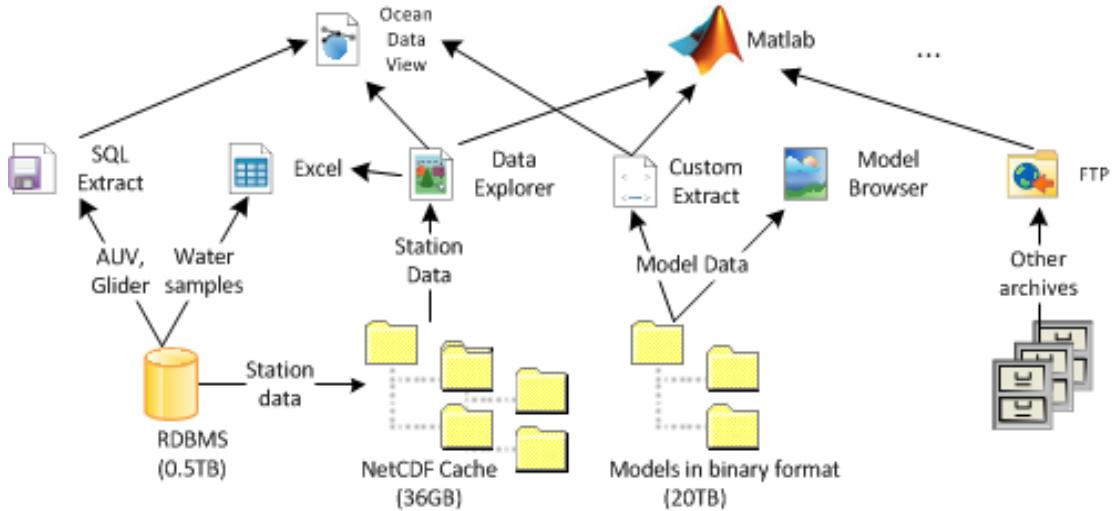


Figure 2.1. Heterogeneity of data formats and data access tools in one scientific archive

order to place a water sample taken there into physical context. An oceanographer may be looking for simultaneous low oxygen and low pH (high acidity) in a river estuary, as it may indicate that upwelling ocean water is entering the river system. He is interested in data from any time period with these conditions.

## 2.4 Current Approaches to Searching for Scientific Data

How do scientists describe their information needs, in order to search for relevant data? To our knowledge, this question is not discussed in the research literature; their search language is generally implicit in or mandated by the system or approach used. Approaches currently described in the literature fall into three main categories:

- Data access via successive menu selections or navigating through a directory hierarchy;
- Visualization of a set of data;
- Text search for words in metadata associated with the datasets.

In this section we describe the use of these approaches in searching for scientific data. Then, in Section 2.5, we take a step back and discuss why the often-suggested alternative solution of portals and gateways (which act as the front-end for one or more of these approaches for some collection of datasets) does not adequately address the scientists' needs.

CMOP scientists have available to them a state-of-the-art set of tools for locating observational data in their archive [82]. The interface presents them with a set of options at each step that narrows down to the (presumably) desired dataset. A scientist can select a category of data (station, cruise, etc.); then select the desired station or cruise from a list; select an instrument; then select the desired time period containing the relevant data, and lastly, the specific dataset can be accessed.

However, the existence of this system, akin to structured data retrieval, does not fulfill many of the information retrieval needs of the scientists. Scientists are now asking questions such as: for a particular location or time, what variables were collected, anywhere in the archive, or even, in other similar archives? For a particular variable, during what time periods and at what locations are there observations? What data exists in the archive where a specific variable has values within some identified range? In each case, the structured-access interface does not afford a rapid answer; the scientist must scan and access a significant portion of the entire archive to answer the question. The access is complicated by the heterogeneity of the dataset locations and formats. While this heterogeneity is partly hidden by the structured-access interface, this kind of search is

not practical with the selection-driven approach the interface uses.

One of the most common methods proposed to help scientists find relevant data is to graph or otherwise visualize a large quantity of data, trusting that the scientists can visually identify the relevant subset of data out of the large volume of data displayed [63, 107, 125]. The scientist specifies the dataset and range of data within the dataset (that she has identified using some other unspecified method). The system then presents a visualization of the specified numeric data. In some cases, a million data points can be represented in a single image, to allow the scientist to visually assess patterns or sought-for irregularities. This method is implicitly assumed by THREDDS, for example. However, these tools must be told the location and name of the specific dataset and the data ranges to analyze or visualize.

Visualizing each dataset individually, looking for possibly relevant data, also has built-in scalability constraints; in all cases, visualization on a per-dataset basis is only practical for a small number of datasets. Visualizing a dataset of observations for a desired location in, say, June may find no relevant data. Potentially relevant substitutes that are “close” in either time or space (say, from late May in the desired place, or from June but a little further away) are not found using current methods. While visualization allows the scientist to review more data at a time than, for example, looking at the same data in the form of a table of numbers, the approach is still limited by the amount of time the scientist is willing to spend in this dataset-by-dataset review; visualization tasks scale almost linearly with data volumes [49]. In addition, the relevant visualization tools

generally change for each data-storage method, so the scientist spends more time dealing with software install and license issues and learning curves for additional tools – a further disincentive to searching for relevant data.

Using the approaches described above, locating a relevant dataset of observations requires that the scientist knows that the relevant data exists, and understands the dataset’s storage location, access methods, tools and format (e.g., NetCDF versus relational data versus unstructured models). Even with a search tool that can find data in a desired range, the scientist may not know how far to set those bounds in order to encompass possible substitutes. If the scientist knows a dataset exists but does not recall the details correctly, then the dataset is effectively “lost.” He is not personally involved in all collection activities and so is not personally familiar with what data was collected, and memories fade even for those he is involved in [93]. If he is not aware of exactly what data exists and which tool to use to access it, or he misremembers the exact combination of search terms or selections to find the desired data, he may find no data. The successful use of any search or query interface that returns a Boolean result (a result is either in or not in the desired set) is dependent on the user specifying exactly the search terms that will find the desired data. Widening the criteria may result in more datasets than he can review or analyze, with no guidance on which are likely to be most relevant.

Another common approach is for the archive curator to add metadata or annotations to the datasets in the form of a catalog, and expose the metadata via a search engine. Some catalogs [43, 103, 109, 129] provide text-search capabilities over the contents of metadata

fields; however, scientists are often vitally interested in the data values contained within a dataset rather than the text description. Often, descriptive metadata may only lead a scientist to conclude that there might be relevant data in a specific collection of datasets; finding the subset of that collection that is relevant requires access to the contents of the datasets themselves. Even when the dataset contents are exposed to a search engine, most text retrieval approaches treat numbers the same way they treat other words; a search for 6798.320 on Google does not return 6798.32 (although it does return the paper that makes this point [4]).

The effort involved in creating metadata over which to search is an acknowledged issue in the literature. Further, having metadata does not help unless the provided metadata matches the kinds of search terms scientists wish to use to find the data. Automatic metadata generation has been identified as an important need, since the manual metadata annotation by scientists required in most systems is considered burdensome and is rarely provided [14, 60, 81]. One group noted that the users wanted more metadata than providers were interested in providing, and that providers stopped providing access to data when more metadata was requested from them [27].

Where only a subset of a dataset is relevant, the search challenge is even greater. Within a dataset with millions of rows, only a few thousand may be relevant; finding the subset containing those few thousand is difficult using any of the approaches described above. Extracting the relevant portion of a dataset, once located, can then be a separate challenge, and the time taken here is a further constraint on productivity.

Our scientists express frustration at the database-style or sequential-selection searches and visualizations that they currently use to locate desired data. Even with the techniques and tools described in this section, their problem remains.

None of the approaches described here adequately address the question of how the scientist can efficiently identify interesting datasets and ranges to visualize. That question is the subject of this research.

## **2.5 Archives, Portals and Gateways**

Other observatory archives we explored have many of the same characteristics as CMOP's archive; these similarities make our work broadly applicable to such archives. In this section we describe the common characteristics that are relevant to our work.

### **2.5.1 Data Access and Sharing**

Scientists now often wish to expand coverage of their analysis or simulation models, spatially, temporally, and in terms of the variables studied. This expansion increases their need for relevant data, and increases the likelihood that they will need to search across multiple archives. At CMOP, scientists are including more of the Pacific coast and ocean in their simulations, and are beginning to simulate microbiological populations. Thus, data relevant to our scientists is being collected by and stored in multiple other observatories and archives, each with its own access methods. The providers of observatory archives often desire, or even have a mandate, to make their data available to researchers both within and outside their organization, leading to issues of data access and sharing.

In the typical data archive there are many categories of data that a scientist must review to find relevant data. Each archive generally has some internal structure based on some logic meaningful during data capture and storage: for observational archives it may be the observation source, the instrument capturing the observation, the time of capture, intended or original usage, owner, or some other construct. The data-storage locations and formats themselves change over time, as the archive evolves. This structure may not, however, have direct applicability to the scientist searching for data. For each scientist to gain familiarity with each of these other archives' holdings is not realistic; however, failure to include relevant data from other archives in a scientist's research may artificially (and inappropriately) limit that research.

In the oceanographic community, efforts to make data sharing easier date back to at least the early 1990s [27], but the challenge is not limited to that community. Barros et al. [13]

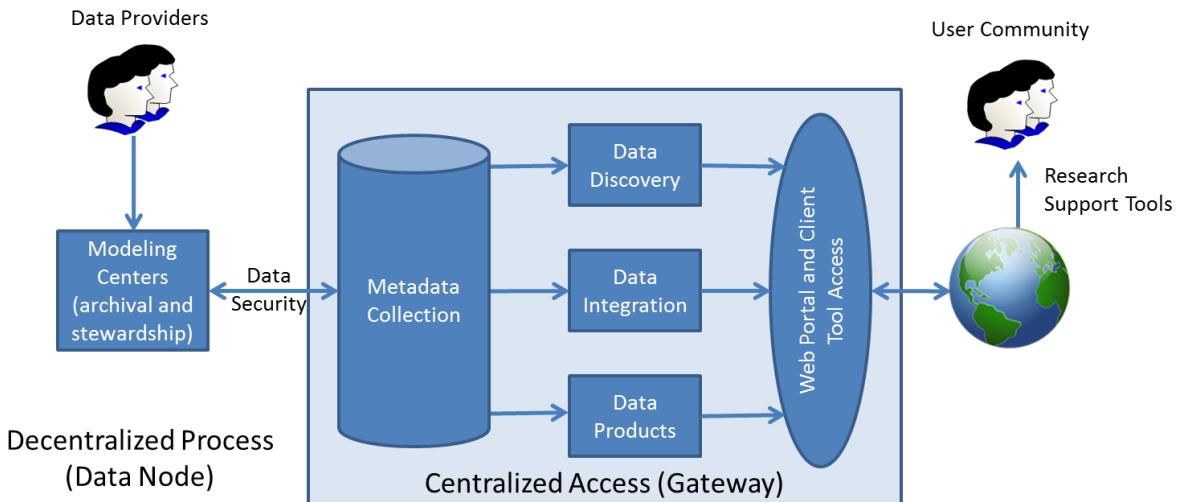


Figure 2.2. A centralized gateway architecture in the climate-change community allows users to discover, integrate and download data from a variety of data providers via a centralized gateway. After Ahrens et al. [3]

describe data sharing via a digital-library approach for uploading, storing and browsing spreadsheets of ecological observations. Ahrens et al. [5] describe the efforts by the climate-change community to share observations and modeled data via a centralized gateway. As shown in Figure 2.2 (after their paper), the gateway consists of: a metadata collection, contributed by data providers; modules for data discovery, data integration and access of data products; and a web portal with client-tool access capabilities, which exposes the data to the user community. The authors use a climate-change case study to draw attention to the growth of data and the challenges in making the data useful to researchers, and the similarity of these issues to other scientific domains. A similar challenge and conceptually similar solution for ecology is presented by Reichman et al. [110] and separately by Baker and Chandler [9]. These portals rely on data owners submitting their data to the portal together with the metadata the portal suggests or requires. Data discovery is addressed by using one of the approaches described previously.

Data access could be simplified by migrating all data to a single format, with NetCDF [111] being a common choice. However, data formats are generally selected by the producer rather than the consumer, who are often not in the same organization. Even when the producers and consumers are in the same organization, it is often not practical to migrate many terabytes of data or the programs and tools that access them from current formats to a new format, and data must often be kept in original format for use by existing processes. Such a migration would reduce the number of tools required, but

would not change the problem of having to locate and scan all the possibly relevant datasets.

One part of the solution is to develop middleware and catalog software such as THREDDS [33]. The goal of THREDDS is to provide an end-to-end system for data access and visualization, based on a loose federation of distributed metadata catalogs and inventories; it is assumed that the user can identify and locate interesting data via separate means. The catalogs, implemented as XML documents served over HTTP, are accessed by interactive data analysis and display tools, of which many have been built. Once relevant data has been identified, Dataset Query Capabilities (DQC) allow users to request a subset of a dataset collection for download. THREDDS can be coupled with a data-transport-and-access protocol such as OPeNDAP [27], which allows an identified dataset or specified subset to be downloaded via standard or self-coded browser plugins for specific data types. THREDDS has focused on the problem of providing middleware that allows clients to access catalogs and download data in formats recognizable to THREDDS plugins, and not on the problem of locating the datasets of interest. (These technologies are all in use at CMOP.) Our work is complementary: we help the user locate and identify datasets of interest that THREDDS gives them access to.

Should the scientist's information needs expand past the geographic region covered by her home observatory, there is much in these efforts that can help her access and analyze datasets – once she has found relevant ones. However, knowing whether relevant data resides in a specific archive is currently dependent on knowing which observatories

operate in which areas, and how to locate, access and navigate the relevant portal or gateway. Once she has found the relevant portal, she still needs to locate the data within that portal, and she is constrained by the same tools and restrictions discussed previously. Her problem of finding relevant data is repeated, at a higher level.

### 2.5.2 Common Characteristics of Scientific Archives

The characteristics common across the archives relevant to our scientists – including that of our primary subject, the archive at CMOP – are heterogeneity, size, variation over time, and read-mostly.

**Heterogeneity:** Scientific data is stored in heterogeneous formats, and the formats selected may change over time. Common formats include scientific standards such as NetCDF (netCDF-4/HDF5, 64-bit Offset Format, classic), binary files created by a variety of custom programs, delimited or positional text files – sometimes delivered as HTML pages – and relational databases. The internal structure or representation may change; for example the columns on an HTML page representing an instrument's readings may change when the instrument is upgraded. Variable names are not consistent. Instruments are added or moved, new variables are measured, with each change adding to the existing variations. To review each variation and understand how to process it to check for relevance requires multiple tools and much time.

**Size:** Archives are now routinely terabytes in size and may contain thousands of datasets of widely varying sizes, and the rate of growth continues to accelerate [81, 142]. An individual dataset may contain millions of records. In other cases, a dataset may consist

of a small number of individual but important observations; for example, in some years CMOP only collects tens of water samples. A single water sample may have hundreds of environmental variables associated with it, including DNA samples. Focusing only on large datasets may mean that small datasets get less attention or are treated in an ad-hoc manner, making them harder to find.

**Variation Over Time:** Many observational archives accumulate data over decades. While a single observation site may exist for that whole period, the specific data collected often changes over time as new equipment or methods become available, or as sensors are added or removed in support of specific research initiatives. Formats, naming standards, measurement units and scales of measurement all change.

**Read-Mostly:** In general, existing data remains in the archive, and is added to over time. Once created, data is not updated except in unusual situations, for example, if the dataset is discovered to have been incorrectly produced, or new quality-control or calibration procedures were applied.

As noted in the introduction, our experimental work focuses primarily on a single archive (CMOP). However, we consider these aspects of other archives to ensure that our work can be generalized and is focused on problems experienced widely across the scientific-data community. We believe these characteristics are common in archives in fields unrelated to physical observations or oceanography.

## 2.6 Adapting Web Information Retrieval Approaches

In this section, we describe at a high level how we adapt web-based IR approaches to searching for scientific data. We see our research as providing a realization for the “data discovery” component of Figure 2.2. Further, we suggest a content model for the metadata catalog, and use that metadata in our data discovery approach. Our work assumes that the source data can be accessed via some (unspecified) tool; in particular, we access the source data to automatically create (much of) the needed metadata over which we propose to search.

In IR systems, the user converts her information need to a set of search terms, usually a list of words, to be searched for in an index representing a library of items (where an individual item may be, for example, a web page). As shown in Figure 1.1, web-based IR approaches separate off-line indexing of a collection of web resources (HTML pages and other documents) from interactive, on-line search. The indexing process is performed asynchronously as a *feature-extraction* task: each web page is scanned and relevant features extracted. A feature may be the count of a word in the page, or it may be an image filename, a title string, or a link found in the page. The features associated with each web page are stored in an inverted index. During interactive search the user’s search terms are compared to an index containing each web page’s features. A *similarity measure* is applied to quantify the gap between the search terms and the index entries and assign a *score* to each item. For example, the similarity measure may be the cosine angle between a vector representing the search terms and a vector representing a document’s

features. Scores are interpreted as a measure of similarity to the search, and hence estimate relevance of the web page to the search. The highest-scoring web pages are returned in a ranked list [83].

We may regard relevance as Boolean (a returned dataset is either relevant or not), or we may posit a spectrum of relevance (a returned dataset may be somewhere between “highly relevant” and “not at all relevant”). We may therefore roughly differentiate between Boolean retrieval and ranked retrieval. In Boolean retrieval, only exact matches are returned. In *ranked retrieval*, each item is given a score representing an estimate of the item's relevance to the search. The list of items is *ranked* by ordering items from highest to lowest score, and the highest-scoring items returned. In our work, we assume a spectrum of relevance (often represented by several points, or levels, on that spectrum); that is, one dataset may be “more relevant” than another, even if both are relevant.

We would like to present our research scientist with a ranked list of the most-relevant datasets, ordered by decreasing estimated relevance to a search she poses; to do so, we need to quickly compute the relevance scores of many datasets against her search. Ideally, we would like to know how well the contents of each dataset matches the search terms. However, comparing the contents of each dataset to a search directly does not scale as the amount of data in an archive increases. It is not practical to scan a large scientific archive of datasets and their contents while still providing interactive response to a search.

Therefore, in common with most IR systems [113], we wish to identify a set of features that we can extract from each dataset, and upon which we can operate quickly rather than operating upon the original dataset. This set of features will constitute a small summary of each dataset. We want a method for searching over those features, scoring them with respect to their closeness (however defined) to a scientist's search, and providing a ranked list of datasets in response to that search. Each dataset's rank should be based on a relevance score that represents an estimate of the dataset content's relevance to the scientist's search terms.

These needs can be matched to our adapted IR architecture, shown in Figure 2.3. As with web-search architectures, we perform asynchronous feature extraction from the datasets, via scanners that locate, access and process each dataset in its source location. As each dataset is processed, its features are added to a metadata catalog. We provide an interactive search (data-discovery) interface in which to express the search conditions, and a scoring-and-ranking component that returns a ranked list of datasets with the

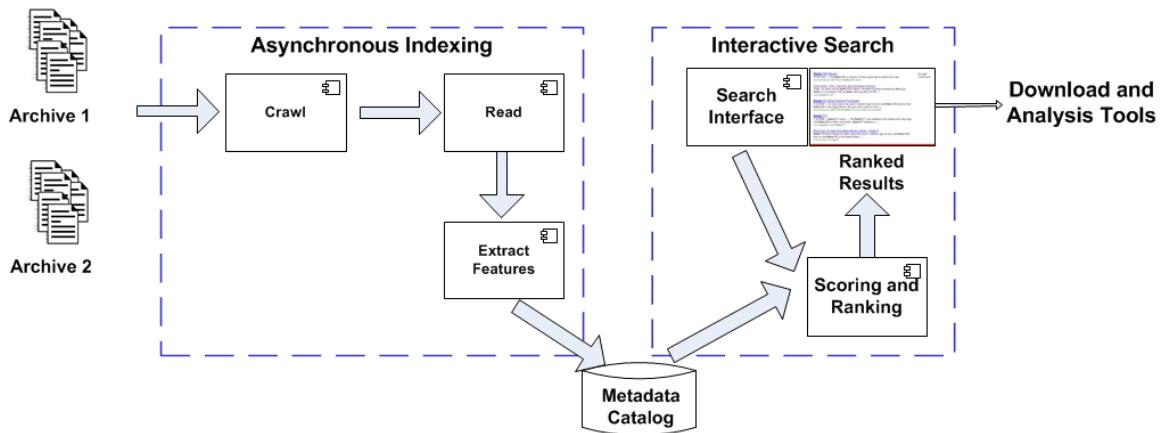


Figure 2.3. High-level dataset search architecture

highest relevance scores. For each dataset, the relevance score is calculated by applying a similarity measure to the search conditions and the features previously extracted.

To make this scheme practical, we must be able to estimate the relevance from a compact dataset summary made up of a set of features. Figure 2.4 shows the high-level concept of what we aim to accomplish: we aim to approximate an “ideal” similarity scoring function over datasets (Figure 2.4a) by finding a light-weight similarity scoring function that can operate over a dataset summary created via feature extraction (Figure 2.4b).

If we can successfully realize this concept, then the application of IR evaluation metrics, such as mean average precision, to the results should also be valid. Further, we must validate that any proposed set of features and similarity measure resonates with potential

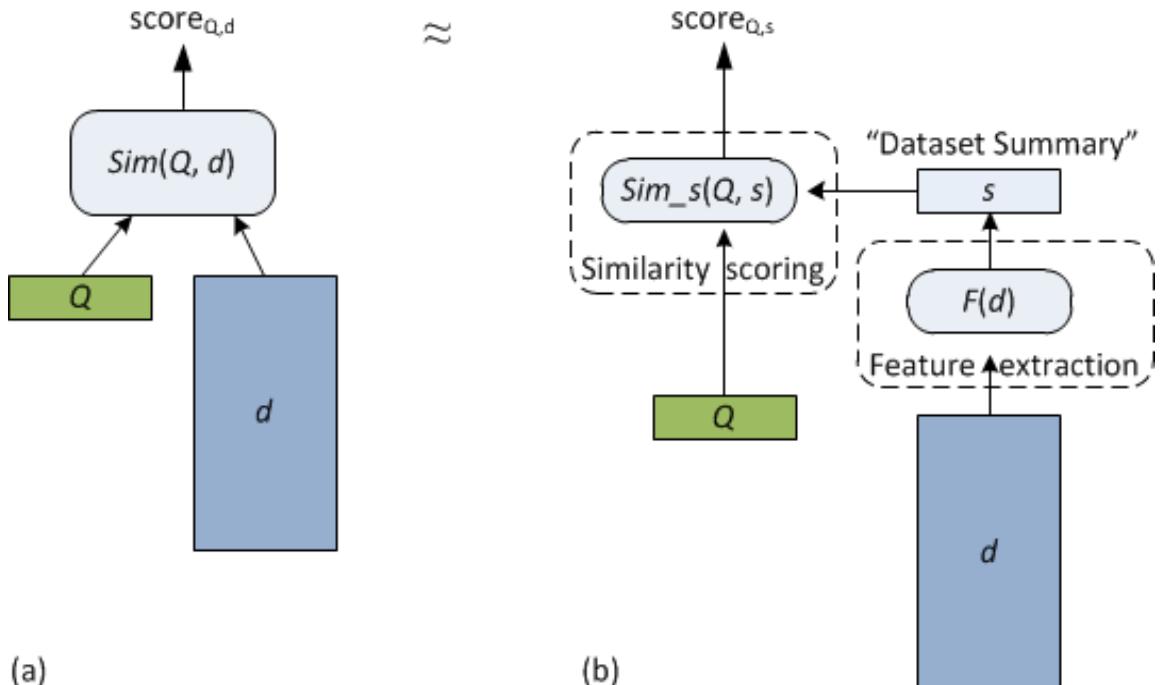


Figure 2.4. Ideal versus approximated dataset similarity scoring. (a) Ideal scoring (b) Similarity score approximated over extracted features.

searchers; and that the similarity measure embodies a notion of relevance that resonates with potential users.

We developed a candidate similarity measure and set of features and instantiated them in a prototype. We tested their utility via two user studies; we describe these user studies in Chapter 6. We show that the search system has utility, and that the similarity measure embodies a notion of relevance that mimics the judgment of potential users.

### 2.6.1 A Notion of Dataset Similarity

We believe that a notion of *dataset similarity* exists in the minds of the scientists. A scientist can generally describe the kind of data he or she is looking for in a quantitative way. As noted in our initial scenarios in Section 2.3, one microbiologist may be looking for “any temperature readings near the Astoria Bridge in August 2011”, while an oceanographer may be looking for simultaneous low oxygen and low pH (high acidity) in the river estuary. Our two scientists use similar descriptions for existing datasets they currently work with. Note that each scientist is, in essence, providing a (partial) summary description of the dataset he or she would ideally like to find but does not give every detail about the dataset’s contents; nor does he or she enumerate the individual observations in the dataset. Further, a scientist can tell us if an individual dataset meets her information need or not; and further, whether it is an exact match, a “close” match, or “not close at all.” We also observe that she often describes the match separately for each part of her information need: she may say that a dataset “is in the right area and has temperature values, but it’s not close to the time I want,” or that “The oxygen values

aren't in the range I'm looking for." These assessments provide a hint on estimating the similarity of a dataset to a search.

Anecdotally, we can describe a dataset as "close to" or "far from" the search, whether we are talking temporally, geospatially, referring to a variable range, or a combination of all three simultaneously. We therefore posit that we can approximate similarity between a search and a dataset by a notion of distance. While it is well known that people are inaccurate in their estimates of absolute distance, research shows that they are relatively consistent in ordinal rankings [100, 112]. Thus, if a distance measure provides overall ordinal rankings similar to those a user would give, it should suffice, even if there is disagreement between the measure and the user on the actual distance. We describe our notion of similarity and our candidate similarity measure in more detail in Chapter 4.

### 2.6.2 Hierarchies of Scale

One of the issues for scientists in finding relevant data is the mismatch between the scales of the data they seek, the scales of observation, and the partitioning of data for convenient processing and storage. Multiple scientists might use the same datasets, but have very different scales of data of interest. We motivate this idea via three examples:

- What is a "meaningful unit of data" for one scientist may not be for another. Lynda could be looking for data for a fairly short time period, since a different time in the tidal cycle is likely to change her results. Figure 2.5 shows a dataset containing several million environmental observations during a specific 2-month science cruise; the overall dataset is split into smaller segments representing specific sections of the

cruise. In Figure 2.5, the most interesting portion of data for our microbiologist, is the cruise segment from July 28, 10-12 a.m., since it is closest in time and space to her search (near the water sample marked “w” in the figure, at that time). Even though another part of the cruise track intersects her water sample, it is not close enough in time to be very relevant. Our oceanographer, in contrast, is looking for simultaneous low oxygen and lower pH; for him, the most relevant data is for the whole day of August 1 – a much larger portion of the dataset.

- What is stored in a single dataset within an archive is generally defined during data collection and refined by data management concerns. However, there is no immediate

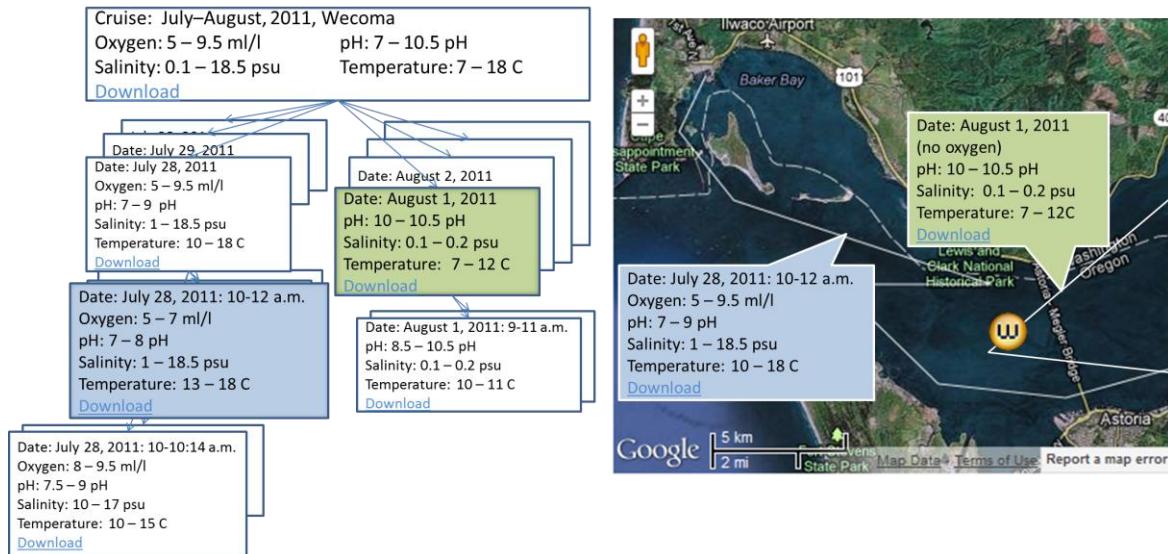


Figure 2.5. Example of a dataset hierarchy: a dataset containing several million environmental observations (taken at 3-millisecond intervals) during a specific 2-month science cruise, segmented into a hierarchy. The white line on the map shows the cruise track, and the marker “w” shows the location of Lynda’s water sample. The most detailed level is a single simplified segment or leg of a cruise, often covering part of an hour or a few hours; these segments are aggregated into successively longer and more complex cruise segments, and lastly into an entire cruise dataset. The most relevant portion to Lynda is shown shaded on the left in the hierarchy, while the most relevant portion to Joel is shown shaded on the right.

relationship between this definition and the needs of the scientists searching for data.

A research cruise may last for several weeks and cover hundreds of miles, collecting millions of observations, and these may be stored in a single dataset. A search performed only at the level of a summary of the entire cruise may cause a highly relevant subset of data to be thought not relevant on the basis of low similarity between the search and the dataset as a whole.

- Consider the difference in scales between a single water sample, consisting of a single observation with a large collection of environmental variables from many tests run on the sample, as compared to a fixed observing station, with millions of observations each consisting of a few environmental variables. For storage management and convenience, archives with both types of data will often store many water samples together in a single file, although the scientists logically regard each sample as separate; again, the overall similarity of the dataset to a single search may be low, masking the high relevance of a portion of its contents.

We wish to mediate within our system between the convenient storage grouping from the perspective of the archive manager and the logically meaningful dataset for each scientist. Further, we want to approximate a concept of the *most useful meaning-bearing unit* [122] for multiple constituencies of scientists, with each constituency having a different granularity in mind. For example, microbiologists tend to operate at different scales than oceanographers, but CMOP's archive serves both communities.

To provide additional expressiveness across the range of possible datasets sizes and scales, we incorporate the idea of hierarchical metadata. As described in our model in Chapter 3, an individual dataset may be partitioned into multiple “virtual datasets”, for example, when a dataset covering a long time period or large geographic area is divided into smaller subsets. A dataset summary may correspond to the contents of a single physical dataset or file as stored, say, on a disk or in a relational database table; however, a summary may also represent a subset or superset of such a dataset. Each subset summary carries with it the relationship it has to contained subsets and to the overall dataset; we organize these relationships into a hierarchy (further described in Section 4.2). We allow summaries from a dataset and its subsets to participate in multiple hierarchies, thus supporting coexistence of alternative partitioning approaches. Dataset summaries from one or more – or even all – levels of a hierarchy can be returned for a single search.

In this way we address the diversity among the “meaningful units” of data for multiple scientists, and between those units and the archive’s unit of convenient dataset creation and management. By adding this additional level of flexibility, we provide the capability for a scientist to locate a small, highly relevant subset of data that might otherwise be lost within a much larger dataset.

In the example in Figure 2.5, a dataset for a two-month cruise is broken up into individual days (a temporal split), and then into individual cruise segments or *legs* (a geographic split). Lynda can find and download just the two hours she is interested in, while Joel can

retrieve the whole day that matches his needs. Neither scientist needs to download the entire cruise dataset just to get the relevant portions.

## 2.7 Related Work

In this section we summarize related work, to provide a context for our own work. Our approach to the problem of searching scientific archives is, to our knowledge, unique. Our work touches on and is informed by many fields: structured data query, scientific search systems, XML search, geospatial search systems, Information Retrieval (particularly online search engines), digital libraries, spatial cognition and cognitive science. Since each of these fields is in itself vast, we select a few representative works in each case. In addition to this overview, each chapter contains a section of related work specific to the contents of that chapter.

*Structured Data Query and Information Retrieval.* Searching for numbers in “big data” is traditionally the purview of database query approaches, for example by using SQL queries to identify a range of numbers to be returned.

As noted by Saracevic [115], the notion of relevance differentiates IR from database retrieval, although databases may be used as an underlying technology for implementing IR. In fact, we use that approach. Database retrieval is set-based. The database query engine compares query parameters to each data item and returns a Boolean denoting whether the item matches the search criteria and should be returned in the result set, or not. Relevance, on the other hand, inherently has a scale associated with it; an item may be very relevant or only somewhat relevant, and this variability in relevance is important.

Boolean search results are not ranked [80]; as the number of possible results increases, use of a ranking function is important in identifying the best results to return, but the concept of “best results” is not defined for Boolean search.

Database retrieval can be contrasted with information retrieval for text in the following way. Database retrieval addresses structured data via formally defined queries and provides a set of exact results with a formal theoretical foundation. Information retrieval searches mostly unstructured, free text to meet imprecise information needs and provides a combination of more-relevant and less-relevant results, ordered according to some notion of relevance. For example: we accept that Google and Microsoft Bing might return different results for the same search, but we would be surprised if IBM DB2 and PostgreSQL returned different results for the same SQL query over the same data.

As a result of these differences, IR research typically focuses on effectiveness of scoring and ranking, while database research focuses on evaluation efficiency. Chaudhuri et al. [26] note that databases and IR systems developed separately since they address different problems.

*Scientific Search Systems.* It is common in science-based data search to apply query-based approaches. For example, Tata et al. [127] query biological sequence data by extending SQL with *Match* and *Augment* operators. Mork et al. [101] extend SQL into PQL, a query language for semi-structured data that operates over a mediated schema of relationships such as is often found in biological databases. Leser [77] develops a declarative query language based on SQL (and also named PQL, Pathway Query

Language) for querying large protein interaction databases. Unlike these field-specific approaches, our goal is to develop as generic a search method as possible while remaining effective.

Many scientific archives support searching the text in scientist-contributed metadata associated with datasets [103, 109]; these searches are primarily Boolean in nature.

*Search Over XML.* Another approach to structured data search is found in research addressing searching of XML documents, such as INEX-XML (Initiative for the Evaluation of XML Retrieval) [32, 153]. Scientific data could be represented as XML (such as by using XSIL [16] or in ChemML [102]), so conceptually XML search applies. Our scientists' queries could be considered similar to the “content and structure” queries described in the INEX-XML entity-ranking track. However, scientific datasets are not described by a DTD, and would need to be converted into XML format. In addition, XML search focuses on finding the text words used in the search in the identified part of the XML structure. Scientists do not generally search for a specific number (“14.239”) in a dataset. Even if the data were converted to XML, similarity must be calculated using a method that can account for many repeated uses of a number, and for relevant datasets that do not contain the searched-for number at all.

In contrast to text or XML document-retrieval systems, our system’s searches have a potential for greater dynamic range in granularity: a scientist may be searching for a single day or week, or for data spanning a year. Search engines usually index material at a single granularity, such as the web page. We operate over multiple granularities and

sources, using our hierarchies to intermediate between meaning-bearing units at multiple scales and thus gracefully adapting to differences in research foci and scientists.

There is much research into ranked relevance of unstructured text documents and XML with text queries (e.g., [8, 32, 83, 87]). We adapt these ideas to ranking the relevance of scientific datasets. Our initial work on search assumes each search term is a numeric range.

*Geospatial Search Systems.* An important characteristic of much scientific data – in fact of much data – is the time and location in which an observation took place (or is predicted to take place, in the case of simulation models). While time and location can be considered to be variables like any others, their special importance has led to much research focused specifically on them, in the form of geospatial-temporal search systems. Most geographic search systems [47, 51] score items based on word matches against metadata without considering the temporal span or geographic content of the items returned. State-of-the-art geospatial portals [151, 152] allow searches using both geographic and temporal criteria; generally, three spatial tests are supported (the map view *intersects*, *mostly contains*, or *completely contains* the dataset), and temporal searches appear to be simple *contains* tests. Other fields, variables and contents are generally not considered, although some systems handle them by allowing text searches for specified words within selected metadata fields. In contrast, we explicitly rank returned items based on combined temporal, geographic and variable “distance” of the dataset contents from the search [91].

Grossner et al. [51] provide a summary of progress in the previous decade in developing a “Digital Earth”, and identify gaps in efforts so far. They note that the leading-edge spatially aware portals and libraries allow a user to locate information by identifying either a place or spatial footprint and then applying one or more filters. They also note that current geographic and temporal search responses provide matches only on one level of a cataloged object, for example, a one-level catalog entry; further, the contents of cataloged digital objects are not exposed and are not searchable, so users are limited to searching the one level of metadata captured for that object. In contrast, we capture and expose in the search engine multiple levels of metadata for a single dataset.

One widely used geospatial search system is Google Maps [94], which supports searches for a place name or a specified latitude and longitude and provides nearby points of interest (“restaurants near here”). They do not currently expose a temporal or attribute-value search capability. It is possible for a website to explicitly link a dataset to a specific location using KML, but it is not currently possible to search ranges within linked datasets. Egenhofer describes some desired geographic request semantics but does not propose an implementation [36].

*Online Search Engines.* Many Internet search engines today optimize search and infer relevance based on a global ranking of documents [80], often by counting citations or links to or from a target document. For example, PageRank [17] counts and weights links to a webpage from other pages, and then normalizes the result. These methods do not appear to apply to dataset search, as the datasets do not (normally) contain links to other

datasets. Furthermore, some web-search engines only index a prefix of long documents [17]. In our context, a dataset with a million observations may have subsets with widely varying relevance to a search; a prefix of a dataset might be very unrepresentative of the whole (for example, a cruise that transits from fresh to salt water). Also, treating the whole dataset as a “bag of numbers” (as documents are often treated as a “bag of words”) does not assist the scientist; the numbers listed in the search may not appear in the dataset at all.

*Other Systems.* Hill et al. present a system for describing and searching a library’s digital collection of geographic items [61]. They apply widely accepted collection concepts from paper-based archives that are based on a textual description of a map series (publisher, title, number in series, etc.) to digital map collections. A single collection may contain a set of maps where each map has a different geographic coverage; however, the specific map’s geographic coverage is an access or index key to that map. The challenge is how to represent these collections by searchable metadata. They differentiate *contextual metadata*, which is externally provided (e.g., publisher), from *inherent metadata*, derived from automated analysis of the data (e.g., count of items included in a collection). This automatic data analysis adds to the metadata but does not allow the content itself to be searched. They do not provide hierarchical metadata, nor do they discuss methods for ranked search results.

The concept of ranked relevance has also been adapted from text document search into other fields, such as similarity and retrieval of music [134] and content-based image

retrieval [30]; in these fields, some extremely successful applications exist, but the technology has not yet been widely adopted. In concept, the datasets we “index” could be image or music files; we hope to test these extensions in future work.

Venetis et al. extract and search numbers in HTML tables [138], but their work focuses on extracting additional semantics. Agrawal and Srikant [4] also search numeric data, but both these approaches assume each “document” is small by our standards (for example, a single web page, say containing a product specification). Cafarella et al. also search over a large number of tables extracted from HTML, with a goal of returning ranked search responses based on table content [20]; they match on contained values or on synonyms of search terms, and appear to treat a number as a word. To our knowledge, ours is the first application of these ideas to archives of large, heterogeneous datasets.

The work perhaps most similar to ours in scope and spirit is by D’Ulizia et al. [28]. Like us, they are interested in providing approximate results ranked in similarity across a combination of geospatial and non-geospatial attributes. They obtain these ranked results via search relaxation. Relaxation is performed on three kinds of constraints: topological (for geospatial similarity), semantic (by evaluating the similarity of the concepts represented by the attributes, using an information-content approach against a given taxonomy) and structural (similarity between the sets of attributes, mediated by an ontology of attributes, and calculated by using a bipartite graph-matching approach on the ontology). Their approach differs from ours in that each attribute is assumed to have a single value, and the attribute and the values are matched via known ontologies and

taxonomies. They provide alternative queries (rather than results) to the user by relaxing the search terms if no matching terms are found, and they compute the similarity of the relaxed search to the original search (rather than to the database contents). All their data exists within their database and they match individual data items, whereas we work with summaries of large datasets where the datasets may be stored elsewhere.

### 3 Model

In this chapter, we describe our search model, using an example search and dataset. We formally describe our model for feature extraction and similarity scoring. We first decompose and describe the feature extraction portion (refer to Figure 2.4) in Section 3.1, then decompose and describe the similarity scoring portion in Section 3.2. Section 3.3 describes extensions to the initial model to provide the adaptability across multiple scales described in Section 2.6.2. Section 3.4 shows how the componentized nature of the model allows individual components to be modified independently.

We use as our running example a scientist searching for observations taken in June 2010 in a specific area (an area near the Astoria-Megler bridge between Oregon and Washington), containing temperature data with values in the range of 5 to 10C. We refer to an example dataset, saturn01.ctd.201005, described in Figure 3.1, containing the temperature and salinity data collected at one location for two weeks during the month of May 2010. This example dataset contains many temperature values within the desired range, without containing the actual values of 5 or 10. We also have other datasets containing temperature values in the range 10.01 to 15C taken in late 2010; we would like to find those datasets in preference to those, say, from 2008 with temperatures between 20 and 25C.

Formally: we wish to relate a dataset  $d$  to a search  $Q$  via a similarity function  $\text{Sim}(Q, d)$  that quantifies the similarity between the two. However, there may be many datasets, they may be large, and there may be many different formats and types. Scanning the datasets

in response to each search is not practical [48, 49]. To provide interactive response times and scalability, for each dataset  $d$  we wish to identify a small summary  $s$  of  $d$  that we can use to compute similarity more quickly. Thus, we desire a feature extraction or summarization function  $F$  that operates over  $d$  to produce a summary  $s$ , and a similarity function  $Sim\_s(Q, s)$  that produces (approximately) the same results as  $Sim(Q, d)$ :

$$s = F(d) \quad (1)$$

$$Sim(Q, d) \approx Sim\_s(Q, F(d)) = Sim\_s(Q, s) \quad (2)$$

Our strategy is to choose summarization and similarity functions that allow us to compute the summary  $s$  in advance, and evaluate  $Sim\_s(Q, s)$  quickly at the time the search  $Q$  is presented. We show the relationship of these functions diagrammatically; Figure 2.4a shows the ideal similarity scoring approach in the absence of any constraints, while Figure 2.4b shows the approximation we seek plus the first level decomposition into extracting features into a dataset summary for each dataset. We will apply our similarity scoring function to this dataset summary.

### 3.1 Feature Extraction

First, we describe the feature extraction function  $F$  that creates our dataset summaries. We consider each dataset  $d$  as consisting of some “global” information  $d_g$  that applies to the whole dataset, such as identifier, physical storage location and format; and a set of  $m$  variables  $\{c_1, c_2, \dots, c_m\}$ . The number and names of the variables may be different for each dataset, that is,  $m$  is dataset-dependent and  $c_i$  for dataset  $d_y$  is not required to be the same as  $c_i$  for dataset  $d_z$ . In our example dataset in Figure 3.1, global information includes

Summary Field	Dataset Example	Feature
Dataset id:	saturn01.ctd.201005	$sc_g$
Description:	Saturn-01 Profiler, May 2010	$sc_g$
Quality:	"Verified"	$sc_g$
# Observations:	247,377	$sc_g$
Data Location:	<a href="http://...">http://...</a>	$sc_g$
Data Format:	NetCDF	$sc_g$
Times [start:end]:	2010-05-14 :2010-05-31	$sc_g, sc_3$
Geolocation, datum:	Point(-123.87,46.23), WGS84	$sc_g$
Elevations, datum:	-13 .. 2.5 (m), NGVD27	$sc_g$
Variables (units) [values]:	salinity (psu) [0 :29.6] temperature (C) [8.2 :14.6] time (secs since epoch) [1,273,869,578 :1,275,378,800]	$sc_1$ $sc_2$ $sc_3$

Figure 3.1. Example of a dataset summary

the dataset id, description, quality, and so on; the set of variables is  $\{salinity, temperature, time\}$ , with the time variable also promoted to and repeated at the global level.

We create the summary  $s = F(d)$  from each dataset  $d$  by applying the summarization function  $F$ . As is common in text and image retrieval, the summary will be a list of  $p$  features, each produced by a function  $f_i(d)$ . We capture dataset-level summary information for the dataset using a function  $f_0(d)$ . Thus, the summary is of the form:

$$s = F(d) = \langle f_0(d), f_1(d), f_2(d), \dots, f_p(d) \rangle \quad (3)$$

Each function  $f_i$  summarizes a particular aspect of the dataset. There are many possible such functions. To maximize utility, we wish to identify functions that capture features that most closely match the way our scientists appear to think about their data. Further, we wish to capture features that correspond to possible search terms. We conjecture that scientists often visualize their data in (possibly multidimensional) tables with each

variable in a separate column; and that their information needs can, in general, be fairly arbitrary subsets of the data in these columns. Therefore, in order to match possible search terms to captured features, we generally summarize each variable or column separately. However, where a group of variables or columns has semantic meaning (such as a latitude column and a longitude column, together defining geospatial location), we may summarize the group with a single feature. We may also choose to do both: summarize a group of variables to produce a single feature in addition to summarizing each of the variables as individual features.

For simplicity, we will describe a tabular example, with each individual observation in a row and each variable stored in a separate column. We refer to the summary for column  $x$  as  $sc_x = f_x(d[c_x])$ , where  $d[c_x]$  is the  $x^{th}$  column of dataset  $d$  and the summary  $sc_x$  is produced by the function  $f_x$ . Dataset-level summary information is represented by  $sc_0 = f_0(d)$ . Thus, a dataset summary is described by:

$$\begin{aligned} s &= \langle sc_0, sc_1, sc_2, \dots, sc_m \rangle \\ &= \langle f_0(d), f_1(d[c_1]), f_2(d[c_2]), \dots, f_m(d[c_m]) \rangle \end{aligned} \tag{4}$$

Figure 3.2 shows diagrammatically the relationship of the functions and components; we highlight in red a feature extraction function creating a single feature within the dataset summary. Note that we may also have  $sc_x = f_x(d[c_a, c_b])$ , or  $sc_x = f_x(d[c_i, c_j, c_k])$ , and so forth.

At an archive level, we have a set  $D$  of  $n$  datasets:  $D = \{d_1, d_2, \dots, d_n\}$ . We perform feature extraction for each dataset, and produce the resulting collection of summaries  $S = \{s_1, s_2, \dots, s_n\}$ .

In practice, there may be a library  $L$  of feature-extraction functions  $F$  from which we choose our summarization functions based on some rules (such as dataset format or type).

In our running example, we have a dataset-level summarization function  $f_0$  that collects dataset-level information such as file type and filename, and adds other static information such as physical data location; some column information may even be repeated at the dataset level, as is shown in Figure 3.1 for time, geospatial location and elevation. For feature summarization, we choose a single function  $f_b$  that abstracts the values for each

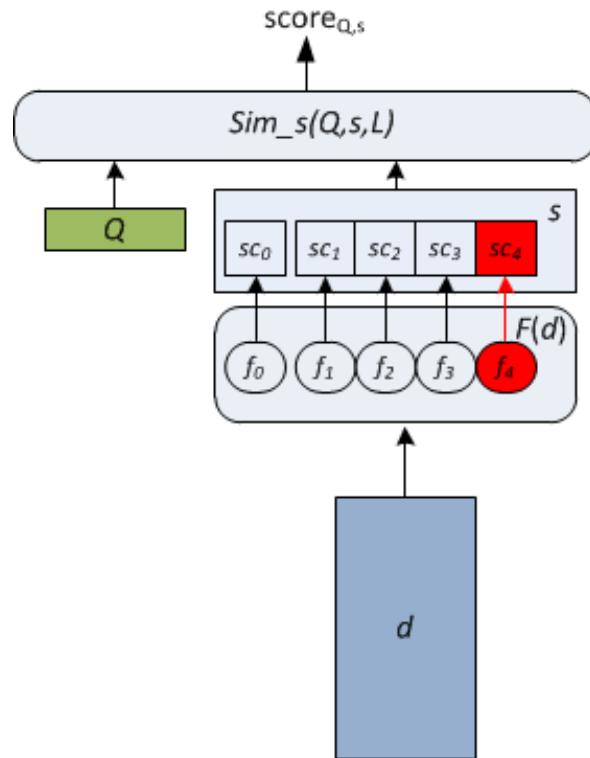


Figure 3.2. High-level depiction of creating and scoring a dataset summary

variable by their bounds, although other choices (such as a median or a Gaussian distribution) are possible. The resulting summary for each variable contains its identity (i.e., name), bounds, and possibly (as shown in Figure 3.1) information such as its units and data type in this dataset. In our example we represent each individual variable summary by a tuple:  $\langle \text{variable name}, \text{bounds}, \text{units} \rangle$ . Thus:

```
s(saturn01.ctd.201005) = F(saturn01.ctd.201005)
= <f0(saturn01.ctd.201005), fb(salinity), fb(temperature), fb(time)>
```

where

```
sc0 = f0(saturn01.ctd.201005)
= <"saturn01.ctd.201005", "verified", point(-123.8, 42.2)>
sc1(salinity) = fb(salinity) = <"salinity", [0:29.6], psu>
sc2(temperature) = fb(temperature)
= <"temperature", [8.2:14.6], C>
sc3(time) = fb(time)
= <"time", [1273869578:1275378800], "secs since epoch">
```

### 3.2 Similarity Scoring

We now turn our attention to modeling similarity scoring. That is, given a search  $Q$  and a set of dataset summaries  $S$ , how do we find the elements of  $S$  that we estimate to have the greatest similarity to our search?

A search  $Q$  is represented as a set of  $r$  search terms  $Q = \{q_1, q_2, \dots, q_r\}$ . In our example, we represent each term  $q_j$  as a tuple of the form  $\langle \text{variable}, \text{range}, \text{units} \rangle$ . Our example scientist's search QE is thus represented as:

```

QE = {("temperature", [5:10], C),
      ("time", [2010-04-15:2010-07-15], date),
      ("location", [Bbox(43.5,-125.2:43.6,-125.5)], lat/long) }

```

We desire a similarity function  $Sim\_s$  that, given a search  $Q$  and a summary  $s$ , produces a similarity score,  $score_{Q,s} = Sim\_s(Q, s)$ , representing the similarity between the search and the summary. We want similarity functions where the similarity computed over the summary is roughly equal to what the similarity computed over the entire dataset would be. We could then separate computing similarity into a one-time, offline function that summarizes the entire dataset, and a lightweight similarity function used during online search that operates only over the summary.

We consider that  $Sim\_s(Q, s)$  is structured as a pair of functions  $Score\_s$  and  $Match$ :

$$score_{Q,s} = Sim\_s(Q, s) = Score\_s(Match(Q, s)) \quad (5)$$

$Match(Q, s)$  matches each search term  $q \in Q$  to one or more features (or, to no feature) in the dataset summary and selects a similarity function  $lf$  (“likeness function”) to use to compare the two. For ease of explication, we use a subscript ( $lf_j$ ) to denote the scoring function used for the  $j^{\text{th}}$  search-term-and-feature combination.  $Match$  returns a list of  $r$  tuples, each one consisting of a search term  $q_i$ , matched feature set  $st_i$ , and likeness function  $lf_i$  to use in comparing them. For each query term,  $Score\_s$  applies the similarity function selected by  $Match$  to the query term and feature it matched to them, and combines the resulting score.

In more detail:

$$\begin{aligned}
Match(Q, s) &= Match(\{q_1, q_2, \dots, q_r\}, \{sc_0, sc_1, sc_2, \dots, sc_m\}) \\
&= \{(q_1, st_1, lf_1), (q_2, st_2, lf_2), \dots, (q_r, st_r, lf_r)\} \quad st_i \subseteq s, lf_i \in L
\end{aligned} \tag{6}$$

The individual similarity functions  $lf_i$  operate over query terms and extracted features. While it might seem that  $Match$  might need a different function for each search-and-feature pair, in practice it can use a library  $L$  of likeness functions, and apply some rules for selecting the appropriate function from that library. For now, we will assume that such similarity functions exist, while deferring a discussion of specific functions to Chapter 4. Different similarity functions can be used for different term-feature matches within a single search  $Q$  and summary  $s$ ; that is,  $lf$  may be different for different features (including column types or columns). There is no requirement in the model that  $Match$  use the same similarity functions across different summaries.

Often we expect a feature to be based on one column (representing a single variable) or a small number of columns of the dataset (for example, where a set of variables make up a single concept, such as location being represented by a combination of a latitude and a longitude column, with possibly an elevation).  $Match$  may choose among multiple features to return for a single search term, or may, as an extension, return a feature that is a combination of variables (such as our latitude-longitude case). If  $Match$  does not identify or choose a matching feature, it may return no tuple for a given search term. Note that the behaviors of  $Match$ ,  $Score_s$  and  $lf_i$ , while partly independent, must be complementary.

We show the relationship of these functions pictorially in Figure 3.3; we highlight in red the flow of a search term being matched to a single dataset summary feature.

In our example, *Match* selects one similarity function for both the temperature and time search terms, and a different one for the geospatial feature. *Match* naïvely matches the variable name *temperature* in the search to the variable name *temperature* in our dataset summary, the *time* term in the search to the dataset's *time* variable, and the geospatial location portion of the search to the *location* information in the dataset summary.

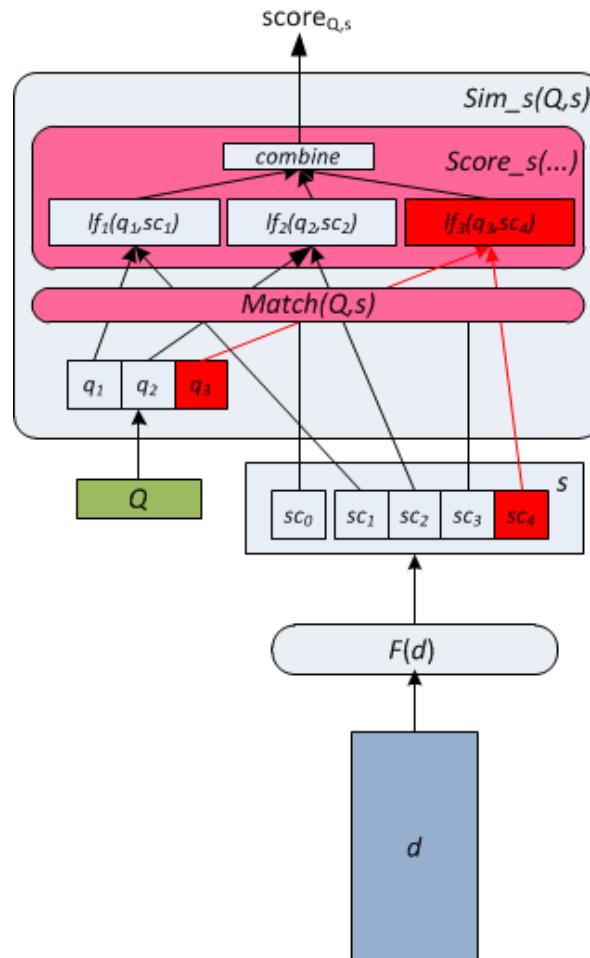


Figure 3.3. Scoring a dataset summary

For each tuple returned by *Match*, *Score\_s* applies the selected likeness function to the search term and selected feature. Each likeness function *lf* returns a score for its term-feature comparison; it must also perform any needed unit and datum transformations, or adjust the scoring for non-transformed data as appropriate. Finally, *Score\_s* combines the resulting scores, using some combination function *combine*:

$$\begin{aligned} \text{Score}_s(\{(q_1, st_1, lf_1), (q_2, st_2, lf_2), \dots (q_r, st_r, lf_r)\}) \\ = \text{combine}(lf_1(q_1, st_1), lf_2(q_2, st_2), \dots lf_r(q_r, st_r)) = \text{score}_{Q,s} \end{aligned} \quad (7)$$

Thus, in effect, we adopt a feature-space model, with each search term being treated as a separate dimension.

In our example search with our example dataset, function *lf<sub>1</sub>* and *lf<sub>2</sub>* (for the time and the temperature terms) are the same function, and *lf<sub>3</sub>* is a different function (for the location search term). Suppose *lf<sub>1</sub>* returns a score of 100 (complete match) for the time search term and variable, and it returns 91 for the temperature term (high similarity, with an overlap between the search and dataset ranges but less than a complete match). Function *lf<sub>3</sub>* is called for the location search term and variable, and returns 100. We choose to combine the individual scores to give a final dataset score by averaging them; thus our final score for this dataset will be 97:

$$\text{Sim}_s(QE, saturn01.ctd.201005) = (100 + 91 + 100) / 3 = 97$$

The purpose of separating *Match* from *Score\_s* (and the *lf<sub>j</sub>*) in the model is to isolate different levels of abstraction, and allow similarity to be assessed at each level independently. For example: we may have a second dataset, containing a variable *temp*

but no variable *temperature*. *Match* may have high confidence that in this case, *temp* is actually the same as *temperature* (perhaps based on matching units (*C*), or other information), and should therefore be returned as the matching feature for the *temperature* search term. For a third dataset, *Match* may make a different choice; for example if *temp* in this third dataset was an alphanumeric code<sup>3</sup>, it may return no feature for this search term. This feature-matching decision is at a separate level of abstraction from calculating the similarity of the summary of a temperature column to a search term for temperature.

Lastly: We desire a result list *Rd* containing the *k* most-relevant datasets in our archive *D* in response to our search *Q*. That is:

$$\begin{aligned} Rd(D, Q, k) = & \langle d_1, d_2, \dots, d_k \rangle, \quad s.t. \\ & Sim(Q, d_i) \geq Sim(Q, d) \quad d \in D - \{d_1, \dots, d_i\} \end{aligned} \quad (8)$$

We actually return our estimate *Rs*(*S*, *Q*, *k*)  $\approx$  *Rd*(*D*, *Q*, *k*), containing the *k* highest-scoring summaries in our metadata catalog *S*. That is:

$$Rs(S, Q, k) = \langle s_1, s_2, \dots, s_k \rangle, \text{ where}$$

$$Sim\_s(Q, s_i) \geq Sim\_s(Q, s) \quad s \in S - \{s_1, \dots, s_i\} \quad (9)$$

We believe the model described here is not limited to any specific scoring or ranking function. In our application of these ideas we have primarily experimented with using the bounds to represent a variable's contents in a dataset, but we believe the techniques are

<sup>3</sup> From Wikipedia: **TEMP** (upper air soundings) is a set of World Meteorological Organization (WMO) alphanumeric codes used for reporting weather observations of the upper regions of the atmosphere made by weather balloons released from the surface level (either at land or at sea). ([https://en.wikipedia.org/wiki/TEMP\\_%28meteorology%29](https://en.wikipedia.org/wiki/TEMP_%28meteorology%29))

applicable to other feature types, such as data distributions. However, the overall results will likely vary in quality substantially depending on the choices made. We require the score-combining function to be monotonic. We expect qualitatively better results if each feature-likeness function produces a score that reasonably reflects subjective distance between the search term and feature. Further, it is desirable that all similarity functions in use within a single system be normalized across features such that the same score implies the equivalent similarity (in the minds of the searcher) between a search term and an individual feature. Widely different feature types (genetic sequence versus geographic location, say) may require widely different feature-likeness functions.

### 3.3 Adaptability: Across Dataset and Search Granularities

Section 2.6.2 raised the issue of potential mismatch between different granularities of data sought by different scientists, and between them and the convenient units for processing and storage of data. We wish to approximate the concept of the most useful meaning-bearing unit [122] for multiple constituencies, with each constituency having a different granularity in mind.

We therefore wish to mediate within our model between the convenient storage groupings from the perspective of the archive owner and the meaning-bearing units of the scientists. We incorporate the ideas of segmenting datasets and creating hierarchical metadata by loosening the correspondence between a single summary and a single dataset, allowing a single summary to represent subsets or supersets of a single dataset.

We can represent a subset  $d'$  of some dataset  $d$  (according to some definition of dataset, such as a single file in an operating system directory, a single directory containing many files, or even an entire archive) by a summary  $s'$ . Thus, for example, a single table stored in a relational database may at times be treated as one or as many datasets. A set of summaries may be composed into a hierarchy, by logically subdividing or composing summaries while retaining knowledge of the relationship of the component parts.

When a dataset is subdivided, we call the individual parts or segments *children*, and we call the original dataset the *parent* of those children. Each child must be a strict subset of its parent, and we must be able to identify its parent (and the children of a parent). The children are not required to be disjoint, and the union of all children is not required to equal the parent. The segmentation of parents into children is neither prescribed nor limited, and is left as a design choice to each archive, dataset collection or feature-extraction process (further discussed in our description of our prototype in Chapter 5).

We call a summary with no parent dataset a *root*; a summary with no children is a *leaf*. Each tree consisting of a root, and recursively, its children, we will for convenience call a *hierarchy*. Each hierarchy is in the form of a tree. The number of levels within the hierarchy is not limited, nor need it be equal on all paths. A *metadata catalog* is a forest of hierarchies, generally representing one or more archives.

To support this extension, we add into our initial model a hierarchy function,  $H(D) = (D', T)$ , where  $H$  is a function that applies a partitioning (or pooling) strategy  $p$  on a set of datasets  $D$ .  $H$  produces a set of datasets  $D'$  and a hierarchy tree (or forest)  $T$  that relates the datasets of  $D'$  in some way. While the original datasets in  $D$  are potentially (but not necessarily) included in  $D'$ , we do wish to ensure that no data is lost in the partitioning; that is,  $\cup D = \cup D'$ . In addition, if there is an edge  $(D_1, D_2)$  in  $T$ , then  $D_1 \subseteq D_2$ . Each

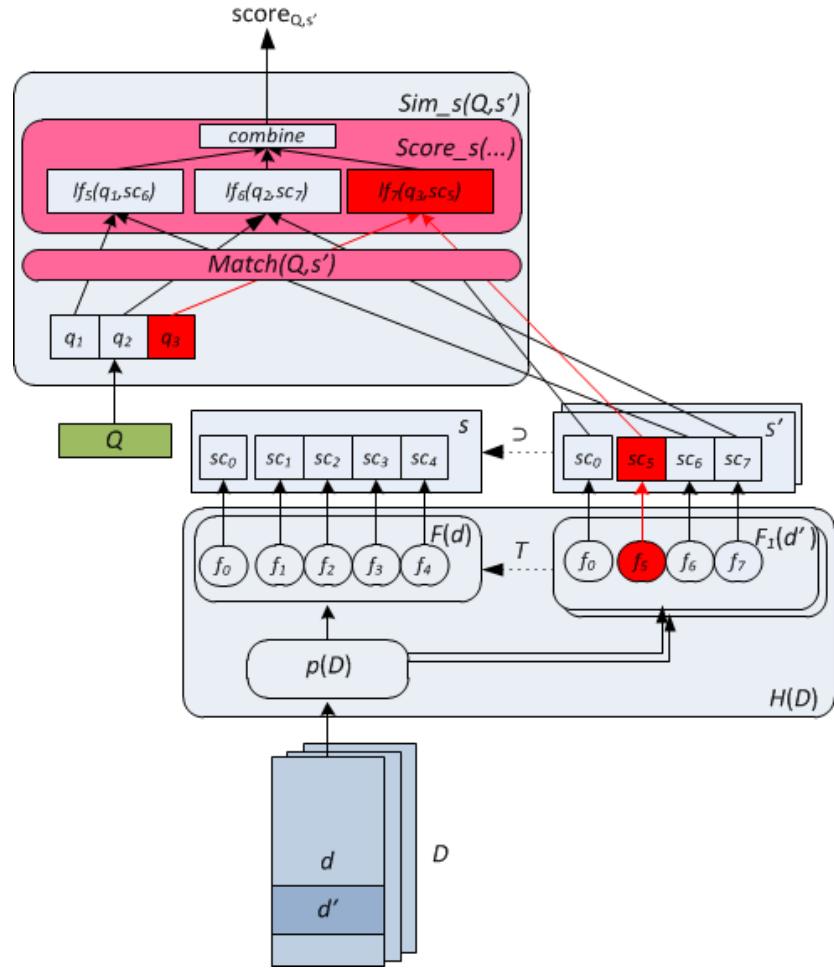


Figure 3.4. A set of datasets partitioned and assembled into a hierarchy, allowing subsets to be separately returned in response to a search.

dataset in  $D'$  is then fed to a feature-extraction function, resulting in a set of dataset summaries  $\{s_1, s_2, \dots, s_r\}$ , one for each dataset; these summaries are composed into one or more hierarchies according to the edges of  $T$ .

There may be multiple hierarchy strategies in use at any one time within a metadata catalog. We may even apply multiple hierarchy-and-partitioning strategies and feature-extraction functions to a single dataset to create multiple hierarchies with different characteristics, for example to meet the needs of different scientist communities or perspectives. In Section 7.5.2 we show different hierarchies built over the same set of data, demonstrating different partitionings representing potentially different perspectives of the same underlying dataset.

Figure 3.4 shows how this functionality fits into the model, extending the diagram from Figure 3.3. We treat parent and child summaries in the same way; that is, both can be matched to a search term and returned in the top-k summaries, and a top-k result list for a search may contain both a child and its parent dataset. Figure 3.4 shows such an additional summary,  $s'$ , representing a subset  $d'$  of a dataset  $d$  being matched to a search term. We can now return multiple potentially relevant subsets (or supersets) of the dataset in response to a scientist's search. Subsets and supersets of a single dataset can appear in the top-k for a single search along with the dataset itself, if the calculated similarity score of each summary is high enough.

### 3.4 Discussion

The model as described gives rise to a componentized implementation architecture.

Various components can be individually modified (see Figure 3.5):

1. Dataset-partitioning approaches
2. Feature-extraction approaches
3. Hierarchy approaches
4. Dataset-summary contents and format

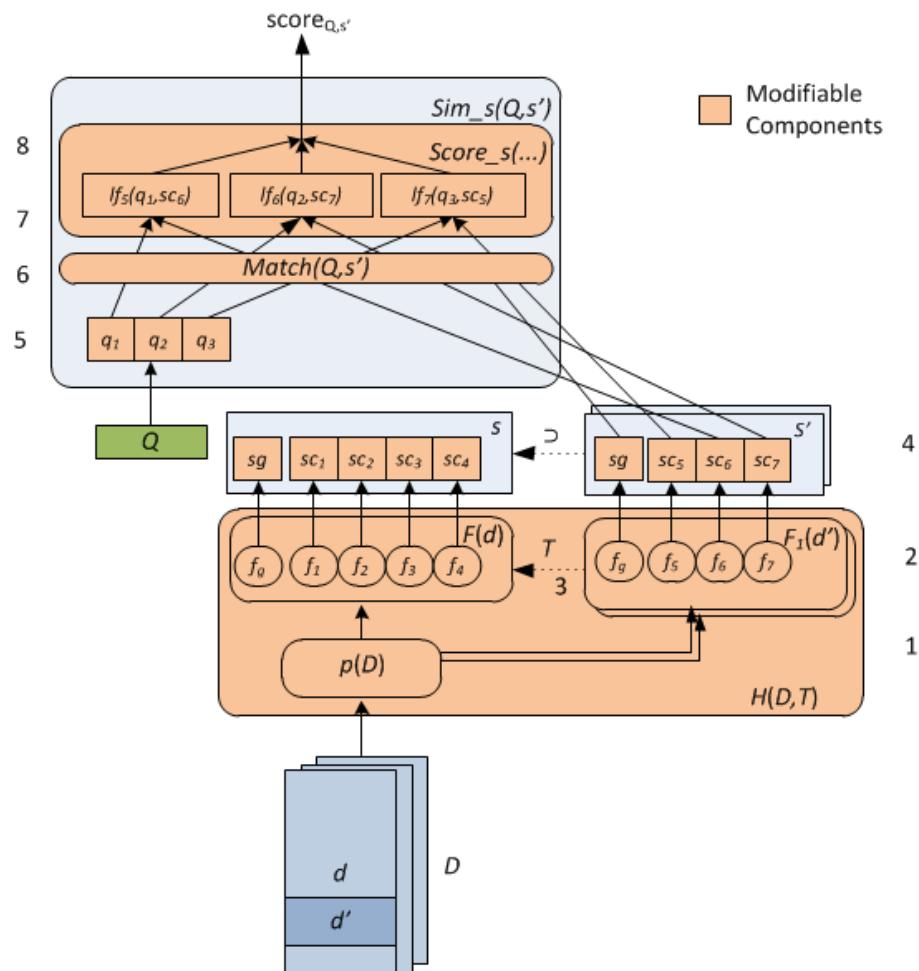


Figure 3.5. Modifiable components in the architecture

Table 3.1. Model Component Dependencies

<b>Model Component</b>	<b>Dependencies and Relationships</b>
Dataset partitioning	None
Feature extraction	Dependent on dataset-summary contents and format
Hierarchy tree	Depends on dataset-summary format, and on partitioning approach used
Dataset-summary contents and format	Used by feature extraction, hierarchy, matching approach, similarity functions
Search term form	Used (and possibly transformed) by <i>Match</i>
Matching	Depends on search-term form, dataset-summary format, and on library of available likeness functions
Similarity functions	Depends on dataset-summary contents. Related to allowable search terms.
Score-combining function	Depends on output from likeness functions

- 5. The form of the search terms
- 6. Matching approaches
- 7. Likeness functions
- 8. Score-combining function

There are some relationships and dependencies between components; for example, the matching function must know how to match the search terms to the dataset summary features. These relationships are outlined in Table 3.1.

Our model addresses and exploits the four identified common characteristics of scientific archives in the following way:

**Heterogeneity:** While the underlying data sources are heterogeneous, we impose a simple, homogeneous metadata model over the data.

**Size:** Our use of hierarchical summaries with dataset subsets and supersets allows us to easily operate over widely differing dataset sizes.

**Variation Over Time:** We can easily accommodate addition of new datasets, new variable names, and new data formats into our model.

**Read-Mostly:** We take advantage of the low incidence of data update by reading datasets only when they are modified; otherwise, the same summary remains in place. Only new data or data that has changed or been regenerated must be read.

Our model provides for significant flexibility. Our prototype is one possible instantiation of the model; many other instantiations are possible. We developed multiple versions of several of the functions (hierarchy, partitioning, similarity) and are running them simultaneously within the prototype, thus showing that the model can be implemented and various components can coexist and be independently modified.

## **4   Dataset Similarity and Metadata Extraction**

The model described in Chapter 3 depends on the availability of appropriate similarity functions and on a metadata catalog of dataset summaries over which to operate. In this chapter we address the question of whether we can realize them:

- Can we identify a similarity measure that will resonate with scientists, and will work with the concept of dataset summaries? Is there any basis in existing research for a similarity function that could apply (relatively generally) to datasets consisting of numeric data, including times and sets of geospatial locations? We explore these questions in Section 4.1.
- Can we create a catalog of dataset summaries from an existing archive, as described in our model? Can we apply the similarity measure we identified to these summaries? We explore these questions in Section 4.2.

Lastly, in Section 4.3, we describe how we evaluate this work.

### **4.1   Dataset Similarity**

We wish to identify and evaluate at least one function that measures the similarity between a user’s search and a dataset. Specifically, we wish to quantify and replicate scientist perception of “similar”; moreover, we wish to order a set of datasets the same way that scientists would, when considering the datasets in terms of relevance to their information need. In keeping with our model, our overall similarity function will be composed from a similarity measure applied to each search term. We recognize that many such similarity measures may exist and that some measures may more closely

replicate scientists' perceptions for some data than for other data. However, initially we seek a measure that can simply and fairly generally be applied and will give "good enough" results; in essence, an analog of an IR measure such as cosine similarity.

We first support our use of a distance-based measure (Section 4.1.1), then in Section 4.1.2 describe our intuition for such a measure. We give the equations for our measure in Section 4.1.3 and then describe the variation we apply to geospatial features in Section 4.1.4. We describe how we apply the equations to a catalog of dataset summaries in Section 4.1.5, and lastly, in Section 4.1.6 we summarize related work in the field of similarity.

#### **4.1.1 Data Distance as Similarity**

Traditional text IR treats a document as a bag of words, with each distinct word a feature; further, a word used frequently across documents is seen as having less value than a less frequently used word. A text IR search also consists of a bag of words, and thus each search term can be matched to a document feature. Our scientists, however, do not search for specific values found in a dataset (that is, they do not search for "water\_temperature = 5.93615C", or even "5.93615"), but rather express their information needs in terms of an observational variable with values in some range ("water\_temperature between 5 and 10C"). Thus, we rejected the bag-of-words model in favor of using variable names and their value ranges as our features and our search terms, and developing a similarity measure that allows us to compare them.

Cognitive science has long recognized that people frequently use distance as a metaphor for similarity, including interpreting time (and other variables) as distance [75]. Tversky and Gati [132] point out that “the notion of similarity – that appears under such different names as proximity, resemblance, communality, representativeness, and psychological distance – is fundamental to theories of perception, learning, and judgment.” They note that similarity relations have been dominated by geometric models; these models represent points in a coordinate space, generally assumed to be Euclidean, and the distance between the points is taken as a measure of their similarity. In the field of spatial cognition, Fabrikant [39] notes that a spatial “near-far” image schemata is often used as an abstraction for similarity (for example in estimating distance between documents based on content similarity), and that this metaphor carries over into multiple (non-spatial) dimensions. In further support, Lakoff notes that interpreting time as distance is a common metaphor (for example, “far in the future” or “they are close in age”) [75]. While it is well known that people are inaccurate in their estimates of absolute distance, research shows that they are relatively consistent with each other in ordinal rankings [100, 112].

Salton’s vector-space model [35], the dominant model in Information Retrieval today, applies this notion of “similarity as distance” to document search. (In fact, strictly speaking, similarity is the inverse of distance, in that small distance is regarded as high similarity.) Each document is represented as a vector of individual terms (often words or word stems) and their frequency within the document. Similarity between a search

(represented as another word vector) and a document is computed by calculating the (inverse of the) distance between the two vectors; a common formula used is cosine similarity [83], which calculates the cosine of the angle between the two vectors. Until recently, words in documents were generally assumed to be unrelated to each other; that is, they were assumed to be “orthogonal” in spatial terms, and were treated as independent vector terms. Salton and McGill [35] reportedly regarded orthogonality as a reasonable approximation.

Given the fundamental nature of the near-far model in human cognition, we believe it can be applied to dataset similarity. For searches over datasets, we hypothesize that the same “near-far” similarity abstraction can be extended to comparing the value range of a variable in the dataset to a desired range of values, and further, to comparing an entire dataset to a search consisting of several individual search terms. We further hypothesize that treating separate variables within a dataset as independent and therefore orthogonal is a reasonable approximation. Variables representing space and time are already often regarded as orthogonal, and modeled as such [105]. Creating a measure that captures absolute distance as perceived by people is unlikely, as people are not consistent amongst themselves. However, we may be able to create a measure that adequately replicates their ordinal rankings, and use that measure to order a set of search results.

Tversky [131, 132] argues on both theoretical and empirical grounds that the metric and dimensional assumptions underlying the distance-based representation of similarity is unfounded. He argues for a set-based approach in which objects are represented as a set

of features, and similarity is modeled as a feature-matching process where the larger the intersection between two feature sets, the more similar the objects are considered to be. In some ways we borrow from this thinking when we encounter overlaps between a variable’s range and a search term, and in accounting for similarity of datasets that contain subsets of the desired variables. It is likely that a dataset containing a desired variable (but with a poor match in terms of the data range) may be considered as “more similar” than another dataset that does not contain that variable (but is the same in other ways). Our work currently has this level of refinement only for a subset of search term types (“existence” search terms).

#### **4.1.2 Dataset Similarity: The Intuition**

If a search term specifies a desired variable with a desired data range and we find a matching variable in two datasets, we seek to rank these datasets in an order that resonates with the scientists. We use a distance-based measure that compares the search range to the data range. While we use distance as a basis, we are approximating similarity, thus, a smaller distance translates to a higher similarity score.

In this section we use a running example, provided by a CMOP scientist: imagine a microbiologist looking for “any observations near the Astoria Bridge in June 2010 with temperature between 5 and 10 C,” because she wants to place a water sample taken there into physical context. In this case, the search has three search terms: one geospatial and two one-dimensional variables, one of them being temporal. Each of the given search terms can be converted to a variable (*location*, *time*, *temperature*), a range (some distance

around a physical area, *June 1:June 30 2010, 5:10C*), and some units in which each range is specified.

The scientist has a qualitative intuition about which observations she considers a perfect match, relatively close, or too far from the search to be interesting. Consider a dataset with a temperature variable whose values range between 6 and 9C. All temperature data in that dataset falls is in the range of 6 to 9C, and so matches the temperature search term. Given three other datasets, with temperatures between 8 and 12C, between 11 and 15C, and between 16 and 22C, we have a good idea how she would rank them in order of closeness to this search term (that is, in the order listed here). We aim to develop a similarity measure that resembles her qualitative intuition; that is, it computes, for a numeric search term expressed as a desired range, the distance between that range and the dataset's values.

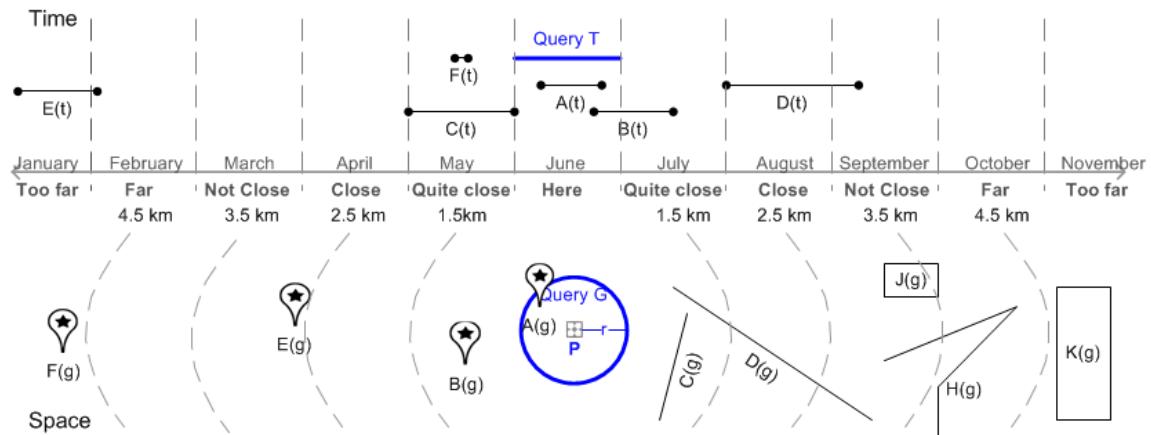


Figure 4.1. Example of qualitative geospatial and temporal ranking: the top section shows a temporal search  $T$  and the time spans of various observation datasets. Dataset  $A(t)$  is a complete match, while datasets  $B(t)$ ,  $C(t)$ ,  $D(t)$ ,  $F(t)$  and  $E(t)$  are at increasing times from the search. The bottom section shows a notional map with geospatial search  $G$ , and with the geospatial locations and extents of several observation datasets represented by points (shown by markers), polygons and lines at various distances. In the middle is a qualitative scale that applies to both time and space.

Figure 4.1 depicts the geospatial and temporal search terms and several datasets for this example; we will use this figure to further provide an intuition about how to measure *dataset distance*. We begin our example with a one-dimensional variable: time. Using the “similarity as distance” metaphor, the top of Figure 4.1 shows a timeline and a temporal search, denoted  $T$ , with a line representing the search time span of *June 2010*. We can consider the temporal search to have a center and a radius; here, the center is midnight on June 15 and the radius 15 days. Lines  $A(t)$ ,  $B(t)$ , ...,  $F(t)$  represent the time spans of observations stored in datasets  $A$ ,  $B$ , ...,  $F$ . Span  $A(t)$  represents a complete match; all observations in this dataset are from June. Span  $C(t)$ 's observations span the month of May and so it is “very close”; Span  $B(t)$  overlaps the search time span and is therefore “closer” than Span  $C(t)$  but is not a complete match. Span  $F(t)$ , representing a small time period in the middle of May, is also “very close”; it is arguable whether or not it is closer than  $C(t)$ . Span  $D(t)$  is further away still and Span  $E(t)$ , with observations in February, is “far” from the June search. Note that we can treat temperature, also a one-dimensional variable, the same way.

The bottom section of Figure 4.1 shows our two-dimensional geospatial search term  $G$  as drawn on a (notional) map. The search area is here represented by a central point  $P$  (in our running example, geo-coordinate (46.23, -123.88), near the Astoria bridge), and a radius  $r$  ( $\frac{1}{2}$  km) within which the desired observations should fall. The marker labeled  $A(g)$  represents the geospatial extent of observations in the same dataset  $A$  whose temporal extent was shown above. For dataset  $A$ , the observations are at a single location,

for example, observations from a fixed station or a set of observations made while anchored during a cruise. Markers  $B(g)$ ,  $E(g)$  and  $F(g)$  represent single-location datasets further away from the search center. Linear extents  $C(g)$ ,  $D(g)$  and  $H(G)$  represent transects traveled by a mobile observation station such as a cruise ship, AUV or glider. Polygonal extents  $J(g)$  and  $K(g)$  represent the bounding box of a longer, complex cruise track. Point extent  $A(g)$  falls within the radius of the search and so is a complete match to the geographic search term. Note that the qualitative measure of “closeness” remains consistent across geometry types. Despite the differences in geometry types represented, marker  $B(g)$  and line  $C(g)$  are both considered “very close” and polygon  $K(g)$  and marker  $F(g)$  are “too far” from the search to be interesting. In fact, the scientist is applying an implicit scaling model that is specific to his task [99]. Support for a “too far” judgment can be found in Montello’s spatialization study [98]; the exact distance at which this judgment is applied may change for different users or even for different tasks [99].

The same intuitive scaling can be applied across the different search terms. For example, temporal observations at  $F(t)$  and spatial observations at marker  $B(g)$  could be considered equidistant from their search centers, as they are both (qualitatively speaking) “quite close.” Further, when considering both the temporal and spatial distances simultaneously, the dataset  $F$ , with temporal observations  $F(t)$  (quite close) at location  $F(g)$  (too far), is further from the search than datasets  $A$  (“here” in both time and space),  $B$  and  $C$  (“quite close” in both time and space). These examples illustrate the situation of one dataset *dominating* another: being closer in both time and space. The more interesting case arises

in ranking two datasets where neither dominates the other, such as  $D$  and  $F$ :  $F$  is temporally closer, but  $D$  is closer in space.

To simplify such comparisons, we use the search radii as the weighting method between the multiple search terms. For example, had the spatial portion of the search been “within 5 km of  $P$ ”,  $D(g)$  and  $F(g)$  would both be considered “here” spatially, but  $D$  would now be dominated by  $F$  since it is temporally dominated by  $F$ . This approach allows us to convert each different unit into a unit-less distance measure, scaled separately for each search term based on its radius; we can then easily compare different search terms on a common, unit-less scale.

#### 4.1.3 Estimating Dataset Similarity

To compute these comparisons across a large number of datasets, we need an efficient computation that characterizes the intuition described above. We develop a simple, lightweight formula that approximates the distance between each search term and a matching variable in a dataset based on summary information about the dataset, such as the contained variables and their bounds or footprints. In this section, we describe that formula.

There are many options for representing the proximity of two entities (in our case, a search term and relevant variable’s values), with varying computational complexities [95]. A commonly used surrogate for distance between two geographic entities is centroid-to-centroid distance. While it is a poor approximation when the entities are large and close together, it is relatively simple to calculate, at least for simple geometries.

However, this measure ignores the radii of the search terms, and does not directly identify overlaps between the geometries or ranges of the variables.

Another well-studied distance measure is minimum (or maximum) distance between two entities. This distance can be estimated by knowing only the bounds of the entities. This measure more closely matches our criteria; it can be calculated quickly using information (the bounds) that can be statically extracted from a dataset. We can use the minimum and maximum distances to identify key characteristics that will drive our ranking: whether a dataset is within our search bounds, whether the search and dataset overlap or whether they are disjoint, and if so by how much. This discussion applies equally to the one-dimensional “spaces” of time or other one-dimensional variables.

In essence, we regard the variable’s values (whether of time, space, or some other variable) within a dataset as representing a distribution of “distances” from the search center, with a single point value (such as a constant value for a variable, or a single time) being the most constrained distribution. Each search term itself represents a desired distribution of the appropriate variable. At present, we regard the contents as being equally distributed between the bounds, as is common in database indexing [84]; future research may consider alternative distributions. Other researchers eliminate the highest and lowest values prior to calculating the bounds [84]; this elimination does not alter our discussion.

Our distance measure for a one-dimensional numeric variable, such as temperature or time, is shown in Equations (1)-(4); Figure 4.2 depicts our measure graphically. Without

loss of generality, we assume that there is a monotonic mapping from the variable's domain to the real numbers; in the case of a temporal search term, for example, we convert time values to "Unix time."

We compare the range of each search term to the range of values in the matching dataset variable. We have three cases:

- If the column bounds are within the search bounds for this term, we regard this term as a "complete match"; for example,  $A(v)$  in Figure 4.2.
- If the two are disjoint, we calculate the number of search radii (half the bounds, and centered on the average of the bounds) that separates the middle of the dataset from the closer edge of the search radius. The closer edge is used since the dataset may be numerically below or above the search radius. The higher the number of radii, the lower the similarity. An example in Figure 4.2 is  $E(v)$ .
- Where the ranges overlap, the similarity is adjusted upwards by the percentage of

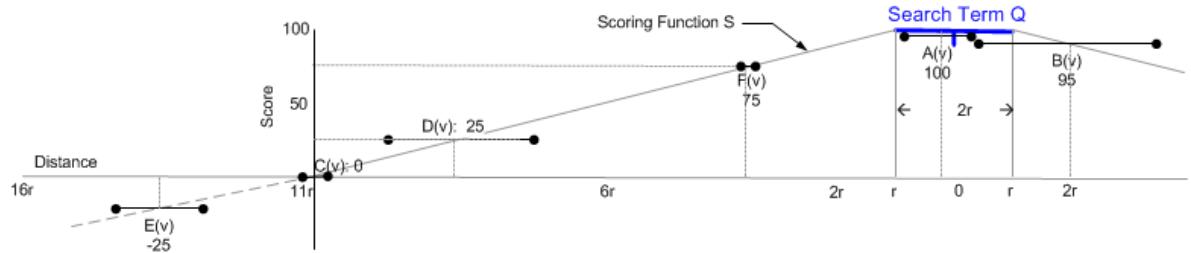


Figure 4.2. Graphic depicting a portion of the candidate distance measure, as applied to a one-dimensional variable ( $v$ ) to calculate a similarity score. The search range  $Q$  is shown in blue; the search radius  $r$  is one-half of the range of  $Q$ . The range for variable  $v$  in each of six datasets (labeled  $A$  to  $F$ ) is shown, along with each dataset's resulting score for this search term. For each dataset, the scoring function  $S$  identifies the middle of the range of the variable  $v$ , scales it by the search radius, and converts the result to a score. Datasets wholly inside the search range (e.g., dataset  $A$ ) are given a score of 100. Datasets wholly outside the search range are given a score based on the number of radii that the middle of the range is from the search center (datasets  $C$ - $F$ ). Datasets whose "middle" is at 10 radii from the search's edge (that is, 11 radii from the center) are given a score of 0, and ones further away may receive negative scores (dataset  $E$ ). Datasets that overlap the search range are scored accounting for the proportion of the dataset range that is inside the search range (dataset  $B$ ).

overlap between them. We compare the ranges rather than the averages or median, since, for example, the sets  $\{10, 30\}$  and  $\{19, 21\}$  have the same average, but they do not have the same relevance to someone searching for data in the range of 18:22. An example in Figure 4.2 is  $B(v)$ .

For a one-dimensional search term, let  $Q_{Rmn}$  and  $Q_{Rmx}$  represent the lower and upper bounds of the term, and let  $v_{Xmn}$  and  $v_{Xmx}$  represent the minimum and maximum values of observations for the matched variable  $v$  in a dataset  $d$ .

Equation 1, shown in Figure 4.3, calculates  $v_{Rmn}$ , the distance of variable  $v$ 's minimum value from the search term's "center", i.e., the mean of  $Q_{Rmn}$  and  $Q_{Rmx}$ , and then scales the result by the search term "radius" (half the size of the term's range). Similarly Equation 2 calculates  $v_{Rmx}$ , the "scaled variable-range distance" of the variable's maximum value. Equation 3 uses  $v_{Rmn}$  and  $v_{Rmx}$  to calculate an overall distance for this variable's range from the search term's range, normalized by the search-term radius. The first subcase shown applies to variable values completely within the term's range, and thus at a distance of 0 radii. Subcases 2 through 4 account for a variable range overlapping the search range at the high end, at the low end, and on both sides, respectively; these subcases adjust the distance calculation based on the percent of the variable's range estimated to be inside the search range. The last subcase accounts for a dataset completely outside of the search term's range.

In Equation 4, we then apply a scoring function  $s$  to  $v_{Rdist}$  to convert the calculated distance in radii from the search term center into a relevance score  $v_{Rs}$ , while allowing a

$$\begin{aligned} v_{Rmn} &= \frac{v_{Xmn} - ((Q_{Rmx} - Q_{Rmn}) / 2 + Q_{Rmn})}{(Q_{Rmx} - Q_{Rmn}) / 2} \\ &= (2v_{Xmn} - Q_{Rmx} - Q_{Rmn}) / (Q_{Rmx} - Q_{Rmn}) \end{aligned} \quad (1)$$

$$v_{Rmx} = (2v_{Xmx} - Q_{Rmx} - Q_{Rmn}) / (Q_{Rmx} - Q_{Rmn}) \quad (2)$$

$$v_{Rdist} = \begin{cases} 0 & v_{Xmn} \geq Q_{Rmn}, v_{Xmx} \leq Q_{Rmx} \\ \frac{(|v_{Rmx}| - 1)^2}{2 |v_{Rmx} - v_{Rmn}|} & v_{Xmn} \geq Q_{Rmn}, v_{Xmx} > Q_{Rmx} \\ \frac{(|v_{Rmn}| - 1)^2}{2 |v_{Rmx} - v_{Rmn}|} & v_{Xmn} < Q_{Rmn}, v_{Xmx} \leq Q_{Rmx} \\ \frac{(|v_{Rmx}| - 1)^2 + (|v_{Rmn}| - 1)^2}{2 |v_{Rmx} - v_{Rmn}|} & v_{Xmn} < Q_{Rmn}, v_{Xmx} > Q_{Rmx} \\ (|v_{Rmn} + v_{Rmx}| / 2) - 1 & v_{Xmn} > Q_{Rmx} \vee v_{Xmx} < Q_{Rmn} \end{cases} \quad (3)$$

$$v_{Rs} = s(v_{Rdist}) \quad (4)$$

$$d_{Gdist} = \begin{cases} 0 & d_{Gmx} \leq r \\ \frac{(d_{Gmx} / r - 1)^2}{2(d_{Gmx} - d_{Gmn})} & d_{Gmn} \leq r, d_{Gmx} \geq r \\ (d_{Gmn} + d_{Gmx}) / 2r - 1 & d_{Gmn} > r \end{cases} \quad (5)$$

$$d_{Gs} = s(d_{Gdist}) \quad (6)$$

Figure 4.3 Formulae

weighting factor to be applied to the distance result, if desired. Per Montello [100], this implicit scaling factor may change for different users or different tasks. Our current implementation (reflected in the example in Figure 4.2) uses  $s(v_{Rdist}) = (100 - f * v_{Rdist})$ . That is, if the distance is  $f$  “radii” (currently we have set  $f = 10$  in our prototype) from the search term’s edge it is considered “too far away” to be relevant and given a score of 0 or less, while a distance of 0 (i.e., completely within the search term’s range) is given a score of 100.

Lastly, the scores  $v_{Rs}$  for each search term, including any geospatial score  $d_{Gs}$ , are combined to give an overall score  $d_{score}$  for this dataset. We currently take a simple

arithmetic mean of these scores. Combining these distance measures results in a multi-component ranking, which is the norm in web search systems today [38, 71, 79, 83]. Note, however, that we scaled each of these rankings by the radii of the respective search terms, resulting in a unitless measure; thus, the user describes the relative importance of time and distance, for example, by adjusting the search-term ranges.

#### 4.1.4 Geometric and Geospatial Similarity

We adapt the measure to the two-dimensional case of calculating similarity of geometric or geospatial search terms and features. By convention, the geolocations<sup>4</sup> within the dataset can be represented by any of the common geometries: point, line/polyline or polygon (e.g., convex hull) [58]. Similarly to the one-dimensional measure in Equations (1)-(4), we use minimum and maximum distance between the search term and dataset feature (see Figure 4.4a) to provide a reasonable approximation of distance for the three primary geometries while minimizing the number and complexity of spatial calculations needed. This approach uses a total of two spatial calculations (maximum distance and minimum distance between two geometries) for each metadata record scored. Spatial functions can be slow, so minimizing the number and complexity of geometries handled is beneficial. We first describe using a geospatial, circular search feature for some area: let  $C$  represent the center location of the geospatial search term and  $r$  the radius. Let the locations of all the observations within a single dataset  $d$  be represented by a geometry  $g$ .

---

<sup>4</sup> By convention, we use “surface of the earth” distance for geospatial search terms and features.

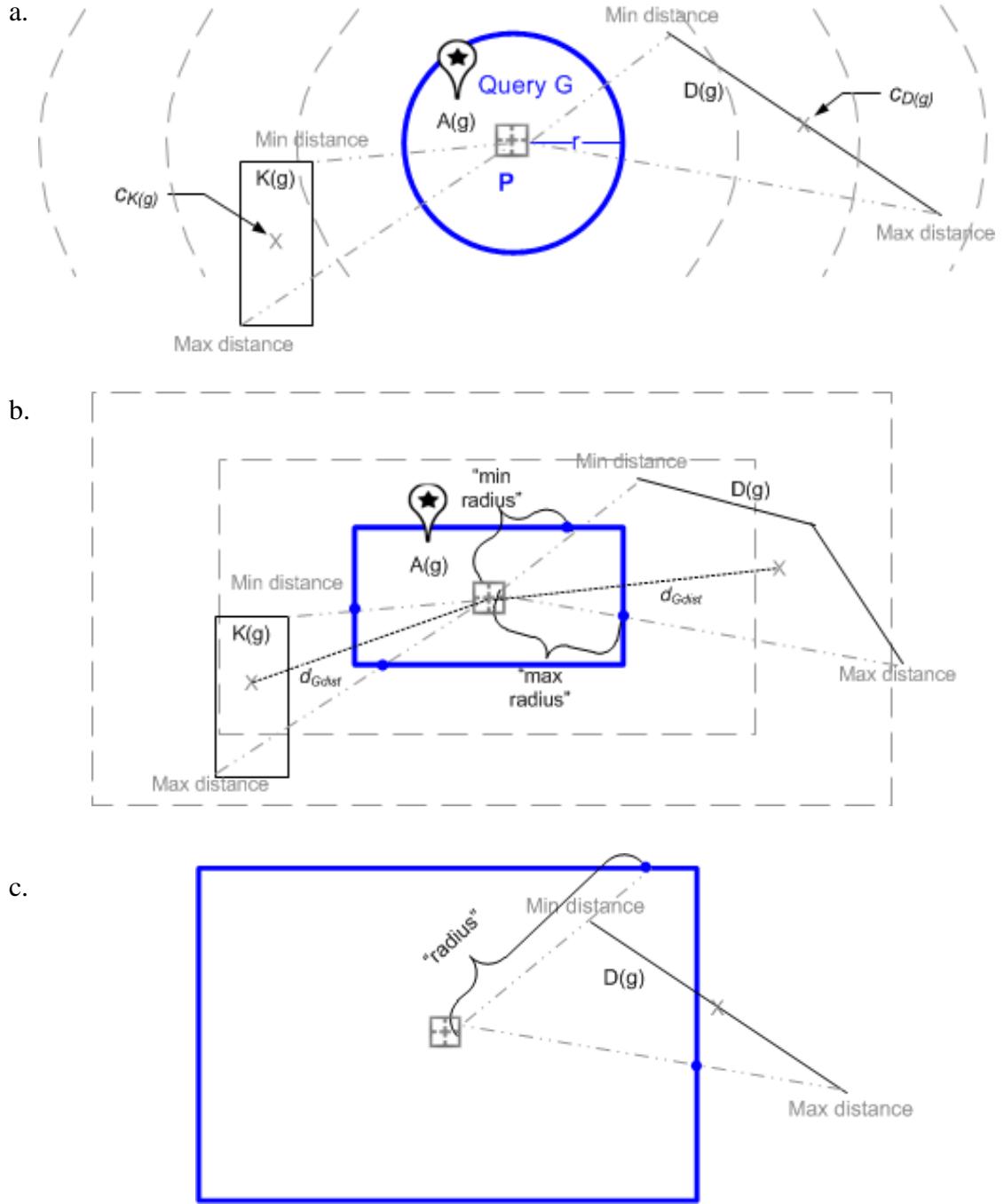


Figure 4.4. Adapting the distance measure to various geometric shapes and search regions. The blue line represents the search region, and multiples of the radius are shown by dashed lines. a. Initial version. Point  $c$  represents the notional “center” for each dataset shown. b. Adaptation from a circular to a rectangular search region. c. Adjustment made for datasets overlapping the search region, with one end point within the search region.

Let  $d_{Gmn}$  and  $d_{Gmx}$  represent the minimum and maximum distances of the geometry from  $C$ , using some distance measure such as earth-surface distance or Euclidean distance. Equation 5 calculates the overall distance measure for three subcases: the dataset's geometry is completely within the search radius; the geometry overlaps the search area, or the geometry is completely outside the search area. Equation 6 gives a geospatial-relevance score  $d_{Gs}$  for dataset  $d$  by again applying the same scoring function  $s$  to the calculated overall distance measure.

Note that we calculate an average between the distance to the closest point and the distance to the farthest point (and scale it by the search-term radius, as before). We treat this distance as notionally being the average distance to the dataset. However, although we show a point in Figure 4.4a that indicates this distance, we do not calculate such a representative point but work solely with minimum and maximum distances.

This measure has several advantages: it takes the overall shape of the geometry into account, unlike a nearest-neighbor approach; it is more nuanced than the often-used categorization of the spatial relationship of two shapes into *contains*, *intersects* or *disjoint*; and it is easy to calculate. A more complex spatial scoring system can easily be devised; what is less clear is whether, given the uncertainties in people's views of distance [99], the additional complexity provides a better distance score as perceived by the user. What is clear is that the additional complexity will add to the computation time.

At the request of the scientists, the geographic search term is represented in the current prototype as a rectangle. We adapted the calculations in the following way. We identify

the closest point of the geometry to the center of the search region, and calculate the distance to that point (see the two examples in Figure 4.4b, for a line ( $D(g)$ ) and a polygon,  $K(g)$ ). Then we calculate the distance to the intersection of a line running between the center of the search region and the closest point and the search-region boundary; this distance becomes the radius we use in scaling the distance to the closest point.

In the case where the dataset overlaps the search region (as in Figure 4.4c), the closest point is within the search region. Here, we extend the line to the search region boundary, and use this distance as the radius of the search region in this direction.

We repeat the process for the farthest point. Thus, the scaled max distance formula uses a different radius from that used in the scaled min distance formula, reflecting the different “yardstick” in that direction. We then proceed to calculate the average of the scaled distances to the minimum and maximum points and the score for the overall geometric search term as before.

We show an example in Figure 4.5. Here, our similarity measure calculates the dataset

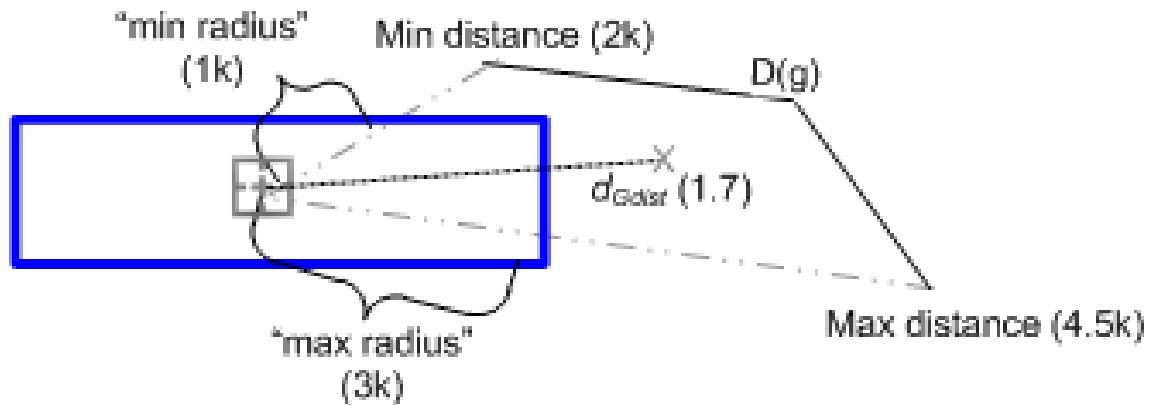


Figure 4.5. Dataset distance calculation for rectangular search region, with distances in kilometers

represented by the line  $D(g)$  as being  $(2/1 + 4.5/3)/2 = 1.7$  radii from the search center, or 0.7 radii from the edge of the search region. We show the calculated notional  $d_{Gdist}$  in Figure 4.5 (although, as noted, we do not calculate an actual point or direction relative to the search region, we work only with distances). Using our current scoring function, the geospatial score for this dataset would be  $(100 - 10 * 0.7) = 93$ , that is, “quite close” to the search region.

This scaling using the intersection point with the boundary works for any convex search geometry. Further, in the case of a circular search region, it reduces to our original measure.

#### 4.1.5 Algorithm for Dataset Scoring

We treat searches as a conjunction of desired features. In general, we represent each search term as a tuple of the form  $\langle \text{variable}, \text{range}, \text{units} \rangle$ . For example:

```
{<"time", [2010-06-01:2010-06-30], days>,
 <"temperature", [5:10], C>,
 <"location", [46.23, -123.88: 47.23, -124.52],
 degrees>}
```

The dataset summaries, stored in our catalog, likewise contain a set of features of the same form. In general, we have one feature for each variable within a dataset. (Exceptions may exist. For example, we combine latitude and longitude into a single geospatial feature, and have considered adding elevation to that.) This model gives us symmetry between the search terms and the dataset features, which allows a returned dataset to itself serve as a search to find similar datasets.

Figure 4.6 shows simplified pseudo-code for our algorithm. For each dataset, we first match each search term to a feature of this dataset, or to no feature (function *Match*). Our initial matching function, used in the user study and reflected in the pseudo-code here, simply looks for an exact match between the variable named in the search term and a named column in the dataset. (This approach is naïve and is not required by our model.

---

```

Inputs: Search specification Q, catalog entry collection D, desired number of
highest-scoring entries k, library of similarity functions LF

Initialize array Scores[]
For each catalog entry d in collection D do:
    Scores[d] = Score(Q,d)
Sort values in Scores[]
Return Scores[1..k]

Score_s(Q,d):
// Calculate similarity score for search Q on dataset d
    C = Match(Q,d)
    L = []           // Accumulate scores in list L
    score = 0
    For each tuple (q,c,lf) in C do:
        score = score + Score_c(q,c,lf)
    Return score / |C|

Match(Q,d):    // Naïve version
// Match search terms to features
    matched = {} // initialize empty set
    For each search condition q in Q do:
        If (requested variable v exists in d):
            Choose lf from LF based on datatype of v
            If q.units != v.units:
                Convert q.range, q.units to match v.units
                Add (q,v,lf) to matched
        Else:
            Add (q,Null,Null) to matched
    Return matched

Score_c(q,v,lf):
// Return score for this search term q, variable v using likeness function lf
// Naive version
    MaxCondScore = 100      // Implementation choice
    NoMatchScore = 0         // Implementation choice
    If lf == Null or c == Null: // No match
        Return NoMatchScore
    Elif (q is a variable existence condition):
        Add s = s + MaxCondScore
    Else:                  // variable with range condition
        Return lf(q, v)     //apply measure

```

---

Figure 4.6. Simplified pseudo-code for dataset summary ranked search algorithm.

Work is underway to address the problems of normalizing the variable names in the archive [90]; see Section 8.3.)

Next, we score each search condition and matched variable.

- If no variable was matched to a search term (for example, the desired variable is *temperature* but there is no matching variable in the dataset), we give that search condition a “null” score.
- If a variable is matched and the search term merely requests the existence of that variable in the dataset (e.g., <“temperature”, , >) we count it as a complete match for the term and give it the maximum possible score (*MaxCondScore*). In concept, this condition specifies a variable with an infinite range of values; thus, any dataset that contains a matching variable is considered “closer” to that search condition than a dataset that lacks that variable. In effect, the resulting score is binary: a dataset is a perfect match to the search condition if the desired variable is found in that dataset, or a complete non-match if it does not. When *Match* is extended to non-exact matches on variables, this score can become more nuanced.
- If a variable is matched and the search term contains a range, an appropriate likeness function is selected. The similarity between the search term and matching feature is scored using the likeness function; our primary likeness functions are described above in Figure 4.3.

After processing all terms, we normalize the final score by the number of terms in the search to give an overall score for each catalog entry. As described above, in our current

implementation we set the *MaxCondScore* to 100; the minimum score is not bounded.

Given a collection of candidate datasets, each dataset's  $d_{score}$  can be calculated.

Optionally, datasets with  $d_{score} \leq 0$  can be discarded if desired.

Finally, we sort the resulting list in decreasing order of  $d_{score}$  into a ranked list and return the catalog entries with the  $k$  highest scores. Conceptually, each search generates a rating of every catalog entry (although for performance reasons, our implementation avoids doing so). We have explored optimizations to this pseudo-code, such as traversing hierarchical relationships between datasets, to derive the same results more efficiently (see Chapter 7).

#### 4.1.5.1 Textual Scientific Data

In addition to the large quantity of numeric data, some datasets contain one or more fields of textual data. For all variables containing textual data, we currently allow only searches for the existence of these variables.

There are a few data fields, such as research notes or study descriptions, for which traditional textual search may apply. However, discussions with our scientists lead us to believe that traditional text search is inappropriate for most textual variables. Examples found at CMOP of such textual variables include quality levels, species names, and unique hybrid textual-numeric identifiers given to water samples and test locations. The notion of distance still seems germane to such variables. For example, quality levels are ordered, two species are more or less related, and the numbering schemes were often

selected for memorability by embedding a notion of geolocation. Thus, we expect notions of distance still apply, though the details of the likeness functions may need to change.

We experimented with adapting our distance measure for ordinal categorical data, such as quality levels. We assigned each quality level a value within a range that respects its ordinal position, with “no quality control” having the lowest value and “full quality control” having the highest. We convert the search term range and data ranges to the matching numeric values and apply our distance measure to these values. While this approach has been well accepted by our implementation’s users, it relies on assessing each textual variable independently for applicability and manually assigning the ordinal values, and so we do not see this approach as viable for large archives with many categorical variables. Other measures will be required for distances that cannot be captured by a linear order, for example, distance between species. In some domains, existing similarity functions may be adapted for use; for example, similarity functions exist for DNA sequences, although an approach is required for specifying a search radius around the desired sequence for our style of search.

Combining and weighting textual and numeric search terms remains an area for future research. While it is mathematically possible to combine scores from these two methods – for example, by including the score for each textual term in the final score normalization – we do not yet have an understanding of how scientists perceive these combinations. In particular, unlike the continuous numeric and existence measures, we have not validated these additional approaches with formal user studies. While there are

certainly technical issues to be addressed, we believe more user studies exploring how users expect these systems to operate to be the most pressing issue.

#### 4.1.6 Related Work: Dataset Similarity

To our knowledge, we are the first researchers to explore the question of what constitutes relevance of a scientific, numeric dataset to a search, and to attempt to develop a broadly applicable relevance measure for such datasets.

Our application of a distance-based similarity measure draws on research in the field of *spatialization of data* [39, 122]. Despite the name, the data searched is not spatial in nature but is nevertheless represented as points in a vector space. Spatial cognition researchers have shown that judging relative distance between individual spatialized data points is practical, and that study participants naturally understand similarity represented as relative distance. We apply their notions of distance more broadly to large sets of combined temporal, spatial and environmental-variable values. Although spatialization research has identified anomalies and inaccuracies in user perceptions at the detail level, we believe a fast approximation of similarity between a search and a dataset has significant value.

D’Ulizia et al. [28] provide approximate results ranked in similarity across a combination of geospatial and non-geospatial attributes; however, unlike our work with data ranges, they assume each attribute has a single value, and they match attributes and their values via known ontologies and taxonomies. Roddick et al. [112] suggest that (even) for numeric values, practical distance may be different from numeric or Euclidean distance

(for example, software version numbers), and go on to develop a unifying model for semantic distance. We believe these concepts might be applicable to matching variable names, and leave further exploration to future research.

As noted earlier, traditional text IR treats a document as a bag of words, with each distinct word a feature. Many similarity functions and measures have been proposed and used for the bag-of-words model. One popular measure in text retrieval is cosine similarity. This measure computes the cosine of the angle between two vectors, one representing the search and the other the document. This measure compensates for two similar documents appearing dissimilar as a result of length differences [83]. In essence, our focus on data ranges has a similar dampening effect on differences, as the counts of specific values occurring are not taken into account in our similarity measure.

One of the assumptions often made in the bag-of-words model is that a frequently used word in the document collection is seen as having less value than a less frequently used word [83]; we can then adjust for these differences in value, as is done when using tf-idf (term frequency-inverse document frequency). However, it is not known that this assumption is valid when applied to data rather than words; for example, while the word “the” clearly carries less information content than a word such as “disintermediation”, it is not clear that a repeated number in a dataset has less value than a rare one. Recently, sophisticated search engines are now taking word co-occurrences and context into account in their similarity-scoring functions [78]; identifying ways to apply these concepts to dataset similarity is left to future research.

Recent text-retrieval systems use a wide variety of ranking criteria. Some of these, such as click frequency, have ready analogs in the dataset world: for example, download frequency as a surrogate for utility. Other criteria, such as reference frequency, will require adaptation of existing scholarly practices to apply to datasets; for example, methods to consistently cite a dataset, and discipline around retaining accessibility to the cited datasets [142]. Research into how to apply these concepts to datasets is warranted. Some scientific fields have developed similarity functions that they apply to their specific data, data formats or problem. For example, Zhang et al. [145] develop a similarity measure for protein structures, using IR feature-indexing techniques to quickly compare their tableau representations. In the field of spatial data, Schwering [118] reviews and compares different notions of similarity in current use, categorizing them into geometric models, feature matching, network distances, alignment and transformational models. Some models apply only to concepts, and others to objects. The notions are described for spatial comparisons only.

Focusing on geographic search, Goodchild [47] notes that most geographic search systems score items based on word matches against metadata without considering the temporal span or geographic content of the items returned (that is, these geographic search systems use text search). Goodchild et al. [46] expand on these concerns in the 2007 review of Geospatial One-Stop (GOS) [151], a state-of-the-art government portal to geographic information. GOS and similar portals such as the Global Change Master Directory's Map/Date Search [152] now allow searches using both geographic and

temporal criteria; three spatial tests are supported (the map view *intersects*, *mostly contains*, or *completely contains* the dataset), and temporal search appears binary: items not matching the criteria are not returned. In contrast, we explicitly rank returned items based on the temporal, geographic and variable-value “distances” of the dataset from the search; the geographic, temporal and variable extent of the dataset are factored into the ranking in a continuous fashion, as opposed to the three discrete spatial tests.

Much of the geospatial search exploration so far has been performed using point data, rather than with the combinations of point and non-point features (lines, polygons) that are common in scientific data. In one exception, Markowetz et al. [85] describe a prototype search engine that uses geographic “footprints” representing the spatial coverage of information. These footprints are associated with webpages to represent the spatial coverage of the information on the page, and are based on geocoding of textual data extracted from the page using a separate database of administrative data. For example, a postcode used on the page may be converted to that postcode’s geographic footprint. The geographic portion of the result is computed by intersecting the search footprint with the page footprint; if they do not intersect, the document is discarded [86]. The results of this comparison are combined with the results from text search to produce the final result. They do not address time and other variables as search criteria.

Addressing a different kind of search problem, Sharifzadeh and Shahabi [119] compare a set of data points with a set of search points, where both sets potentially contain geographic attributes, and identify a set of points that are not *dominated* by any other

points. An example problem addressed is: which restaurants best meet a set of criteria such as good price, wide food selection, and convenient location to the current locations of a set of traveling salespeople? A restaurant is dominated by another if, for example, it has the same location convenience but higher prices and a narrower food selection. They do not specifically address time, but could presumably treat it as another attribute. Their approach develops the database search and algorithm to return the best points. Unlike our research they do not return ranked results, nor do they place the queries within the context of a larger application.

## 4.2 Making Metadata

The scoring-and-ranking approach described in Chapter 3 and earlier in this chapter assumes availability of a suitable collection of dataset summaries against which to apply these formulae. This section describes how to create this metadata for our collection of datasets, using CMOP’s observation archive as our example. We first summarize the challenges in creating metadata for collections of scientific datasets, in Section 4.2.1. In Section 4.2.2 we describe the dataset summaries that we use. Section 4.2.3 addresses how to create hierarchical metadata, while Section 4.2.4 describes how hierarchical metadata is used during a search. We then discuss some practical experiences in Section 4.2.5, and lastly related work in Section 4.2.6.

#### **4.2.1 The Metadata-Creation Challenge**

Metadata creation is an ongoing issue for scientific data collections. One group notes that users want more metadata than providers are interested in providing, and that providers stop providing access to data when more metadata is requested from them [27].

A separate problem is determining what metadata should be collected or provided. There are numerous metadata standards for scientific data, and most standards are constantly evolving. Each field within science has multiple relevant metadata standards, each addressing a different set of concerns or specialties; for example, Green et al. [50] lists the Marine Environmental Data Inventory, the National Biological Information Infrastructure, and National Oceanographic and Atmospheric Administration as all having their own metadata standards, while a webpage at the Marine Metadata Interoperability organization lists 65 standards relevant to their field [150]. In the area of geospatial standards, a 10-year effort led to a book documenting and analyzing the numerous spatial metadata standards in use worldwide [96]. Taking a different approach, the U.S. Federal Geographic Data Committee has a base standard for geospatial metadata that they propose should be further specialized by field-specific profiles, allowing different fields of scientific study to specify what metadata is most meaningful to them [133]. Thus, the question of what metadata to collect or provide does not have a simple answer.

Hill et al. [61] differentiate *contextual metadata*, which is externally provided (e.g., by a scientist), from *inherent metadata*, which can be derived from automated analysis of the

data (e.g., a count of items included in a collection). Automatic metadata generation is ideal, since the manual metadata annotation by scientists required in most systems is considered burdensome and is often ignored, incomplete, or incorrect. However, most metadata standards specify a set of metadata items that can be automatically provided only in part. For example, FGDC Content Standard for Digital Geospatial Metadata lists as required items the abstract, purpose, and citation information, which must (at some level) be provided manually; it also lists as required items that may be amendable to automatic collection, such as the spatial and temporal coverage [133].

It is the capturing and searching of inherent metadata, that is, the information derived from the datasets themselves, that has been our initial focus. Among the main reasons we opted for inherent metadata are uniformity and coverage across repository holdings, the ability to regenerate it as we refine and extend the features we want to capture, and, simply, success in using it. We do, however, use contextual metadata in limited forms, such as the quality level assigned to the data (as can be seen in Figure 3.1).

As our work expands to cover additional archives, the importance of contextual metadata will increase. We believe that contextual metadata items often have different characteristics from the inherent metadata we focus on in this work, including:

- Context is often specific to particular scientific fields; context for ecologists is different from microbiologists, for example. Many of the context items must be manually specified by the scientists (e.g., responsible party, project description). A single context often applies to a collection of datasets, and thus could perhaps be

specified once at a collection level and propagated through to individual datasets.

- Contextual metadata has a greater tendency to be textual; thus, it may be amenable to traditional textual search approaches (project description, scientific method used). A combination of traditional textual search and scientific dataset search may be appropriate.

In future work, we would like to provide a method for an archive to identify a relevant metadata standard, and then identify which fields can be automatically generated from the dataset collection. The system could then request contextual metadata from the archive curator to complete the required data.

#### **4.2.2 Representing A Dataset Collection: Dataset Summaries**

The discussion on our similarity measure above informs us in creating a useful dataset summary. We can summarize a variable in a dataset by the variable name, data type, units (if known) and the bounds of its values; in IR terms, we can consider this information a *feature* for this dataset. For example, we could summarize a column of temperature data with the range of 8.2C to 14.6C as `{"temperature", [8.2:14.6], C, float}`. Similarly, we can simplify spatial data to a geometric feature such as a box or polyline. We can create a dataset summary by capturing such a feature for each variable or column in a dataset, table or spreadsheet, perhaps combined with external information such as file name and file type. We pre-compute this summary for each dataset by performing a one-time scan of the dataset in its original location and format (using a feature extraction

component), and store the summary in our metadata catalog, along with a pointer to the original data.

Figure 3.1 shows an example dataset summary. Each dataset summary contains a unique identifier for the summary; some general information about the dataset, such as its storage location and format; a small amount of contextual information (built automatically using a rule for this archive’s file organization); and information extracted from the dataset contents. This last set of information includes the temporal bounds of the dataset, represented as a minimum and maximum time; the spatial footprint of the dataset, represented by a basic geometry type such as a point, line or polygon; the ranges and units of all variables; and a count of the number of observations in the dataset. The temporal and variable bounds can easily be extracted by scanning the dataset. If geospatial information exists in the dataset, we can extract the geographic bounds. For mobile sensors that follow a path or a series of transects during which the observations are collected (as in our case), a more informative alternative is available; the series of points can be translated into a polyline with each pair of successive points representing a line segment. If appropriate, the polyline can be approximated by a smaller number of line segments. The simplified polyline can be compactly stored as a single geometry and quickly assessed during ranking. We can accommodate alternative representations of dataset components, as long as search terms and the similarity functions are adjusted to match.

### 4.2.3 Hierarchical Metadata

As described in our model, a dataset can be segmented into multiple subsets (or combined with others into supersets), and a separate metadata entry created for each segment. The entries for a dataset and its segments can then be organized into a hierarchy, with the entries classified recursively into parents and children. A parent record's bounds (both temporal and geospatial) includes the union of the bounds of its children. However, the children's regions might not cover all of the parent's, for example, if there are gaps in a time series. The hierarchical relationship of the contained subsets can be captured in the metadata catalog. The number of levels within the hierarchy is not fixed; for instance, we might decompose a cruise temporally by weeks and days within weeks, then segment each day spatially, while a water sample might have only a single hierarchy level.

We give an example using data collected from a mobile sensor. Mobile sensors are deployed in a series of *missions*, each of which may span hours, days or weeks. Observations may be captured many times a second, either continuously or according to some schedule; there may be a half million or more observations per mission.

As is shown in Figure 4.7, the track for a mobile-sensor mission can be represented by a polyline. We extract the polyline from the individual observations (each of which has a time and location) by using the PostGIS *makeline* function to convert each full day's worth of observations into a polyline, then applying the PostGIS implementation of the Douglas-Peucker algorithm, *simplify*, to create a simplified polyline. The simplified

polyline, along with the day's start and end time, is stored as a metadata entry. We then extract each line segment with its time range and store it as a leaf metadata entry, with the day as its parent. We create a “root” metadata entry for the lifetime of the mission, and make the day polylines its children; this root is simply the bounding box of the polylines plus the begin and end times of the overall mission. This three-level hierarchy for mobile sensors can be created quickly, and provides multiple scales of metadata. Where a parent has only one child, we collapse the parent and child into a single entry; the hierarchy tree is not required to have the same number of levels along each possible path nor the same number of children at each level.

A varying number of levels can be used for a subset of the collection or even a subset of sensors within a specific category; we may wish to, for example, add a daily metadata record for specific fixed sensors. In other cases, such as water-sample data, we may

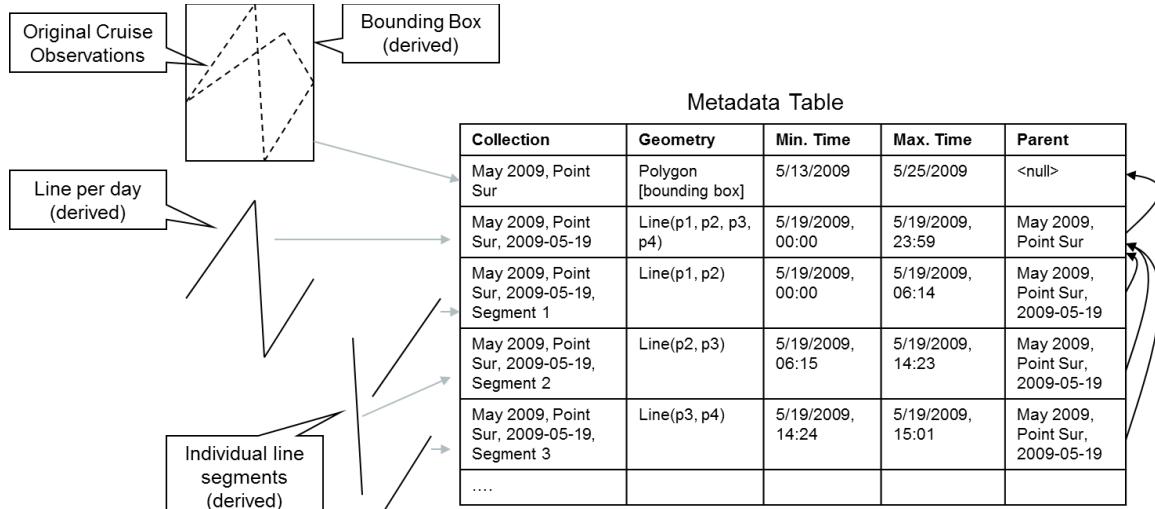


Figure 4.7. Spatial metadata entries for a mobile station (here, a multi-week cruise) is created by generating a line from point observations and simplifying it (middle hierarchy level, on line 2 of the table), then splitting the line into detailed line segments for the leaf records and extracting a bounding box for the parent record. (Note that “Point Sur” refers to a cruise vessel here.)

choose to only have one level in the hierarchy.

#### 4.2.4 Using Hierarchical Metadata

At search time we apply the scoring formula to our collection of dataset summaries to quickly estimate scores for a large number of datasets. The search engine returns datasets or dataset segments from all levels of the hierarchy based on their scores, allowing the “closest” dataset segment to be returned for a search. Thus, subsets and supersets of the same data may be returned for the same search, but at different places in the ranking, with the objective of returning the most useful dataset subset for the current search.

We show via a temporal example how our score can vary across a single hierarchy tree. For this example, we use datasets from a fixed sensor station that reports data only during some months, shown graphically in Figure 4.8. Geospatially, the station’s location is represented by a single point. Its continuous observations are, for convenience, stored in multiple datasets, each containing a single time range such as a month. In this example, three levels of metadata were chosen; an overall “lifetime” record, an intermediate level consisting of a record for the portion in each year that the station reports data, and a detailed level consisting of a single record for each month or partial month. Each light-gray block in Figure 4.8 represents a metadata record, showing its minimum and maximum time.

We show the score for a specific example search term, July:August 2010, next to each metadata record. There are two individual months that are wholly inside the desired time period, and thus score 100 (where 100 is a complete match). Datasets on either side score

in the 90s; the year in which those months occur scores 88, whereas years that do not overlap the search range at all receive negative relevance scores. The overall lifetime record, which overlaps the search at both ends, receives a score of 22. Several outlying leaf datasets receive negative scores. By including these different levels of information in the search results, the scientist can choose between accessing only the months of interest or the entire year (if the access tool allows the datasets to be aggregated).

A temporal search for a time far outside the bounds of a station's lifetime can quickly eliminate this station. Similarly, for a geographic search (say, "near the Astoria-Megler bridge"), a fixed station that is far distant can be recognized and ignored by looking at a single lifetime entry for the station.

During search, we operate over the hierarchy in the following way. We apply the scoring method recursively to the collection of metadata records, starting with the root records. We first retrieve and

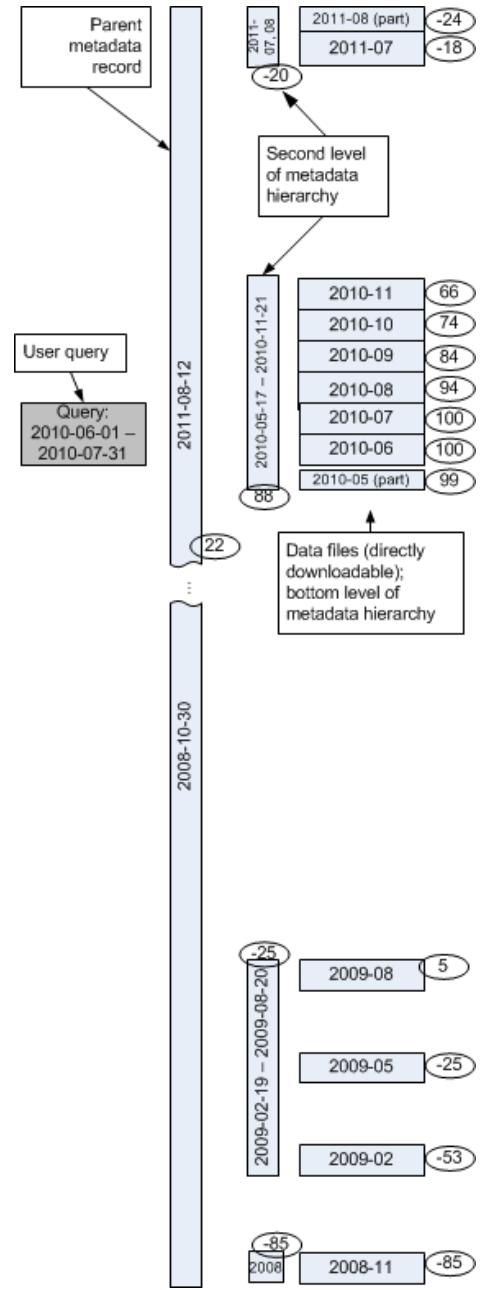


Figure 4.8. Scoring example for intermittent data: the right-most blocks represent downloadable data-sets; the left-hand blocks represent the metadata hierarchy and curation choices (one record per year, plus one for the lifetime). Ovals show the scores given each dataset relative to the search query.

score only the root metadata entries in the collection. We deem an individual entry “interesting” if the minimum geographic and time range distance is not “too far” and if the minimum and maximum scaled time or geographic range distances are different from each other. The second criterion implies that if children of this entry are available, some of these children may be more highly relevant than the parent entry itself. The process recurses until either the list of records to be retrieved is empty or no interesting records have children.

#### **4.2.5 Experiences in Creating Metadata**

How much work is required to create our metadata catalog? Our collection methodology is “semi-curated”, aiming to limit human involvement in metadata gathering as much as possible. In general, the data owner or curator must configure or code certain options once for each new kind of data cataloged.

To set up a new category of data, we must understand the data format, decide the number of hierarchical levels to define and the download granularities to support, and then set up the appropriate scripts to scan the data and create the hierarchies – a process we call “extraction”. Each novel format requires a new extractor. Where the content, volume and meaning warrant separate processing, extraction may be specialized further (for example, mobile versus non-mobile stations, both stored in NetCDF files). We leave the data in its original format and location, and build links to provide direct access to the data from the summary (for example, parameterized URLs for a data-download program). In common with the approach used by Internet search engines in crawling the web, we perform the

maximal possible preprocessing of metadata during extraction, to minimize computation during search. Since metadata creation is infrequent compared to search, extraction-processing speed is not critical.

To keep the metadata catalog up-to-date, we must add new metadata entries whenever a new dataset is created, and update existing entries if a dataset changes. For each category of data, we regularly run a set of scripts and triggers that check for new or updated datasets and execute the predefined steps. The moment a new metadata entry appears in the metadata catalog, it is available to be searched. Generally, we use the same extractor to update entries as to build the initial entries.

How do we partition (or merge) datasets? Currently, the approach is manual at the category level: when a new kind of data is added to the catalog, we consult scientists on what partitioning strategy might make sense for that type of data, and whether an existing partitioning strategy can be reused. Once a partitioning strategy has been decided and coded, it can be applied as broadly as desired. Partitioning choices must in general be made before a dataset is scanned and its features inserted into the metadata catalog.

Currently, deciding what partitioning strategy to use for specific data collections is an art, although we see common patterns emerging that are likely possible to abstract and automate. We note that having subsets of data at multiple levels requires us to pre-compute metadata summaries at all levels for efficiency. However, we generally can collect the metadata for all levels in a single pass of the dataset, so hierarchies do not significantly increase the cost of generating metadata. We have had success in abstracting

and generalizing a partitioning generator for at least one type of data (satellite data from NOAA).

At CMOP, these choices are made once for each major category of data collected; for example, temporary sensors mounted on mobile platforms are partitioned by time and path segmentation, while permanently installed sensors on stationary platforms are partitioned by time only. While partitioning on a variable value is also possible, we have not yet encountered a case where this partitioning was requested by our scientists; however, a smaller time period or geographic area will generally translate to a smaller range for each observed variable. When we added support for autonomous unmanned vehicles (AUVs), for example, we asked if we should treat them the same as existing mobile platforms, or whether there was a reason to segment the data differently. (There was not, and the existing mobile-platform extractor was reused with minor modifications.)

Our first program for each novel type of data takes two or three weeks to program and test. For example, our first mobile-collection-platform extraction program took approximately three weeks to program, and now automatically creates metadata as part of normal data handling of observations collected during additional cruises. Adding the next kind of mobile data, for AUVs, required a few days of additional work due to differences in how that data was stored. Now, additional AUV missions are handled automatically. Handling a third type of mobile observation collection was also quick.

#### **4.2.6 Related Work: Metadata Extraction**

As noted earlier, the challenge of creating metadata over which to search is a known issue for scientific data. Goodchild notes that the number of metadata and catalog formats makes it unlikely that complete automation is possible, but that even partial automation would be a significant step forward [46].

A few existing systems automatically add information such as file-creation date and owner's userid [129], while several infer metadata from sources such as the directory structure within which the dataset is stored [14, 62]. One of these approaches [14] includes geographic metadata created by another application (ArcGIS) used to create the data and stored with their data format, but no search methods are suggested. We could identify only one system that extracts geographic metadata automatically from data [103], but unlike our research, the methods and models are not described.

A few systems extract some metadata from the data itself; one example is inferring likely data types from the data, for use in applications. For example, Google Fusion Tables [44] infers data types such as date, number and geometry on a best-effort basis to decide the type of visualizations to provide; we use techniques such as these when no data type is given by the file format or storage system.

Some search systems address search of data by providing text search over manually provided metadata associated with datasets [43, 103, 109, 129]. In contrast, we search the content of the datasets themselves, as represented by the extracted metadata.

Grossner et al. [51] note that current geographic and temporal search responses provide matches only on one level of metadata; the contents of cataloged digital objects are not exposed and are not searchable. Geographic portals such as Geospatial One-Stop (GOS) [151] and Global Change Master Directory’s Map/Date Search [152] only consider one level of metadata; if a relevant item is embedded within a larger item (Fairbanks within Alaska), the relevant item does not have its own metadata, and thus is not found or returned. In contrast, we address containment by providing multiple levels of metadata; we may return multiple levels of metadata simultaneously (or multiple children at the same level) in response to a single search, with different similarity scores and thus different rankings for each entry.

Few researchers have created hierarchical metadata over data. Vanea et al. [135] describe hierarchical metadata in the context of a data warehouse for pulmonology patient data, with aggregate metadata stored in hierarchical XML nodes. Specific indicators are calculated from underlying data and stored as dimensions in the warehouse, with summaries stored in metadata. Their summaries are specific to particular expected queries, whereas we provide a more general search capability. Rajasekar and Moore [109] describe a layered approach to metadata that differentiates lower layers such as physical characteristics from higher layers, such as scientist-contributed descriptions; their system allows creation of logical collections of datasets associated with a scientist’s research project. Pallickara et al. [103] perform data summarization of large datasets and allow customized combinations of subsets or supersets of existing datasets; however they

appear to support only one level of data, and their search types (subset, superset, intersect and exclusive; over time, over space, or a name-value search) are Boolean and do not support a relevance or distance concept.

### **4.3 Evaluation**

To evaluate our similarity measure, we performed two user studies, described in Chapter 6.

We evaluate our metadata extraction concepts in the following ways:

- We tested the practicality of our metadata extraction ideas by implementing a number of metadata scanners in the prototype, as described in Chapter 5.
- We tested the utility of the metadata for searching by implementing the prototype search engine. We then used the search engine in a user study, described in Chapter 6, and made the search engine available to scientists at CMOP.
- We explored the effect of the hierarchy on performance; we describe the results in Chapter 7.

## 5 Prototype Design and Architecture

In this chapter we describe our prototype design and architecture, which instantiates the concepts and model described previously. Section 5.1 explores the high-level architecture and its implications in more depth. In Section 5.2 we describe our current implementation; in our prototype we have explored some aspects of our ideas in more detail than others. We then perform an architectural evaluation of the implementation in Section 5.3.

### 5.1 High-Level Architecture

At the high level, our architecture follows and adapts the generic search architecture used by most Internet search engines today, shown in Figure 1.1. The adaptation to dataset search is (again) shown in Figure 5.1.

The architecture consists of two sections, Asynchronous Indexing and Interactive Search, each of which communicates with a Metadata Catalog.

The primary processes in Asynchronous Indexing are the following:

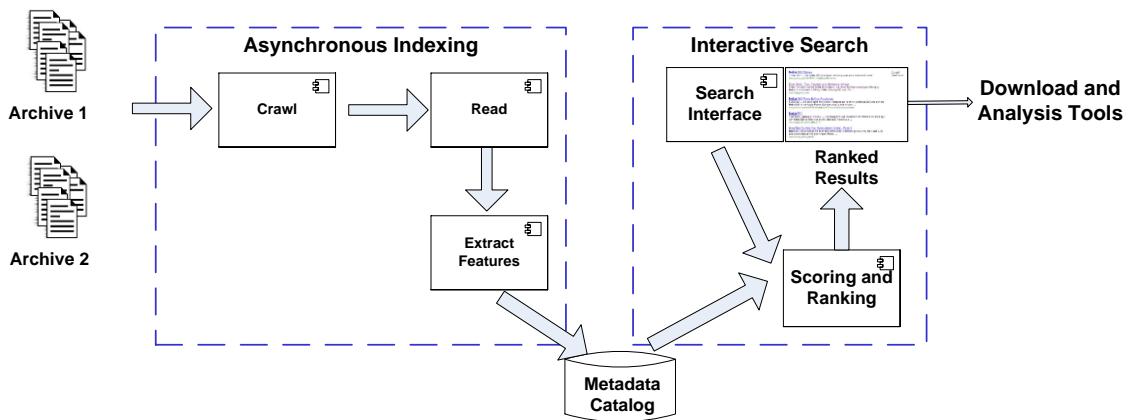


Figure 5.1. High-level dataset search architecture

- The *Crawl* process identifies datasets to summarize. It is given one or more starting points (for example, a link to a THREDDS catalog) and navigates from there, creating a list of datasets for summarization.
- The *Read* process takes each dataset identified by the crawl process and attempts to read it. It may apply rules to identify the correct reader, based on clues such as the file type of a file.
- The *Extract Features* process asynchronously summarizes each dataset read into a set of features and stores the summary in a metadata catalog. The feature extraction process encapsulates the dataset partitioning, hierarchy creation and feature extraction functions (as described in Chapter 3).

The primary processes in Interactive Search are the following:

- Searchers use the *Search Interface* to specify their search criteria. The Search Interface passes the search criteria (the search  $Q$  in our model) to the Scoring-and-Ranking component, and displays the results returned.
- The *Scoring-and-Ranking* component is called by the Search Interface with the user's search criteria. The component accesses the metadata catalog, identifies the highest-ranked summaries for the given search criteria, and returns them to the Search Interface. In terms of our model, this component applies the function  $Sim_s$  to the search criteria and the set of summaries  $S$  stored in the metadata catalog.

The *Metadata Catalog* stores the features, dataset summaries and hierarchy relationships, and makes them available for use by other processes, such as the Scoring-and-Ranking

process. As soon as a feature or summary appears in the catalog, it is available to client processes. The catalog represents the set of dataset summaries  $S$  in our model.

The components are decoupled from each other as much as possible in both the model and the implementation. This general architecture ensures separation and independence between the compute-intensive, asynchronous functions of the system, and the interactive search, which aims to provide interactive response times to searchers. The loosely coupled nature of the components allows maximum flexibility in altering the internal design, the methods used by any component, or the implementation technologies and details without altering the remaining components.

The combined system context and component diagram in Figure 5.2 shows the major components and users of our system, along with external systems with which it interacts. The Search Engine realizes the Scoring-and-Ranking component from Figure 5.1, while the Metadata Creation component encapsulates the Crawl, Read and Extract Features

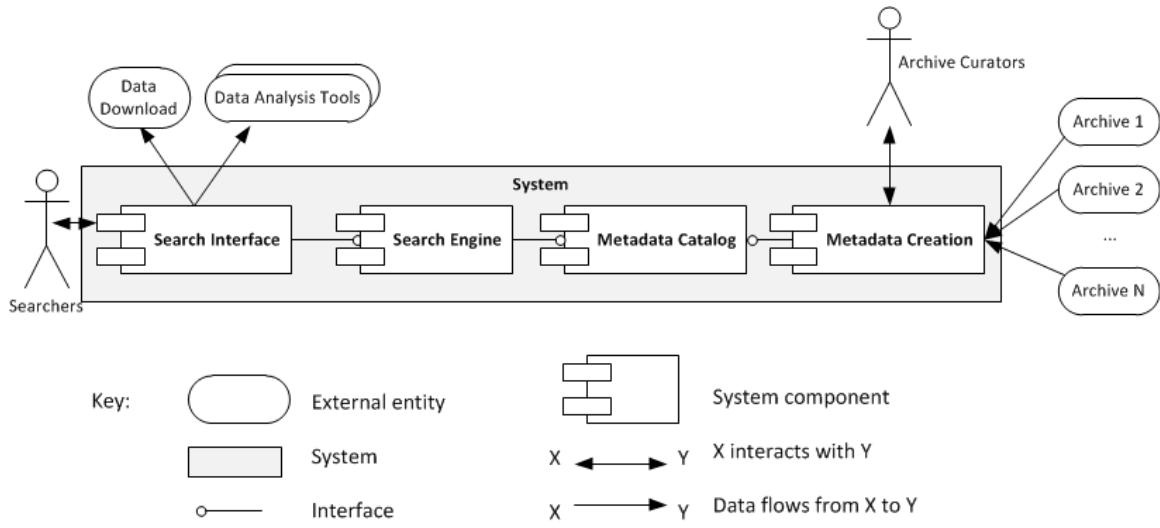


Figure 5.2. System context and component diagram

processes.

There are two primary sets of users of the system: searchers and archive curators.

A searcher interacts with the Search Interface. The Search Interface may include in the list of search results it displays links to directly download the referenced data, or to open the dataset in other tools (where the tool can be opened from a link by providing relevant identifying information).

Each archive may have one or more curators who interact with the metadata-creation process to ensure their archive's data is appropriately represented. Currently, these activities include ensuring that all desired data is being scanned and that the scanning processes are operating correctly. Possible future activities include metadata cleaning and reducing variable- and unit-name diversity, as is described by Megler [90]. In future versions, archive curators may be able to add to or modify information gained via the feature extraction process. For example, a curator might add collection-level information to a set or collection of datasets within the archive, such as contact details for the responsible party or usage restrictions.

The Metadata Creation component encompasses the Crawl, Read and Extract Features processes. Additional curator capabilities, such as those just described, would be supported by this component. The Metadata Creation component interacts with one or more archives, by harvesting metadata from them. The component is made up of a number of individual metadata-creation processes. Each metadata-creation process uses some selection criteria to identify a set of datasets to scan within some archive (for

example: all NetCDF files in a specific OPeNDAP directory tree). Multiple extraction processes may be running simultaneously, with each process scanning a different set of datasets (possibly, but not necessarily, from a different archive). Each dataset is scanned once to perform initial feature extraction. As the datasets may be large and possibly stored remotely from the feature-extraction processing location, this processing can be compute-intensive and time-consuming. A dataset may be rescanned if a change in the dataset is detected, or if the feature-extraction method is modified in a way that requires rescanning.

## **5.2 Current Implementation**

This section describes the design and implementation of our prototype, called “Data Near Here” (DNH). We implemented the prototype at CMOP for use by its research scientists internally; it will be opened to external users after internal validation. Our prototype follows the high-level architecture shown in Section 5.1. This section begins by describing the current architecture in Section 5.2.1, provides detail on Data Near Here’s current data model in Section 5.2.2 and reports on current catalog contents in Section 5.2.4. We describe the search interface in more detail in Section 5.2.5. In Sections 5.2.6 through 5.2.8 we describe plans and extensions for additional types and sources of data, working with variable and unit names, and some of the changes required to move the current prototype to a more robust product.

### **5.2.1 Implementation Architecture**

We implemented the approaches described in this dissertation in a prototype at the Center for Coastal Margin Observation and Prediction (CMOP), described in Section 2.3. We implemented a search engine using the similarity measure described in Chapter 4, operating over a metadata catalog we created.

The implementation is running in production at CMOP. However, the implementation is a research prototype, and is not intended for widespread deployment in its current form. Our goal was to create a usable implementation, sufficient to show the viability of the architecture; conduct user studies and performance testing in support of evaluating our dataset search concepts; and to create a substantial demonstration system based on a real scientific archive.

In choosing our implementation approach, we favored computationally lightweight approaches intended to achieve speed and scalability; we did not emphasize the management, curation or search interfaces. We choose to exploit well-studied and optimized underlying functions, techniques and existing software wherever possible.

The system was designed to be added to the existing CMOP infrastructure without requiring changes to existing components. Figure 5.3 shows the implementation architecture. In keeping with the Information Retrieval concepts and the high-level architecture, the system is composed of four loosely-coupled components that extend the existing observatory repository; these components are analogs of and implement the components in the high-level dataset search architecture. The *Metadata-Creation*

*Component* extracts features from the database or datasets in the observation repository to represent the source observations, and stores these features in the *Metadata Catalog*. The *Search Engine* implements the *Scoring-and-Ranking Component*; it receives search terms from the search interface and interacts with the Metadata Catalog. It scores candidate metadata records, and returns to the search interface a ranked set of records. The scoring-and-ranking algorithm is loosely coupled with the metadata and is independent of the search interface, allowing different algorithms to be easily tested without modifying the other components.

The *Search Interface* is responsible for collecting the search terms from the user and presenting the search results from the Search Engine; it also provides the user with some control over the results presented (e.g., the number of search results to return). The

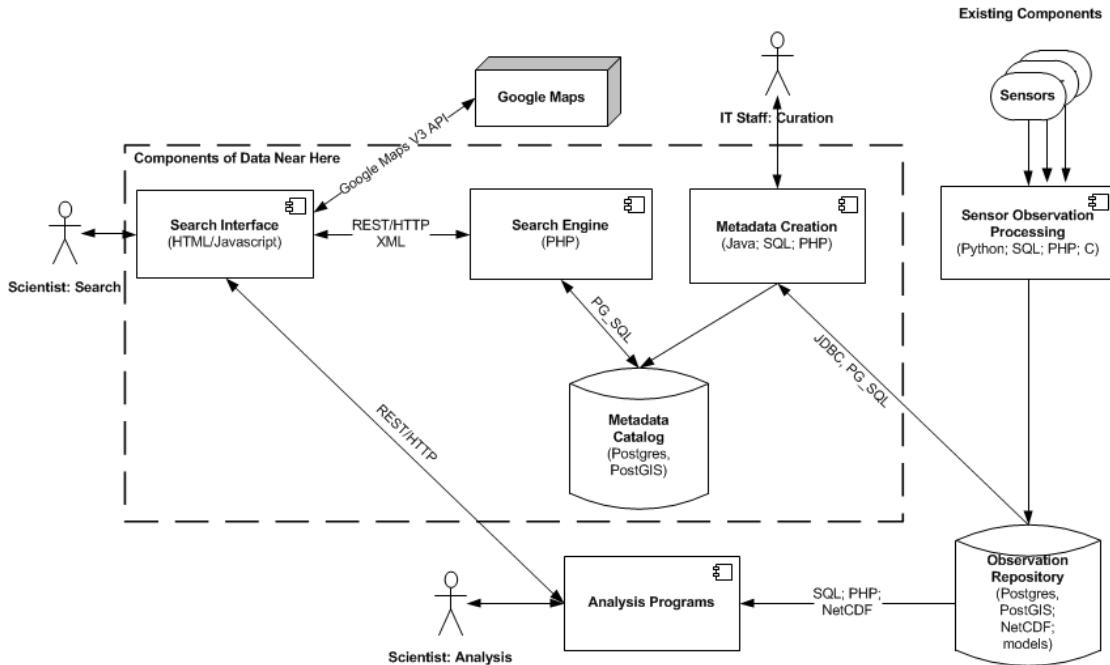


Figure 5.3. “Data Near Here” implementation architecture. The components within the dashed box are the components of Data Near Here that were added into the existing archive architecture.

Search Interface exploits Google Maps [156] for geospatial representation of the search and results. The sole direct interaction between the Search Interface and the Metadata Catalog is when the search interface requests metadata information to populate the search interface’s selections (for example, the “Category” drop-down menu in Figure 5.7). The search results link to the datasets within the repository and optionally to analysis programs.

The loosely coupled nature of the components allows maximum flexibility in altering the internal design or methods used by any component without altering the remaining components; the additive nature of the architecture minimizes changes to the existing infrastructure necessary to add the dataset search capability.

We selected the implementation technologies for the architecture based on existing technologies in use in the CMOP infrastructure, to allow for easy integration, extension and support. In particular:

- The Metadata Catalog is implemented as a separate schema in CMOP’s production PostGIS-Postgres database and is accessed via dynamic SQL. All supporting spatial functions are provided by PostGIS (an open-source software program that adds support for geographic objects to PostgreSQL object-relational database management system [154]), and the geospatial geometries are stored in PostGIS geometry columns. The current data model for the catalog is described below.

- The Search Engine is a monolithic PHP module that is invoked via a REST call, and returns XML. Geometric functions are evaluated by PostGIS during data retrieval from the catalog, with final scoring and ranking performed in the PHP module.
- The Search Interface is implemented using Javascript, JQuery and the Google Maps API. A production version of the Search Interface executes inside CMOP’s Drupal portal. The same code executes outside of Drupal for testing and demonstration purposes.
- Metadata-creation scripts are written in Python 2.6, CMOP’s language of choice for back-end processing. The database is updated via SQL calls; for consistency, we primarily use PostGIS functions called from these scripts for geospatial processing, such as geographic-reference-system transformations, computing convex hulls, and creating polylines to represent a cruise track consisting of millions of point observations.

Current experience leads us to believe these technologies will scale to support the observatory’s repository for some time. The design choices are known to constrain performance; for example, we could achieve faster performance by re-implementing the search engine in a compiled language such as C. However, the current implementation choices provide a useful lower bound on achievable performance, and revisiting these choices is unlikely to lead to multiple-orders-of-magnitude improvement in response time for CMOP. For a much larger observatory, other technology choices would provide

greater speed. The architecture allows us to easily make these choices per component as needed.

In our prototype implementation, we made a number of choices for the purpose of testing these concepts. These choices are assumed in our description and include the following:

- We represent each search term by a tuple of the form  $\langle \text{variable}, \text{range}, \text{units} \rangle$ .
- We summarize the values for each variable into a bounds-based “footprint.” For numeric variables, we use the upper and lower bounds. For text variables, we use the lexicographic minimum and maximum text string found. For (recognizable) geospatial data, we may use a point, a polyline to represent a mobile device’s track, or a convex hull or simple bounding box for more complex data. We represent each variable in a dataset summary as a tuple:

$$\langle \text{variable}, \text{range}, \text{units}, \text{data type}, \text{count} \rangle$$

The range and units combination represent the “footprint” for this variable in this dataset. We include the count on a per-variable basis in order to capture the actual number of observations for each variable in the dataset; for example, a dataset may contain readings from multiple instruments that were operational during different time periods.

- Our feature matching function (*Match*), when matching variables, currently matches on lexicographically equivalent variable names only; a search for “temperature” will not match “air temperature” or “airtemp.” Other search options exposed by the search interface (location and quality, for example) are handled on a case-by-case basis.

- Similarity scoring is primarily based on the distance measure described in Chapter 4.

Figure 5.4 shows the sequence diagram for Data Near Here. The interaction between the components is simple: the Search Interface makes an HTTP GET request to the Search Engine and passes the search parameters in the URL. The Search Engine interacts with the Metadata Catalog (using SQL), and returns the list of datasets to the Search Interface in XML. The Search Interface displays the results.

We performed initial performance characterization of this architecture using a set of arbitrarily selected test searches, prior to implementing any techniques to improve performance. We analyzed the proportion of elapsed time the system spent in the sequence shown in Figure 5.4 with the following conclusions:

**Issue Search Request (Steps 1 and 2):** The Search Interface time spent in preparing the search request (Step 1) and in sending the request to the back end (Step 2) is negligible.

**Search and Rank Results (Step 3):** The search-and-rank-results task in the Search Engine contributes the greatest proportion of the response-time latency and shows the greatest variability across different searches. This task is made up of three steps: understanding the search request; performing the search and identifying the top-k results; and preparing the ranked results for return to the requester. Time taken by the first and last step is negligible. The time taken in the middle task dominates the overall response time, and will continue to do so as the size of the database and the complexity of the searches grow.

**Send Results (Step 4):** The network delay is driven by certain factors outside our control

(distance, connectivity, bandwidth), and by the amount of data transmitted between the search engine and the search interface. We could possibly reduce the data transmitted by compression; however, it is unlikely that such a change would make a dramatic difference to the overall response time.

**Build Results Page and Map (Step 5):** The time spent in Step 5 is primarily driven by the number of elements we draw on the map. We subsequently limited this time by drawing only the top 25 results returned and providing the user with the capability of choosing to draw more results, as a compromise between providing sufficient mapped results and taking too long in the mapping task. In the absence of mapping delay, the primary driver is the number of entries returned by the search. We provide default “minimum and maximum entries returned” guidelines to the back-end, to guide the search-and-rank-results task in the number of the entries desired in the response. The user can request additional items using a “get more” button, if she wishes. The interface

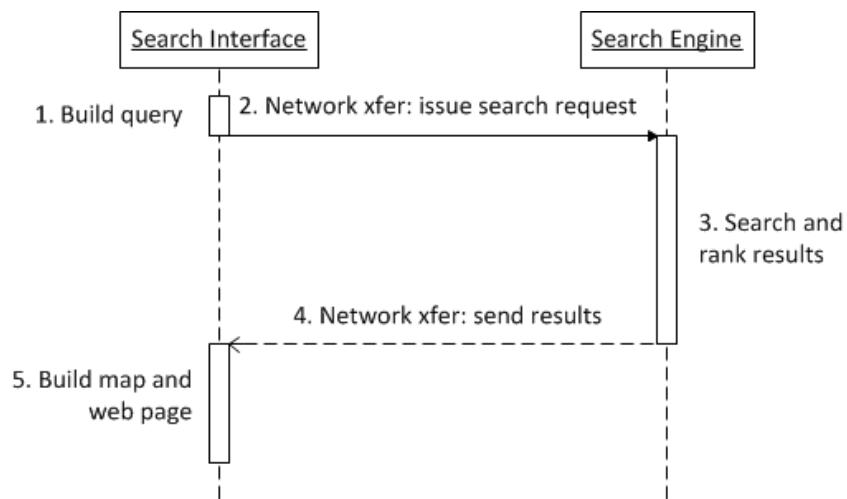


Figure 5.4. Search-execution sequence diagram

currently blocks until a set of results is returned; we researched implementing a non-blocking interface but did not do so as it would require installation of an Apache module not currently installed at any of the demonstration or development sites we use. Alternate search interface designs are possible that would allow data to be displayed with less delay. These UI design alternatives are not the subject of our research.

### **5.2.2 Data Model**

The initial data model was designed using traditional data modeling and normalization approaches. The fundamental object in the catalog is a dataset summary. The summary has a set of global information (*sg* in our search model) associated with it; we store that information in one table. Each dataset summary may have one or more variables ( $sc_1 \dots sc_m$  in the model); we store this information in another table.

We require that each dataset summary in the catalog has a unique identifier. In addition, each summary identifies the name of the owning agency, and the producer (“program\_name”) that added or maintains it.

Some variables have been reified and are reflected at the “summary” level: specifically, time, geographic location and elevation (inverted as “depth”, since CMOP’s scientists are primarily interested in below-surface observations). These variables have special importance to our user community in defining the context of the other observational data in the dataset, and are almost always part of the search criteria. At the time the code was developed, the then-currently used version of spatial tools (PostGIS 1.5) did not fully support three-dimensional spatial functions. As a result, depth is currently treated as a

separate search condition, and the search condition is given the same weight as geospatial location. An alternate approach is to treat the geospatial locations including depth as true three-dimensional locations. The current spatial-distance metric would work with fully three-dimensional data, although some implementation details would need to change.

Splitting variables into a table separate from the dataset-summary-level information supports our simple, consistent abstraction and is convenient for metadata extraction, as we can easily add new field names and variables without changing the data model. During a variable search, we currently “pivot” the variables table on-the-fly during each query that contains a variable search term. We implement the pivot by using the crosstab function in the optional PostgreSQL “tablefunc” module. This pivoting is quite slow; while it is fast enough for DNH’s current metadata catalog, it will limit performance as the catalog continues to grow.

Figure 5.5 shows the current data model for the Metadata Catalog. There are three main tables used for the dataset summaries:

- `metadata_files`. This table contains one record for each dataset summary. This record contains the dataset-level metadata (such as the owning agency, and the collection of which this dataset is a member), along with the reified observational variables. Each summary is identified by a unique id, and contains a reference to its parent if it has one, or `NULL` if it is a root. Each summary also has a count of the number of children (“kids”), that is, the number of summaries that list this summary as their parent.

Currently the primary use of this count is to identify leaves, but there is potential to use it to improve performance.

- `metadata_vars`. This table contains one record for each variable in a dataset. A foreign key links the variable entry to its corresponding `metadata_files` entry. Reified observational variables are repeated in their raw form; for example, a time variable in Unix time will be stored in this table in its source units, but converted to a timestamp and reflected in the `metadata_files` time column-pair. The table differentiates between the “field” and the “variable”; the “field” entry reflects what the column in the source dataset is called, while “variable” allows some renaming to be applied during metadata extract or later (see Megler [90] for additional details on this usage). Due to CMOP preferences that the varmin and varmax values for data with numeric data

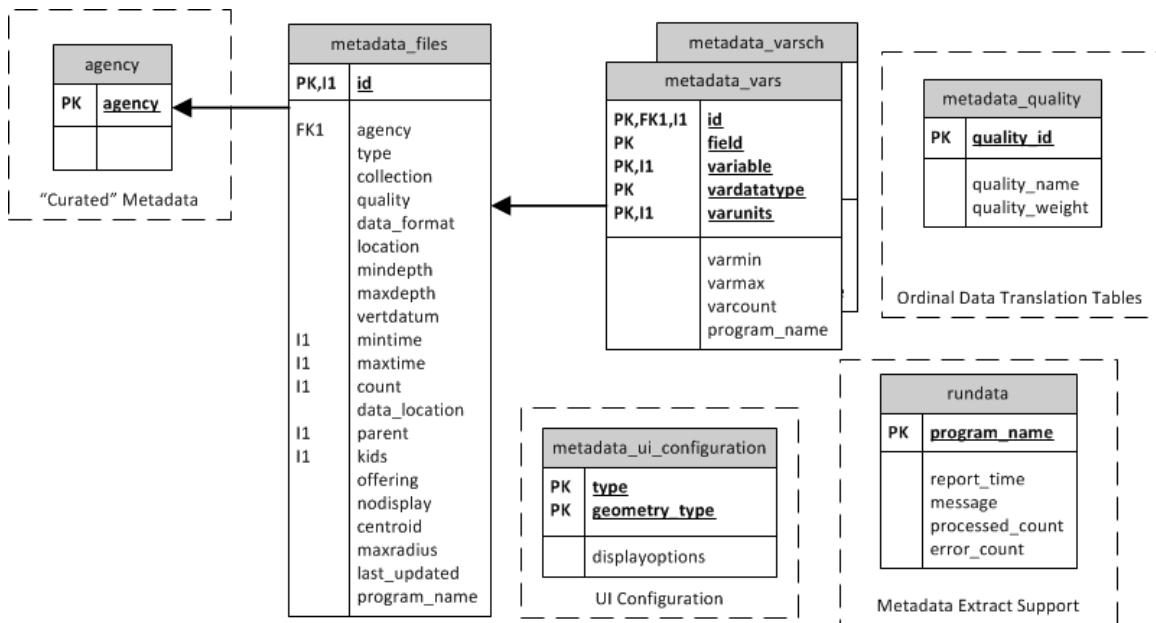


Figure 5.5. Current data model. Dashed boxes show sections of the data model that support experimental extensions.

types should be forced to be numeric (to prevent, for example, a text string being thought of as a boundary value due to an error in the source dataset), these fields are defined as “double precision” fields in PostgreSQL, and non-numeric fields are stored in a separate table.

- `metadata_varsch`. This is a variation on the `metadata_vars` table created for non-numeric variables. The table is identical to `metadata_vars`, except that the `varmin` and `varmax` columns are defined as “text.” The two tables are merged in views that are used by the search engine.

Nascent implementations of extensions and support concepts (“Curated” Metadata, UI Configuration, Metadata Extract Support, Ordinal Data Translation Tables) are shown in Figure 5.5 in dashed boxes, along with the tables we use for the current implementation; we expect that future implementations will flesh out these sections of the data model into their own subsystems. We created several indexes to assist with search engine performance. (The indexes that the majority of the queries in the search engine use are shown in Appendix A). We tested other indexes, including several spatial indexes, but we found that the PostgreSQL optimizer did not choose them for the SQL queries in our code. There is potential for more research here.

### 5.2.3 Making Metadata

An asynchronous, batch-oriented metadata extract suite written in Python updates the metadata catalog. As soon as a metadata entry is inserted or updated, it is available to be returned in response to a search.

We selected metadata summarization functions and implemented them as metadata extract processes. Global metadata for datasets is composed from a combination of sources, including the operating system, curator-provided metadata, and dataset contents. The variable names, units and data type are read from the dataset’s NetCDF [111] header, or extracted from relational-database catalogs. Alternatively, they may be supplied as part of externally provided metadata, for example, from FGDC metadata [133], or even inferred during the summarization process. As described in Chapter 4, for most variables we use the bounds as the summary. Data types are treated the same way; automatic approaches such as those in UCheck [1] or Google Fusion Tables [44, 45] could be applied here to infer likely data types from the data itself, but these methods are not the subject of this research. If the units for a variable cannot be inferred, they are shown in the catalog as “unknown.” In this case, the search engine assumes the variable is in the desired units, but discounts the score to compensate.

We implemented metadata hierarchies such as those described in Chapter 4 for datasets containing point data and stored in NetCDF files, and for both point and mobile (polyline) data stored in a relational database. We implemented adaptive hierarchies, where the number of hierarchy levels varies by observation category and also within a single category or hierarchy; for example, where there is data for only one month in a year, we remove the year level on that branch.

We currently have three extractors running that cover the majority of CMOP’s observational data holdings. One extractor runs against several thousand NetCDF

datasets for fixed-location observing stations, and uses a single time-based policy for defining hierarchies. A second extractor runs against an RDBMS that stores observations from the three kinds of mobile platforms: science cruises, AUVs, and gliders. This extractor uses a mix of temporal and geographic policies for its hierarchies. A third extractor runs against the water-sample collection. This dataset is small but has high value to the scientists, thus making the building of a custom extractor worthwhile. We have an additional three extractors (one for non-CMOP buoys, one for NOAA satellite data, and one for CMOP models) running in various test and development platforms. The initial metadata scanners we wrote hard-coded the knowledge about both the external sources of metadata, and the format of the metadata catalog. In later scanners we began to abstract some of this knowledge into configuration files.

The IT staff can add new categories of observations (e.g., new types of mobile devices), change the number or grouping of hierarchical levels used to represent data, or change the representation of a category of observations (e.g., treating cruises solely as lines rather than as lines and bounding boxes at different levels of the hierarchy); this activity is a *data curation* process [59]. At present, these changes involve writing or modifying scripts; an informal set of patterns is emerging and could be formalized and abstracted to configuration files if desired.

#### **5.2.4 Catalog Contents**

Since we initiated Data Near Here (DNH) automatic feature extraction in April 2012, we have seen steady growth in both datasets and total observations. Figure 5.6 shows the

growth since we implemented DNH, with a ramp-up prior to production release and slower growth since. The sudden jumps in April 2012 and July 2012 are due to the deployment of additional scanners; the remainder of the growth reflects the growth in number of observations from platforms of types handled by existing scanners.

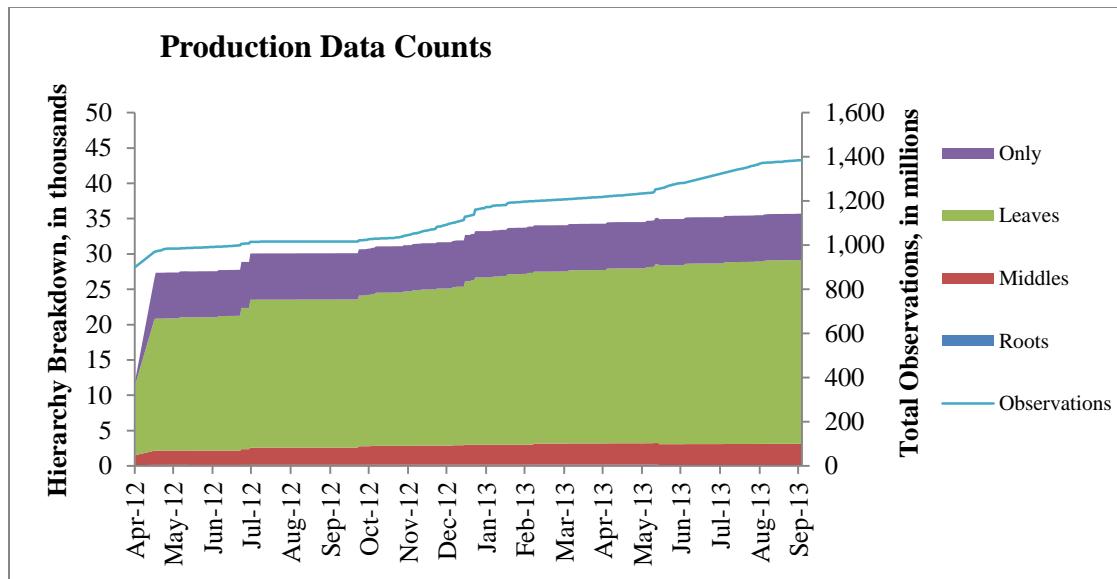


Figure 5.6. Production counts of datasets by hierarchy level and total observations, as of October 1, 2013. The “Roots” line is barely visible, just above the X-axis. “Middles” are records that are inside nodes within a hierarchy, and “leaves” are leaf records in a hierarchy. “Only” represents metadata entries with no parent and no children. The “observations” line shows the total observations represented by these metadata entries, in millions.

As of October 1, 2013, the metadata catalog represented a total of 32.7 thousand datasets, containing a total of 1.38 billion observations. Observations come from fixed stations, cruises, casts, and water samples, split between three quality levels (raw, preliminary and verified). Figure 5.6 also breaks out the different hierarchy levels.

Of the 32,753 datasets, 30,061 contain data collected by CMOP; the second largest organization represented was NOAA and affiliates, and the remainder were collected by

Table 5.1. Characterization of Data Near Here Metadata. This table summarizes characteristics of the metadata entries representing the 1.384 billion observations currently searchable (numbers as of October 3, 2013).

Entity	Count
Metadata entries	36,083
Number of observation categories	7
Records at each hierarchy level:	
Roots without children (Only)	6,554
Roots with children (Roots)	184
Children with children (Middles)	3,146
Children with no children (Leaves)	26,199
Observations represented	1,384,352,599
Average observations per metadata record	38,365

seven other organizations. Table 5.1 gives summary counts for our currently existing metadata entries, representing a subset of CMOP's repository.

The breakdown by category in Table 5.2 highlights the different curation choices made for different observation categories.

At one extreme, the fixed stations have an average of 10.9 million observations each, and

Table 5.2 Characterization of Existing Metadata entries by Category, as of October 3, 2013

Category	Hierarchy Level	Geometry	Number of Records	Number with Children	Total Observations Represented ('000s)	Average Observations per Record
AUV	1 (mission)	Polygon, Line	47	27	452	9,618
	2 (segment)	Line	79	1	297	3,757
Casts-Binned	1	Point	3,030	0	367	121
Casts-Raw	1	Point	2,821	0	39,855	14,128
Cruise	1 (mission)	Polygon	20	20	8,064	403,192
	2 (day)	Line	289	234	8,064	27,902
	3,4 (segment)	Line, Point	8,621	0	7,905	1,426
Fixed Stations	1 (lifetime)	Point	121	121	1,323,420	10,937,352
	2 (year)	Point	893	885	1,323,420	1,481,993
	3 (month)	Point	8,384	1,303	1,323,196	157,823
	4 (dataset per instrument)	Point	8,863	0	1,104,249	124,591
Water Samples	1	Point	681	0	1.7	2
Glider	1 (mission)	Line	18	16	9,335	518,625
	2 (segment)	Line	494	408	13,550	27,429

here we chose to create a four-level hierarchy (lifetime, year, month, dataset) with discussions under way on whether a fifth is warranted (daily).

At the other extreme is the water sample collection, with an average of 2 observations taken per location and time; we represent this data with a single level, that is, with no hierarchy. The same “cast” data is available in two forms: one is the unprocessed, or

Table 5.3. Extract Process: Current Implementations

Categories Processed	Data Volumes	Frequency of Extract Processing	Attribute Characteristics	Hierarchy (from leaf to root)	Source Format
Cruises; AUV; glider	Millions of observations per cruise; months between cruises.	As data is downloaded during or at the end of cruise.	Temporal: weeks Geospatial: hundreds of miles Variables defined by equipment taken on each cruise	Individual observations converted to each leg of cruise (leaves); 1 linestring per day; All days in one cruise, convex hull of total cruise track (root)	SQL
Casts	Millions of observations per cruise; months between cruises.	During cast processing at end of cruise.	Temporal: minutes Geospatial: single spatial location (x,y), but vertical variability (z) Variables defined by equipment installed per cruise	One level; one tree per cast	SQL
Fixed Stations	Tens of thousands every month, for years	Daily	Temporal: taken continuously over years Geospatial: single geospatial location (x,y); equipment may have a fixed or variable vertical location (z) Attributes vary over time depending on equipment currently installed	One dataset per instrument per month (leaves); All datasets per month; All datasets per year; Lifetime of station (root)	NetCDF
Water samples	Approx. 1,500	Weekly	Temporal: instantaneous Geospatial: fixed x,y,z Variables added over time: additional tests may be run long after sample taken, adding new attributes	Samples taken at a single (x,y) point are grouped together, and are simultaneously a leaf and root.	SQL
External fixed stations [prototype]	Hundreds of observations per month	Monthly	Temporal: hours Geospatial: fixed x,y Variables different for each station, and over time.	Each month's data is treated as a dataset (leaf); All datasets per year; Lifetime of station at location (root)	HTML / Text

“raw”, collection of observations; in the other, the same data is binned to specific depths and averaged into a much smaller collection of measurements.

Table 5.2 also shows the variation in geometric representation. For cruises, for example, we commonly use line segments (specific cruise transects) to represent the most detailed level in a hierarchy, but sometimes we use a point to represent a longer period of time when the cruise vessel was anchored in a single location. We can easily discern these different geometric representations programmatically from the data, but they are difficult for a user to identify from the source data without significant effort. For one “cruise” (“Forerunner Daily”) that has operated over the course of a decade, we introduced a fourth level of hierarchy, demonstrating the flexibility of the adaptive hierarchy.

Table 5.3 shows the current observation types we process and their characteristics; we anticipate adding more variations over time.

### **5.2.5 Search Interface Components**

The prototype’s search interface consists of three primary pages: a page with the search interface and search results; a variable-details page; and a dataset-details page. These pages are described in this section. The pages are implemented using Javascript, JQuery and the Google Maps API. Note that search interface design is not the focus of this work. Other search-interface designs are possible and may, in fact, be preferable. The goal of the current interface was to be usable by the scientists, relatively consistent with the current interfaces they use for other tools, easy to develop in the current CMOP environment with the available staff, and usable to test the concepts in this research.

Figure 5.7 shows the tool's search interface and the results of a search, with the top few matching datasets shown. The search interface combines three interacting elements: a set of search-entry fields, a Google map that can be used to specify the geospatial search terms and on which the geospatial positions of highly ranked results are shown, and the search results list.

A scientist can search on time, location and depth [91], and on one or more variables [82] (the variables currently available in DNH are listed in Appendix B). For each added

**Data Near Here V0.6 (Research Edition)**

Please enter the following parameters:

Categories	ALL	Quality	ANY
SW Corner:	6.245381,-124.039 [dec.deg]	NE Corner:	6.309512,-123.944 [dec.deg]
Depth:	from [m]	Depth to: [m]	
Start date:	2010-05-01	End date:	2010-08-31
with variable:	temperature (temp) {cruise_flothr}		
	Range: 5 - 10	Units: c	<a href="#">More</a> <a href="#">Delete</a>
Min. Obs. Count:	1		
<input type="button" value="Get 'em!"/> <a href="#">Click here for Usage Notes</a> <a href="#">Comment</a>			

There were 50 results returned; all are listed, and 25 initially shown on map. Temp was found in 50 entries.

Display	Type	Collection	Quality	Start Time	End Time	From Depth	To Depth	temp	Observations	Data Location	Score	DNH
<input checked="" type="checkbox"/>	cruise_flothr	<a href="#">April 2010, Wecoma, 2010-04-17, Segment 4</a>	preliminary	2010-04-17 04:06 PDT	2010-04-17 04:26 PDT	-5	-5	10.60:10.85 C	21	<a href="#">Download</a>	97	<a href="#">DNH</a>
<input checked="" type="checkbox"/>	cruise_flothr	<a href="#">May-June 2010, Wecoma, 2010-07-16, Segment 3</a>	preliminary	2010-07-16 05:16 PDT	2010-07-16 05:29 PDT	-5	-5	9.89:12.14 C	14	<a href="#">Download</a>	97	<a href="#">DNH</a>
<input checked="" type="checkbox"/>	cruise_flothr	<a href="#">April 2010, Wecoma, 2010-04-17, Segment 11</a>	preliminary	2010-04-17 18:52 PDT	2010-04-17 23:59 PDT	-5	-5	10.88:11.21 C	244	<a href="#">Download</a>	96	<a href="#">DNH</a>
<input checked="" type="checkbox"/>	ctd-casts	<a href="#">July-August 2010, Wecoma, cast041 (Binned)</a>	preliminary_data	2010-08-01 19:03 PDT	2010-08-01 19:16 PDT	-11	-2.5	10.83:11.23 C	20	<a href="#">Download</a>	96	<a href="#">DNH</a>

Figure 5.7. User interface for Data Near Here, showing an example search for a geographic region (shown as a rectangle on the map) and date range, with temperature data in the range 5:10C. Result datasets (or subsets) are shown as points and lines in the map pane, together with their relationship to the search region. In the ranked list of answers, no full matches for the search conditions were found; four partial matches to a search with time, space and a variable with limits are visible, and more are shown on the map. The 'DNH' button reissues the query with modified query terms centered on the values for this dataset.

variable search term, she can select from one of the available variables using a drop-down box that lists the variables found in CMOP datasets. She can request that the variable has values within a specified range, or she can select a variable without specifying a range (called an “existence” search). She can limit the results returned to one or more categories of data (for example, water samples, stations, or cruises), and she can limit the data qualities returned.

We also implemented a prototype of a categorical distance measure across the main quality levels found at CMOP, so that “raw” quality is judged to be further away from “verified” quality than “preliminary” is. Not all search terms must be specified; for

#### Data Near Here V0.6 : Variable Summary for 'oxygen'

	Context	Units	Datatype	Minimum	Maximum	Instances	Observations
+	APL	ml/l	double precision	0	13.60	10	21,078
+	CMOP	mg/l	double precision	-1,297,000	17,318	4,225	33,982,997
-	CMOP	ml/l	double precision	-2.06	16.45	3,218	77,818,671

Agency	Collection	Units	Datatype	Minimum	Maximum
CMOP	Baseline May 2012, Forerunner, cast001 (Binned)	ml/l	double precision	7.38	7.87
CMOP	Grays to Quinault , phoebe, 2011-09-12, Segment 2	ml/l	double precision	0.50	7.14
CMOP	Grays Harbor on 2009-05-17, phoebe, 2009-06-01	ml/l	double precision	0.93	11.80
CMOP	SATURN-01, Oxygen, 2011, August	ml/l	double precision	1.65	7.32
CMOP	Grays to Quinault on 2010-05-12, phoebe, 2010-05-28, Segment 3	ml/l	double precision	2.28	6.36
CMOP	Grays to Quinault , phoebe, 2011-08-01, Segment 6	ml/l	double precision	1.73	7.29
CMOP	SATURN-06, YSI6600V2, 2009, April	ml/l	double precision	7.99	9.18
CMOP	Grays to Quinault on 2010-07-08, phoebe, 2010-07-13, Segment 3	ml/l	double precision	1.85	7.07
CMOP	Grays to Quinault on 2011-09-01, phoebe, 2011-09-10, Segment 5	ml/l	double precision	0.62	6.15
CMOP	Grays to Quinault on 2010-05-12, phoebe, 2010-05-28, Segment 5	ml/l	double precision	2.33	6.31

+	CMOP	unknown	double precision	-9,999	99.00	1,177	3,671,835
+	CMOP	V	double precision	0	5.00	2,269	20,385,132
+	OrCOOS	ml/l	double precision	-1.48	7.25	17	6,984
+	UNKNOWN	ml/l	double precision	-2.07	17.24	10	7,812
-	WDoE	ml/l	double precision	-2.80	47,601,600	79	458,337

Figure 5.8. Variable details page for “oxygen”

example, it is possible to search only on a single variable, or only on time, leaving all other search terms unspecified.

Results are returned within normal interactive response times, that is, within a few seconds. Results are in the form of a ranked list with the highest scoring datasets first; these datasets also shown on the map. The user can access the data directly from this results page via a data download; we provide this capability wherever we can build a link to a tool and specify command-line parameters identifying this subset of data.

While searching for variables, the scientists can request additional data about an individual variable by a small link beside the variable search box. Figure 5.8 shows the details page for the variable “oxygen”. The table at the top lists the various units, minima and maxima, number of datasets (“instances”), and the total number of observations for these units. The second, indented table is expanded from a row in the first table (in the figure, it is an expansion of the first table’s third row), and gives some specific examples of datasets containing oxygen. The “Collection” column gives the dataset description and a link to more details for that dataset.

Such a link leads to a “dataset details” page that shows the dataset metadata. Figure 5.9 shows a dataset in the “middle” hierarchy level, captured by a mobile device. This page can also be accessed directly from the search results in Figure 5.7 via the link in the “Collection” column. Global metadata – metadata that applies to the whole dataset – is shown in the table at the top left. A map displays the simplified track of the device, as calculated from the individual observations.

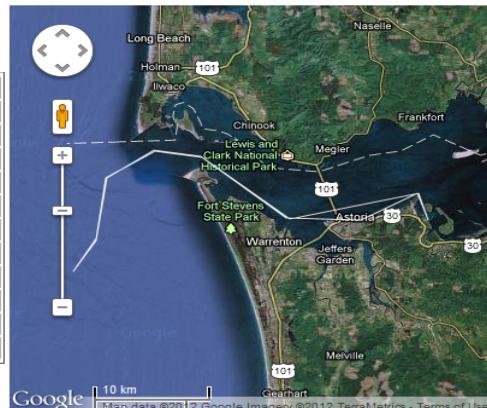
The page also displays each variable in the dataset, its data type and its value range. Links allow navigation to a superset of this data (“Click here for this dataset’s parent”), and to the individual line segments at a more detailed level (links under “Additional information”). In this way, the metadata hierarchy is exposed to the scientist for direct browsing. User feedback tells us that this information and the hierarchy exposed thereby is in itself useful for the scientists in identifying the most relevant subset of data to analyze.

#### Data Near Here V0.5: Dataset Details

##### Dataset Summary

Agency	<a href="#">Center for Coastal Margin Observation and Prediction</a>
Description	Forerunner Daily, Forerunner, 2009-05-28
Type	Cruise
Data Format	CSV
Quality	raw_data
Time: Start	2009-05-28 08:05 PDT
Time: End	2009-05-28 16:05 PDT
Depth: Min	0m (free surface)
Depth: Max	0m (free surface)
# of Values	2,775
Data Location	<a href="#">Download</a>
Last Updated	2011-12-01 08:12 PST

[Click here for this dataset's parent.](#)



##### Variables

Variable	Description	Units	Datatype	Minimum	Maximum	Number
conductivity		unknown	double precision	0	0.32	2,774
salinity		unknown	double precision	0.06	26.54	2,774
temperature	c	double precision		12.23	18.02	2,774

##### Additional Information

This entry has a next level of detail available in the following entries:

- [Forerunner Daily, Forerunner, 2009-05-28, Segment 1](#): with count 446
- [Forerunner Daily, Forerunner, 2009-05-28, Segment 10](#): with count 356
- [Forerunner Daily, Forerunner, 2009-05-28, Segment 11](#): with count 122
- [Forerunner Daily, Forerunner, 2009-05-28, Segment 12](#): with count 02
- [Forerunner Daily, Forerunner, 2009-05-28, Segment 13](#): with count 201
- [Forerunner Daily, Forerunner, 2009-05-28, Segment 14](#): with count 234
- [Forerunner Daily, Forerunner, 2009-05-28, Segment 15](#): with count 212
- [Forerunner Daily, Forerunner, 2009-05-28, Segment 16](#): with count 3
- [Forerunner Daily, Forerunner, 2009-05-28, Segment 2](#): with count 138
- [Forerunner Daily, Forerunner, 2009-05-28, Segment 3](#): with count 127
- [Forerunner Daily, Forerunner, 2009-05-28, Segment 4](#): with count 172
- [Forerunner Daily, Forerunner, 2009-05-28, Segment 5](#): with count 94
- [Forerunner Daily, Forerunner, 2009-05-28, Segment 6](#): with count 117
- [Forerunner Daily, Forerunner, 2009-05-28, Segment 7](#): with count 147
- [Forerunner Daily, Forerunner, 2009-05-28, Segment 8](#): with count 169
- [Forerunner Daily, Forerunner, 2009-05-28, Segment 9](#): with count 161

Figure 5.9. Dataset details page for a middle hierarchy levels of a cruise

The set of data this entry describes can be downloaded directly from this page, using the “download” link. Where possible, we add a link to a plot for each variable using CMOP’s plotting service. The scientist, having identified interesting items in the search results, may wish to use visualization techniques [63, 107, 125] to confirm the relevance; we can take a scientist directly to such tools when they can be invoked via a URL.

### 5.2.6 Plans and Extensions: Data Access

We continue to add to the categories of CMOP data that we are processing. Some categories of data are “like” existing data, or are similar enough to existing categories that we can reuse the same processing routines. For example, while the data from kayak missions is obtained and processed in a different way and is stored in a different location, it is very similar to the cruise data. Thus, the kayak metadata extractor can be a minor modification (to take into account the storage differences) to the cruise extractor.

We have a working extractor for NOAA satellite data. This data is 2-dimensional dense data captured at regular intervals since the 1970s and is pre-processed by NOAA to remove cloud cover and other bad values. We read the satellite data values for CMOP’s region of interest from the NOAA website and generate “leaf” records for each (configurable) geographic “tile”. This tiled data tests the spatial-search approach in different ways from the existing, highly geospatially skewed data. We have extended this extractor with a configurable hierarchy generator, and use this data in our performance testing for the larger volume tests.

We are currently building a metadata extractor for simulation-model data. CMOP develops detailed simulations of the Columbia River and the coastal shelf of Oregon and Washington. Each model run represents multiple simulated environmental variables at each of thousands of locations and at multiple depths, as calculated at fifteen minute intervals. Each model covers a large physical area, and the data is inherently four-dimensional in nature. The data is stored in a binary storage format; a typical model run is around 20GB in size. There are many different model versions being run over the same area and time frame. We subdivide each model run geospatially into a set of “tiles”.

We also experimented with adding metadata to our catalog that represents data at other archives, based on scientist requests. We prototyped extraction from remote sites that are using text formats. We read the data from the source location and create metadata summaries. We add these summaries to our catalog and they are instantly searchable, with no differentiation from our own data. We do not replicate the data; the “data download” links point to the source archive and use their data access methods. In our prototype implementation of a set of observation stations, this approach worked smoothly. We expect to expand this capability, and have a list of requests from the scientists for data they would like to search.

### **5.2.7 Extensions: Variable and Unit Names**

The move into production focused our attention on a problem with inconsistent variable and unit names. The “variables” selection list in the DNH search interface listed 318 unique variables, all of which are currently in use within the CMOP observational

environment. The actual number of distinct environmental variables in the minds of the scientists is much smaller. A brief review identified a number of different issues. In one example, we saw the use of multiple terms for the same variable: “salt”, “salinity”, “salinty” and “water\_salinity” are all used for salinity. A similar problem exists where the spelling of units (as recorded – correct or not) may be different. Automatic adjustment is tricky, even for capitalization variants: “rfu” and “RFU” are the same, but “c” and “C” are not. In another example, data collections may use the same term for different variables: temperature may variously be equivalent to water\_temperature or airtemp, depending on the dataset in question. This problem was recently explored by Megler [90], but remains an open area for future research.

### **5.2.8 Extensions: Moving from Prototype to Product**

As noted earlier, the prototype was developed as a research project, sufficient to test the ideas and to operate in production for CMOP’s scientists. We anticipate a number of components would need to be redesigned and rewritten to formalize the prototype into a product. For example:

- Currently, the search engine is implemented as a single, monolithic PHP program. We expect that rewriting the module in, for example, a compiled language would gain some amount of improvement in performance. Greater improvements might be achieved by using other approaches, such as segmenting the data into multiple databases, processing each segment in parallel and then merging the results.

- We currently pivot the variables tables for each SQL query that involves variables. It would be more efficient to materialize the result since the catalog is updated less frequently than it is queried, or to redesign the data model and related code to avoid pivoting altogether.
- The first scanners we wrote updated the metadata catalog tables directly. More recent scanners build catalog entries in shadow tables, then add or replace the relevant rows in the catalog tables. We believe this approach should be formalized and extended, with a rigorous separation between a “working catalog” and a “production catalog”, and a “publish” process (that can be triggered by both manual and automatic actions) to move entries from one to the other.
- We envision the ability for an archive curator to review the metadata currently being created for her archive and to configure those scanners via an external interface. The curator should be able to request that the archive or any defined subsection be rescanned using a new set of configuration options.
- Methods for more effectively monitoring and tuning the scanning should be added.

### **5.3 Architectural Evaluation**

There are numerous methods available for architectural reviews; a survey by Barcelos and Travassos [12] lists over twenty. They classify the techniques as questioning-based, measuring-technique-based (primarily via simulations), or hybrid methods, and identify ATAM as the most popular method of the hybrid architectural-evaluation techniques. A survey by Mattsson et al. [89] reviewed 240 articles and identified Architecture Tradeoff

Analysis Method (ATAM) [68], developed by Software Engineering Institute at Carnegie Mellon University, as the most mature and broadly applicable method. We therefore selected ATAM for this evaluation.

ATAM is intended as a risk-identification method, and thus operates against architectural design artifacts rather than against the code itself [68]. ATAM “provides insights into how quality goals interact and trade off against each other. It uses both scenario-based analysis and theoretical models of each considered quality attribute to evaluate an architecture” [12]. The steps of ATAM are shown in Figure 5.10. The output of an ATAM is a collection of quality-attribute scenarios, associated risks, and risk themes related to the business goals.

ATAM was updated in a revised method, Analytic Principles and Tools for the Improvement of Architectures (APTiA) [69]. APTiA extends ATAM by adding steps for selecting which subset of scenarios and business goals the stakeholders wish to focus on, and then generating, ranking and selecting from alternatives to the current architecture for those scenarios.

In our case, we presented the business drivers in Chapter 2 as part of our motivation. We describe the current architecture earlier in this chapter. We therefore move directly to step 2, investigation and analysis.

1. Presentation Present the Architecture Tradeoff Analysis Method Present business drivers Present architecture
2. Investigation and Analysis Identify architectural approaches Generate quality attribute utility tree Analyze architectural approaches
3. Testing Brainstorm and prioritize scenarios Analyze architectural approaches
4. Reporting Present results

Figure 5.10. Steps of ATAM (after Kazman [68])

### 5.3.1 Investigation and Analysis

ATAM and APTIA depend on understanding and being able to analyze the effect of the architectural styles or patterns in use in an application. Architectural styles provide a shorthand for describing key characteristics of a system design. *Attribute-based architectural styles* (ABAS) [70] explicitly associate a reasoning framework with each architectural style, to provide a basis for reasoning about the (qualitative or quantitative) characteristics of the system.

Klein et al. [70] provide a set of sample ABAS, covering many common styles. Their Data Indirection style matches our architecture. In this ABAS, coupling is reduced by interposing an intermediary – in our case, the metadata catalog – between the producers (our metadata extractors) and consumers (the search engine). This architectural style is relevant since we anticipate continual change in the producers of metadata, and wish to allow for changes on the client side as well. Figure 5.11 shows the data indirection ABAS with the component names slightly adapted.

In this ABAS, the producers and consumers both know the details of the repository's layout; producers place their data in the repository and consumers retrieve it. In Klein et al.'s words, "modifiability is enhanced by reducing the data or control coupling between distinct components." The ABAS places no restrictions on the run-time configuration of the components. Modifiability parameters for the architecture are as follows:

- Topology: star
- Persistence of data: persistent
- Client knowledge of data: complete knowledge
- Activeness of repository: passive [70]

ATAM then focuses on the notion of *quality-attribute characterization*. Quality attributes are elicited via the definition of scenarios by the stakeholders (which may include the system architect). Each quality attribute is characterized by external stimuli, architectural decisions, and responses.

The quality attributes commonly selected for exploration are: performance, modifiability, availability and security. Data Near Here is a research project and uses the same

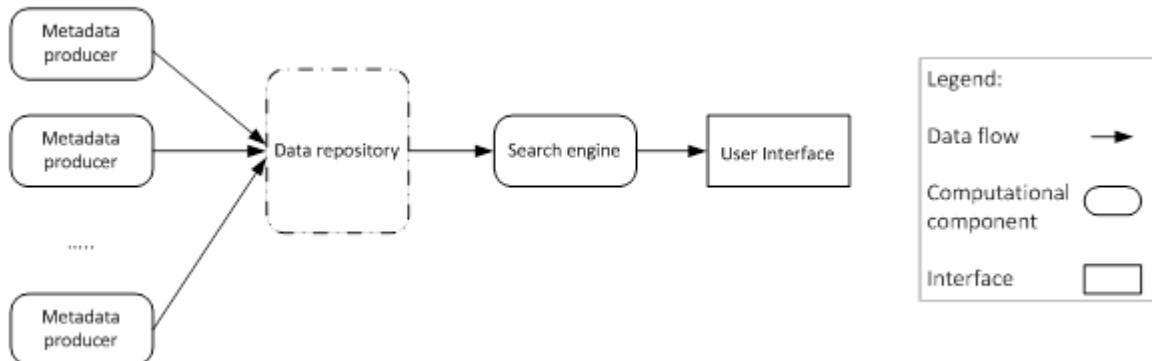


Figure 5.11. Architecture of the data indirection ABAS, as applied to "Data Near Here" (after Klein [68])

infrastructure as the production CMOP systems, with the addition of a few new components that use existing CMOP technologies. The system is subject to the same production requirements as CMOP’s other systems (for example, the database schema is read-only for all but a few select users, who can access the database only from a known range of IP addresses). In the areas of availability and security, no interesting issues were addressed by Data Near Here, and thus these quality attributes were not further explored. Performance under growth scenarios is known to be constrained by the current implementation choices, and is discussed in Chapter 7. Therefore, we focus in this evaluation on the attribute of modifiability.

### 5.3.2 Testing

For the modifiability attribute, Kazman et al. state “the external stimuli are change requests to the systems software, architectural decisions include encapsulation and indirection methods, and the response is measured in terms of the number of affected components, connectors and interfaces and the amount of effort in changing these affected elements” [68].

Change scenarios represent the kind of change requests expected by stakeholders. Kazman et al. subdivide change scenarios into *growth scenarios* (ways in which the architecture is expected to accommodate growth and change in the medium term) and *exploratory scenarios* (extreme forms of growth or change, dramatic new requirements) [69]. Klein describes five scenarios, or stimuli, to consider for this architectural style, all of which we consider to be growth scenarios [70]:

1. Adding a new consumer of data
2. Adding a new producer of an existing data type
3. Adding a new producer of a new data type
4. Changing the internal representation of an existing data item
5. Deleting an existing data type

Here, we interpret the “data repository” to be our metadata catalog. We interpret a “data type” as being a category of data with substantially similar handling characteristics. For example, we would consider the following to be data of an existing type: a new type of mobile device that produces data in an existing data format (e.g., NetCDF) but where the data is stored in a different directory structure and some associated metadata is stored in different place type. However, simulation-model data, although stored in the same data format (NetCDF), would be considered a different data type, as the data represents a 4-dimensional grid, and the grid-node locations are stored separately from the individual observation values.

We add three exploratory scenarios to Klein’s list:

6. Dramatic changes in the contextual metadata associated with datasets. For example, allowing contextual metadata to be added by archive curators, which may require or drive such changes.
7. Dramatic increase in number of users.
8. Increase in the number of catalog entries, by, say, a factor of 100.

We consider each of these eight scenarios in turn.

*Scenario 1. Adding a new consumer of data*, for example, by adding a new search engine.

The new consumer accesses the catalog in the same way as the existing search engine. No change to other search engines or metadata producers is required. The new consumer must be created. Since a search engine performs only read-only access to the database, no additional concurrency issues are created as a result. For example, we have created multiple versions of the search engine to test different performance characteristics, without requiring any modification of the metadata catalog. Since the search engines interact with the data repository via SQL calls handled by the PostgreSQL middleware, new versions can be deployed instantaneously, limited only by the amount of effort required to develop the new search engine.

*Scenario 2. Adding a new producer of an existing data category*. As with adding a new consumer, adding a new metadata producer requires the creation of a single new component with the same constraints as existing producers: that is, detailed knowledge of the relevant subset of the repository data model. We require that the data placed in the repository is consistent with the then-current data model. We require that each producer uses a unique identifier for each catalog entry it adds. In addition, each entry identifies the name of the producer that added or maintains it. The internal design of the new metadata producer is not mandated by our architecture.

For example, we were able to add AUV and glider processing into the system by adding a single producer, without changes to any existing components. In this case, the additional code to handle the new devices was incorporated into the same Python module that

handles data collected from continuous-monitoring instruments during science cruises because of the similarities in the data being processed. In other cases, additional sources are handled with no change to any component: for example, if a new fixed station or additional AUV is added to CMOP’s observation network, the existing components will include data from the new platform automatically as part of their normal processing.

*Scenario 3. Adding a new producer of a new category.* A new component must be developed to handle the new data category. The ABAS notes that this scenario generally requires changes to the data repository and client. However, in many cases, our system as currently designed requires no changes to either. The requirements and constraints for adding a new producer of an existing category carry over to this scenario.

At CMOP, we include the category name in the entry identifier, which helps uniquely identify datasets. We assume that a new category will have a name that uniquely identifies data of that category. We have been able to add new metadata extractors that include external agency buoy-collected (single location, point) data; satellite data from NOAA (2-dimensional gridded ocean surface, polygons); and simulation model data (3-dimensional polygons in a dense grid) without changing the existing system, catalog or other producers in any way. Adding new variable names does not require any changes to the existing schema, as a result of the current data model. Also, the current search processor and search interface will incorporate new variables without change.

*Scenario 4. Changing the internal representation of an existing data item.* In the case where the data being changed is of a single category and the change is to another

category that fits within the existing data model, only the producer(s) of that data must be changed. For example, if a field in a data source was previously processed as a text field and is re-interpreted as a numeric field (as a result of better understanding the source data), the data source can be reprocessed but the results will still fit into the existing data model.

If the data structure itself is changed, it may be possible to create a database view that hides the effect, in which case producers and consumers may remain unaffected. If it is not possible to create such a view, this change may affect all producers and all consumers.

*Scenario 5. Deleting an existing category.* If a category is simply not being produced (or updated) any longer, entries remaining in the repository can still be accessed by consumers. If a category is removed from the repository, then entries of that category will no longer be returned by the search engine. No changes are required to the clients or to other producers.

*Scenario 6. Dramatic changes in the contextual metadata associated with datasets; for example, changes caused by allowing contextual metadata to be added by archive curators.* DNH exposes most of the contextual metadata associated with datasets to the user, and the user can search on those metadata fields. The search interface assumes certain metadata fields have “more meaning” than others; for example, some fields are chosen to be represented in the dataset snippets in the search results, and in the top section of the dataset details pages. In addition, we currently reflect this differentiation in

the data model (by the split between fields in the “files” table and those only represented in the “variables” tables), thus the search engine and the metadata producers are knowledgeable about these fields.

For an archive curator to add contextual metadata (for example: archive-level or collection-level information, such as the group responsible for a set of datasets), we must add new capabilities to the metadata-creation component. These capabilities include: an interface through which the archive curators can review, add and manage the metadata representing their archive; the capability to configure the individual scanners (for example, by specifying which scanner should be used for which subset of their archive, or what rules to use in interpreting the dataset contents); and the ability to specify an external function that can add appropriate facts to each dataset summary. The existing scanners and any new scanners written must have a clear separation between inherent and contextual metadata. It is likely that the added metadata would require changes to the schema, and possibly also to the search engine if the contextual metadata is to be searchable and a schema change has occurred.

Thus, a dramatic change in contextual metadata will affect all components in the architecture.

*Scenario 7. Dramatic increase in number of users.* The users interact only with the search interface, which runs in a browser window. The search interface issues calls to the search engine running under Apache, which in turn calls the database. As searching only reads the catalog, no concurrent-access issues exist except while the metadata catalog is being

updated by the metadata-creation process. The metadata creation process does not change.

As users increase beyond the ability of a single Apache instance and single database instance to handle the load, alternate physical implementations must be explored. For example, it is possible to set up multiple servers to run the search engine and split the user load amongst them using an IP sprayer. If the database engine becomes overloaded and slows down, we can replicate the database instance. Copies of the search engine running under different Apache instances can be configured to direct their requests to different database instances. A mechanism to keep the data synchronized must then be implemented. However, the synchronization mechanism used could be fairly weak; as long as the same client IP is directed to the same database instance, it will see a consistent picture of the catalog contents.

The point at which this limit is reached depends partly on the hardware the system is running on. Apache/PHP and PostgreSQL each use one core during the transaction; increasing the number of cores on the server(s) increases the number of simultaneous searches that can be handled (assuming sufficient memory).

*Scenario 8. Increase catalog entries by a factor of 100.* This scenario has the potential to affect all components in the architecture, in different ways.

**Metadata Producers:** We can increase the number of metadata producers as needed to handle the additional load. Metadata producers generally perform two functions:

- Scanning or re-scanning historical data, which they may perform when the data is first loaded or has been regenerated.
- Scanning incrementally to capture changes since the previous scan.

If the historical data increases by a large factor (for example, a large archive of historical data is being scanned for the first time), it may be appropriate to run a set of scanners, each scanning a subset of the archive. At CMOP, we can specify a set or subset of data to scan for each scanner we have written so far; for example, we can specify a list of directories or cruises.

If the volume of data added by each additional scan is such that running a single scanner is impractical (for example, each run is deemed to take an unacceptably long time), multiple scans can be run, each focusing on a subset of the data.

**Metadata Catalog:** The catalog is currently stored in PostgreSQL. Assuming the current table sizes and hierarchy approaches, an increase in a factor of 100 gives a files table of around 3 million rows and variables tables of around 30 million rows. While PostgreSQL can manage these table sizes, the indexes will no longer fit into the server's available memory and SQL query response times will increase substantially, affecting user response times. Thus, the table should be partitioned prior to reaching that size and the application split over multiple servers. Some of these changes would require changes to the search engines to consolidate results from multiple repositories.

We performed tests, described in Chapter 7, that show that a files table with 1 million rows gives adequate performance for a subset of searches tested. Searches that include

variables will require a different data design due to the cost of pivoting the variables table. The change in data model could be handled by adding a data-structuring step. The change in the data model may also force changes in the metadata producers.

**Search Engine:** The search engine is currently implemented as a single, monolithic PHP module. We can modify the existing code to handle some level of increase in the underlying data through the addition of filters and relaxation, as described in Chapter 7. However, if a large set of rows is returned from the database, the response times may extend beyond what the searcher is willing to tolerate. It is also possible to exhaust the main memory available to the search engine, causing the engine to crash. A major redesign of the search engine will be required to keep search response times acceptable. The **Search Interface** will not change as a result of the increase in number of catalog entries. However, as the number of variables increases (for example, if the added catalog entries contain many different variables), the search interface may require redesign to make the desired variable easier to locate or identify.

### 5.3.3 Results Summary

In analyzing this ABAS, we wished to understand how changes to producers, consumers or the repository ripple through the system. As one would expect with this ABAS, changes to the data repository will ripple through both consumers and producers; however, even here, the design allows for a significant amount of variability in the inputs without requiring any change beyond the component directly handling that input.

Klein et al. point out that the architectural parameter with the greatest impact on the Data Indirection ABAS is the client's level of knowledge of the data schema [70]. Since we are using SQL to access the data, the client's level of knowledge is only partial; underlying details of the exact data model can be (and are) hidden by views, and knowledge of the physical storage is handled by PostgreSQL.

For the majority of change scenarios, our analysis indicates that the resulting changes to the prototype are isolated to a single component, and our experience in making such changes supports the analysis. The greatest challenges to the architecture come from the three exploratory scenarios. We can handle a dramatic increase in the number of users (Scenario 7) via methods external to the prototype code. However, Scenario 8 (increasing catalog entries by a factor of 100) requires a major redesign of the search engine and the data model or addition of steps to the metadata-creation process.

Likewise, Scenario 6 (Dramatic changes in the contextual metadata associated with datasets; for example, allowing external metadata to be added by archive curators) will drive changes to the majority of the components. For this scenario, it is likely that moving to an Abstract Data Repository ABAS, a refinement of the Data Indirection ABAS [70], would be appropriate. This sub-ABAS protects data producers and consumers from changes to the underlying data repository by abstracting the interface to the repository. In addition, formalizing the process for archive curators to manage the entries for their archives will drive additional requirements for the metadata creation process, and will likely require additional components.

This analysis leads us to conclude that the chosen architecture is a good fit for the existing and expected needs for Data Near Here operating over CMOP’s archive, and other archives of similar size and nature. The three exploratory scenarios create challenges for the existing architecture; research discussed elsewhere (specifically, performance research in Chapter 7, variable diversity discussed in Megler [90], and external metadata discussed in Chapter 8) begins to address the challenges identified.

## 6 User Evaluation

Based on our experience, we assert that the IR concept of relevance, IR similarity measures and IR evaluations are all applicable to ranked retrieval of scientific datasets. Without such a concept, the usefulness of a scientific archive declines as the archive grows beyond the ability of an individual scientist to find data relevant to his research interests in it, and the marginal benefit of new or added data declines. In Section 6.1 we give some background on the concept of relevance in Information Retrieval.

As noted in Chapter 2, any proposed set of features and similarity measure must resonate with potential searchers; that is, we must validate that the similarity measure embodies a notion of relevance that resonates with users, and that the proposed search system has utility.

We therefore performed two user studies. In Section 6.2, we demonstrate with our first user study that the concepts of “dataset relevance” and “dataset similarity” are meaningful, implying that Information-Retrieval-style ranked search over scientific data is reasonable. This first study, a paper-based questionnaire, explored the concept of dataset similarity as described in Chapter 4 for temporal, spatial, and joint temporal-spatial conditions, as these features are critical in many areas of scientific research.

In Section 6.3, we present the second user study, which took the form of an operational test of the search tool described in Chapter 5, operating over CMOP’s observational archive. We describe related work in Section 6.4 and discuss the results of the studies in Section 6.5.

## 6.1 Measuring Relevance

The notion of varying levels of relevance to an information need for different items allows those items to be ranked based on those levels. In Information Retrieval systems, we approximate those levels via a similarity measure of each item to the search, and rank returned items based on the scores produced by that similarity measure. Information Retrieval has a well-developed practice of measuring utility of IR systems by comparing user relevance judgments to system-returned document rankings. If our hypothesis that scientific data search is an IR problem holds, then the suitability of IR measures to the resulting approaches should also hold.

One classically evaluates a system that retrieves text documents or web pages by measuring the precision and recall of the system [83].

*Precision* measures the relationship of “true positives” to “true + false positives.” Our approach may return false positives; an example of the potential for false positives is shown in Figure 4.8 in Section 4.2.4. If the search were for data from April 2010, the summary metadata record is a match (with a low score) for the lifetime record because April is within the metadata record’s range, even though no actual data for April was recorded. However, datasets such as May and June 2010 would be given much higher scores. The overall precision, therefore, is driven partly by the size of data gaps tolerated during metadata creation and hierarchy generation, which can be tuned during the curation process. In a geospatial example, we may judge a bounding box for a cruise to be relevant even if the cruise track itself did not pass close to the section of the bounding

box closest to the search. The size of the bounding box would likely cause the entry to be given a low score; individual legs of the cruise may receive higher scores than this entry. As noted by Moffat and Zobel, the widely accepted measure of *mean average precision* and its variants have weaknesses when the number of relevant documents is not known, when multiple queries with different result sizes are grouped, and in the presence of uncertainty in returned results [97]. All of these conditions apply to our study.

*Recall* measures the proportion of relevant items that are returned by the system. It is calculated: the ratio of “true positives” to “true positives + false negatives.” A “false negative” is an item that the system believes is not relevant but the person with the information need judges it as relevant. There has been significant debate about the appropriateness of recall as a measure of IR systems [97, 115, 116].

To accurately measure precision and recall requires that a person individually judges the relevance of every item to a specific information need. We could then compare the judgments to the relevance as scored by the system. For larger catalogs such as that used in our second study, individually judging every item is not practical. Collections such as TREC and INEX reduce the effort of relevance judgment by assuming documents not retrieved by any system in their test are not relevant, thus restricting relevance judgments to documents already judged relevant [32, 139]. TREC’s multiple contributing systems increase the certainty that any possibly relevant document is included, and lends validity to the assumption that all unjudged documents are true negatives. Researchers have shown that relevance judgments are inconsistent across judges; these inconsistencies are

generally avoided by assigning a single judge for each document [54, 115, 116]. Thus, measures of precision and recall for larger collections are always to some extent or another estimates.

## 6.2 User Study 1: Testing the Similarity Measure

Our first user study was designed to test the feasibility of ranked retrieval of scientific datasets, using the similarity approach described in Chapter 4. The initial study focused only on the geospatial and temporal characteristics of observational datasets, two features that are critical for our scientists. In this study, we wished to test a number of hypotheses about our approach:

- Searchers can relate to a brief summary of a dataset, similar to a webpage snippet used in web search.
- There is general agreement about what is considered “closer” across collections of data, at least with respect to time and space, and we can approximate this distance (or similarity) in a simple way.
- Searchers accept joint comparisons of space and time.
- Relative distance is more difficult for users to judge consistently for items at similar distances to the target.
- Geospatially, “closer” makes sense across geometry types, such as points, lines and polygons.

- Scientists and non-domain experts, in general, have similar views on what constitutes “closer”; if so, then the proposed approaches and methods are likely generalizable beyond the scientific community.
- Our candidate distance measure sufficiently captures searchers’ intuitive notion of distance.

### **6.2.1 Methods**

Two populations, each of size 20, of scientists and non-domain experts, were asked (with Human Subject Research Review Committee approval) to respond to a paper questionnaire. The scientists consisted of CMOP professors, post-docs and graduate students; these scientists study spatial and temporal distributions of phenomena or populations. The non-domain experts included professors, graduate students and college-educated professionals, primarily in the field of Information Technology. While accustomed to analytical and problem-solving activities, they do not generally search for large scientific, spatial or temporal datasets.

Drawing on psychophysical ordinal-scaling techniques used in cognitive-distance research [100], the questionnaire contained 60 pair-wise comparisons. Each comparison was between a graphical representation of a search and two datasets represented graphically. Respondents were instructed that, given no other information, they should presume the dataset’s contents were spread equally across the entire spatial and temporal “footprint.” (Such a uniformity assumption is common in dealing with data summaries, such as in database optimization [84].) Respondents were asked which (if either) of the

two datasets (marked A and B) was closer to the search, or whether they considered the two datasets to be equidistant. The questionnaire included comparisons of just the time feature, just space, and combined space and time features; the sizes of the query area and their relative areas were also varied. Some datasets overlapped the search area. The spatial representations included points, lines and polygons; like and unlike shapes were compared. Figure 6.1 shows four examples of the spatial comparisons. We also calculated the result of applying our candidate distance measure (as described in Section 4.1) to the questions, and compared the responses generated by this procedure to the respondents' responses.

### 6.2.2 Results

Figure 6.2 plots, for the spatial components only, the change in respondent agreement against increasing distance between the geometries compared. Two levels of agreement

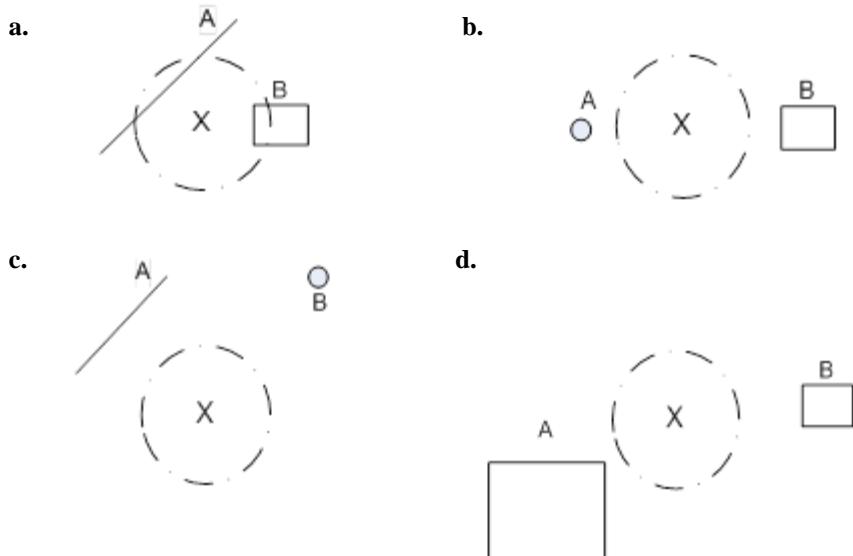


Figure 6.1. Four examples of spatial dataset summary comparisons. The circle marks the area of interest (search). A and B represent the two-dimensional spatial extent of two datasets to be compared to the search circle marked X.

are plotted: the percent of respondents who agreed with the candidate distance measure's assessment of which alternative is closer, as well as that agreement plus the percentage who judged the two options equidistant ("non-disagreement"). While respondents had the option of judging the alternatives equidistant, our distance measure almost always calculates that one is closer, although the difference may be very small. The graph demonstrates that as the difference in distance from the search center to the two shapes becomes small (less than around one-third of the search "radius"), the respondents' level of agreement become inconsistent. In fact, in this range, the respondents often disagree with each other (data not shown), not just with the distance measure. Certain complex configurations or shapes (for example, complex multi-segment lines) increase respondent variability. Plots of time-only and of time-and-space comparisons (not shown) are almost identical to Figure 6.2, despite the difference in search type (time versus space versus

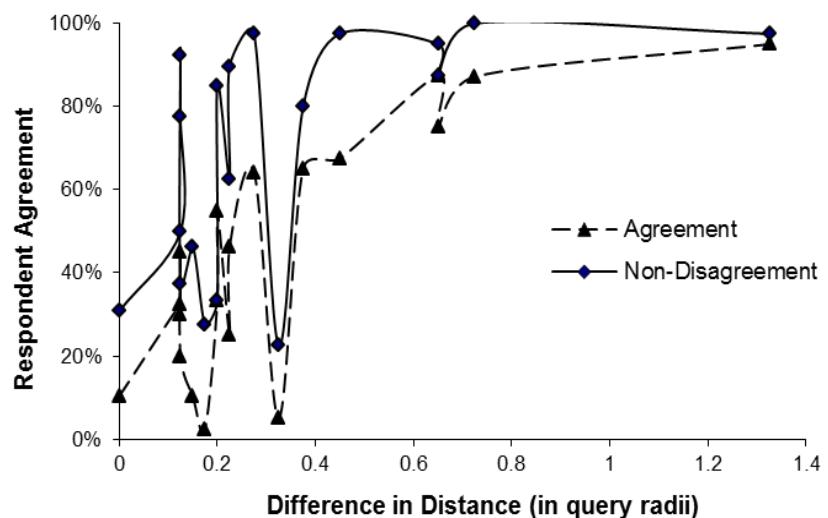


Figure 6.2. Level of respondent agreement with the distance function, plotted against the difference in distance of two spatial-only choices (scaled by search radius).

space plus time), shapes, and dimensionality (one dimension for time only, two for space only, or three for space plus time).

Visual inspection of respondent agreement across graphs (such as that shown in Figure 6.2) of difference in scaled distance between each choice and the search (by the proposed distance measure) revealed a consistent pattern, with strong shifts at approximately .35 and 1 radii difference. In order to statistically test this pattern, we separated the questions into three groups: difference < 0.35, difference > 1, and those between.

We used an ANOVA to test for the variation in agreement with the distance measure amongst the three groups. The ANOVA showed the proportion of “equidistant” responses differed significantly among the three groups,  $F(2, 56) = 20.45, p << 0.001$ , with the variability within the “> 1” group being smaller than in the “< 0.35” group, as expected. In addition, the proportions of inter-resident agreement with the proposed distance measure differed significantly across the three groups,  $F(2, 56) = 30.93, p << 0.001$ , with the level of agreement increasing as the difference in distances increases, as expected.

We present this data in a different way in Figure 6.3, which plots for each question the percent of the population that chose the same option. The horizontal axis represents the percent of respondents who agreed with the distance measure, while the vertical axis represents the highest percentage of inter-resident agreement out of the three options. For example, the left-most point represents a question for which 75% of respondents agreed with each other, but only 5% agreed with the candidate distance measure. Figure 6.1d illustrates this case, where most respondents chose A as being closer to the search

while the proposed distance measure selected B. Points within 0.30 radii of the same distance from the search center (representing the inconsistency seen in Figure 6.2) are removed from Figure 6.3. The remaining scattering of points in the top left represent differences of less than 0.35 (according to our distance measure); with greater differences, we see a high level of agreement amongst most respondents and the distance measure. The lower section, below 33%, is empty since the maximum possible level of disagreement amongst the respondents is when 1/3 of them choose each option.

The study found only one statistically significant difference between the two populations: scientists had a larger standard deviation in their responses to time comparisons. This difference can be explained by scientists' comments that they considered additional factors, such as seasonality, in their assessments of temporal relationships; for example, some regarded September 2002 as "closer" to September 2003 than to July 2002, for certain research questions.

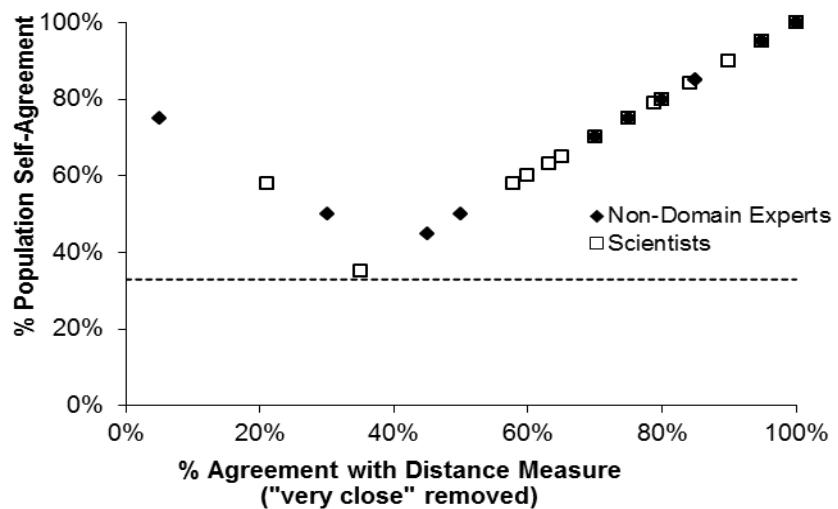


Figure 6.3. Percent agreement with distance measure

### **6.2.3 Discussion**

There were no questions, comments or objections from respondents in either study population with respect to representing dataset contents graphically or as a dataset “footprint”, or with the concepts of dataset closeness to a search or ranking datasets by distance from a search.

From this preliminary study, it appears that our candidate distance measure approximates user expectations of which dataset is judged “nearer” when the difference between them is greater than approximately one-third of the search radius. The consistency in relative ordering agrees with findings in spatial cognition literature [100]. We do not consider the inconsistency seen for nearly equidistant datasets a major issue for our measure; such datasets are likely to appear close to each other in a results list. Note also that we cannot be more consistent with our user population than our user population is with itself (that is, if one third of the users respond with each of the three possible responses – A, B, and equidistant – how do we describe the relative distance of A and B from X?). Many respondents commented on the difficulty in providing what they felt would be consistent judgments across the different questions; despite this concern, the results we received were remarkably consistent outside of the expected ambiguous cases. While the study focused on expected confounding cases and asked few questions comparing choices with widely different distances, the results are statistically significant, supporting the utility of the candidate distance function as a similarity measure.

Opportunities for improvement in the candidate measure exist where the level of overall respondent agreement with the measure is low. Users appeared to weight the dataset edge closest to the search more heavily than the centroid; it appears that adjusting the distance measure to match that weighting could further improve formula-respondent agreement. The optimal weighting could not be determined from this user study; in our second user study, we test a few variations. Identifying optimal weighting remains an opportunity for further research. Other methods of estimating similarity, such as weighting by the data contents (such as by using histograms to compare data distributions), may also be applicable. In all cases, however, the accuracy of any formula in replicating respondent judgments is limited by the amount of agreement amongst the respondents themselves; where the respondents' responses are highly diverse, the formula can at best replicate the most popular response.

Of our study hypotheses, we conclude that there is general agreement about what is considered “closer” to a search with respect to time and space, and that we can approximate this distance in a simple way. We substantiated that “closer” applies across geometry types, such as points, lines and polygons. We confirmed that respondents understand joint comparisons of space and time, and that relative distance is viewed fairly consistently by respondents when the items to be judged are placed at distinct distances to the target search. Our respondents appeared to adjust for the relative sizes of the time and space query portions in their judgments, as does our candidate measure. It appears that respondents can relate to a footprint of a dataset.

We conclude scientists and non-domain experts in general have similar views on what constitutes “closer”, which provides potential to extend this approach to users beyond the core scientific community.

The results of the user study support the hypothesis that a ranking approach based on the concept of dataset distance is feasible. We judged that these results were sufficiently consistent and the candidate distance measure was a sufficiently good approximation to justify implementing these concepts in a prototype search system for further testing.

### **6.3 User Study 2: Fidelity**

The results of the first user study support the hypothesis that a ranking approach based on the concept of dataset distance is feasible.

We performed a second user study using an implementation of this approach in the tool “Data Near Here” to explore two questions. First, is the search tool a useful one? Second, does the scoring and ranking method proposed and implemented provide a good – or “good enough” – approximation of user views of comparative relevance? In the following sections we describe the methods used (Section 6.3.1) and our results (Section 6.3.2).

To address the first question, we asked respondents (described below) questions regarding the overall performance of the system in responding to their information needs, separate from rating dataset relevance. Sanderson reviews studies that show that little agreement exists between IR measures and user satisfaction [114]. Su found that value of search results was more highly correlated with search success than precision [126]. Su’s test population was similar to ours (Ph.D. students and faculty members in scientific

disciplines); if the precision reported by our study was high but user satisfaction low, or vice versa, those results would influence our future research.

To address the second question, we asked respondents to rank the relevance of individual datasets returned in response to a search; these rankings are used to evaluate and compare the performance of search engines. While relevance judgments are known to be inconsistent across judges, a less frequently discussed concern is the gap between the person with the actual information need and the relevance ratings made by assessors for specific documents. The need as interpreted by the assessor may be different from that intended by the person who originally framed the description of the need for use in the study; this gap would obviously influence the relevance judgments made. We address this concern by asking each study respondent to use one of his own information needs as the source for his searches, and asked him to judge the relevance of the returned items for his own searches. While we might lose some theoretical repeatability (although it does not appear that repeatability has been proven in text retrieval [116, 123]), we gain insight into the applicability of the approach and implementation with this rating scheme.

### **6.3.1 Methods**

Our second study followed a common IR approach, adapted appropriately for datasets. With Human Subject Research Review Committee approval, the study used a convenience sample of 12 scientists. These scientists were professors, post-docs and graduate-level students, all intended future users of the tool at CMOP and existing users of CMOP's data archive. None of the scientists had previously used the tool.

Table 6.1. Count of Test Collection Catalog Entries, Represented Observations and Geometry

<b>Observation Class</b>	<b>Geometry</b>	<b>Count of Entries</b>	<b>Total Observations</b>
Stationary platform	Point	14,648	744,174,016
Stationary, variable depth	Point	6,677	42,850,403
Mobile, fixed depth	Point, line, polygon	7,938	3,922,736
Mobile, variable depth	Point, line, polygon	1,161	6,982,008
Totals		30,424	797,929,163

We used as our test collection the catalog of datasets constructed from CMOP's observational archive, as described in Chapters 4 and 5; thus the test collection was identical to the system's production catalog. We asked that the searches be of certain forms (see below). Table 6.1 summarizes the major types of data within the collection, the count of catalog entries and observations they represent, and the geometries used to summarize each type of data.

The search tool's search interface was modified for the user study by adding features to administer survey and rating questions and capture the responses. The results page was modified to return exactly the top 100 results, if necessary including results judged by the system to have low relevance. The searches and survey responses were captured using Google Analytics. No data was captured that linked a respondent to his or her responses or searches.

The study procedure is summarized in Figure 6.4. Each respondent was given a ten-minute tour of tool operations; the same information was provided as an appendix to the survey instrument. The instructions then asked her to think of a recent information need for data supporting her research, and to perform three or more searches. In order to

collect a range of searches we asked for (at least) three different combinations of conditions: one search using only one or more of location, time and (if desired) elevation constraints; a second search adding a variable-existence condition; and the third search adding constraints on the values of the variable (minimum and maximum values, in some units). In order to capture searches representative of real operations, no restrictions were specified on the kind of information need, locations, times, or variables to be used. The respondent was asked to review the results returned for the search.

To measure tool utility, we asked five questions for each search, shown in Table 6.2. Our questions were adapted from Su [126] and represent the major categories of search success (Question 1), utility (Questions 2, 4 and 5), efficiency (Questions 4 and 5), and user satisfaction (Question 3). The answers were rated on a 7-point Likert scale, with 7 = excellent and 1 = not at all.

After the scientist answered the five questions, we presented her with a subset of 25 of the 100 results. We include in the list of 25 the top 10 results returned, the lowest three in the list, and 12 randomly chosen items. These items were chosen to ensure that we could

- 
1. Respondent is given a brief overview of tool usage, and is given the opportunity to familiarize himself with the tool, if desired.
  2. Respondent considers a recent information need. Respondent formulates the need as a set of 3 or more searches. The set should contain at least one of each of the following types of search: a spatial or temporal search (or both); a variable-existence search; a search containing variable limits.
  3. For each search:
    - a. Respondent enters search conditions.
    - b. System retrieves and presents a ranked list of 100 items. Respondent briefly reviews results, then proceeds to “survey” step.
    - c. System presents the 5 qualitative questions. Respondent rates questions using Likert scale.
    - d. Systems presents 25 datasets selected from the results, in random order. Respondent rates each dataset on a 4-point Likert scale from “not relevant” to “relevant”.
- 

Figure 6.4. Second user-study process

Table 6.2. Responses to User Satisfaction Questions and Comparison of High vs. Low Scores:  
All Searches and by Type (NM = Not Meaningful)

Median [Interquartile Range]  High versus Low Scores: z-score (probability)	All (n=30)	Space + Time (n=8)	Variable Existence (n=13)	Variable with Limits (n=9)
	6.5 [0.5]	6.5 [0.5]	7 [1]	6 [2]
1. How successful was this search in helping with your information need? [success]	6.5 [0.5]	6.5 [0.5]	7 [1]	6 [2]
	2.79 (<0.01)	NM	2.05 ( 0.03)	1.04 (0.16)
2. How well does this style of query allow you to express your information need? [qryexpr]	6 [1.0]	6 [0.25]	6 [0]	6 [1]
	3.74 (<0.01)	NM	NM	1.84 (0.05)
3. How confident are you in the completeness of search results? [confcamp]	6.5 [2.5]	6 [1.25]	7 [3]	6 [4]
	2.02 (0.03)	NM	1.40 (0.09)	0.61 (0.28)
4. Was using this tool quicker than finding the most relevant results by other means? [quicker]	6.5 [0.5]	6.5 [0.75]	7 [1]	6 [0]
	NM	NM	NM	NM
5. How valuable are the search results versus time expended? [time/effort]	7 [1.0]	7 [1.25]	7 [1]	7 [1]
	2.98 (<0.01)	NM	1.78 (0.05)	NM

report traditional “at ten” IR precision measures; they also ensured variety in the items presented for rating, in the absence of a large collection of pre-existing ratings. We removed the dataset score and position on the original list, ordered the 25 items randomly, and renumbered the items. We asked the scientist to rate the relevance of each result to her search. We used a four-point scale (3 = high relevance, 0 = no relevance ) adapted from Sormunen [123]. Our focus was on the search behavior, as the archive is already known to fulfill only a subset of scientist information needs. Our analog to topic relevance is the applicability of the dataset’s contents to the searcher’s search. Our

chosen analog of Sormunen’s “degree of topical relevance (the extent to which the text discusses the topic)” is the proportion of a dataset’s contents that the user believes is directly relevant to the search. These choices allow independence of relevance from dataset size while providing the same intent as topic coverage; a small dataset of highly relevant observations may be more useful than a large dataset with few relevant observations.

### **6.3.2 Results**

The 12 scientists returned 35 responses during the study period. Of these, five were tests submitted with no ratings, leaving 30 responses usable for dataset ratings. We report results separately for the qualitative questions and for the ratings results.

#### **6.3.2.1 Results for Qualitative Questions**

In order to better understand differences between the different types of searches we present the overall results, then break out the searches by type: geospatial-temporal only, searches with variable existence, and lastly searches with limits on variables. Figure 6.5 shows the results graphically; Table 6.2 presents the median and interquartile range for each question, for all searches and by search type; Figure 6.5 shows the range, median and outliers graphically. The median for each question is 6 (very good) or better. With the exception of Question 3, “confidence in completeness”, answers were clustered fairly closely about the mean.

To assess the overall utility of the tool, we compared the proportion of high scores ( $> 4$ ) to low scores ( $< 4$ ) for each question, using a two-sample test for the differences in

proportions. Results are shown in Table 6.2. The high-satisfaction responses to the overall-success question were statistically significant in all cases. Separating out by search type, the geospatial-temporal and variable-existence searches had statistically meaningful high responses. We could not calculate the z-statistic for a number of the search subsets or for the “quicker” question as there were no low scores in the responses for these sets (shown as “NM”, not meaningful). In all combinations but three, the high satisfaction responses were statistically meaningful. The exceptions are all in the variable-with-limits searches. For these questions, the median response was highly positive, but the variance high. Responses for overall search success were highly correlated with the other responses (Pearson’s r for correlation of overall search success with search expression, 0.72; with confidence in completeness, 0.85; with quicker, 0.98; with time versus effort 0.95; n = 30, p<0.0001 in all cases).

### **6.3.2.2 Ratings Results**

Of the 30 usable responses, two judged relevance for three or fewer datasets. These two responses were excluded from the search-level analysis, leaving 28 usable searches with associated dataset judgments. The mean number of judgments for these remaining searches was 24.5, as a few datasets were not rated.

For eight searches, all four values (0-3) were assigned to datasets; an additional seven searches assigned three values, and six searches assigned only two values. In seven searches all datasets were given the same rating. In six of these cases, all datasets were rated as highly relevant; five of these six were variable-existence searches. In one case,

all were rated as not relevant. Not surprisingly, the searches with “highly relevant” (value of 3) assigned to all datasets were associated with high satisfaction measures, whereas the sole search with “not relevant” (value of 0) assigned to all datasets was associated with the lowest satisfaction measures in the study. Even for this search, however, the “quicker” and “query expression” scores were high, signifying that even when no relevant data is found, the fact that this situation can be ascertained quickly is likely to be of value. This experience is consistent with Su’s findings [126].

As our relevance data is similar to that collected for IR studies such as TREC and INEX, we believe using IR metrics is justified.

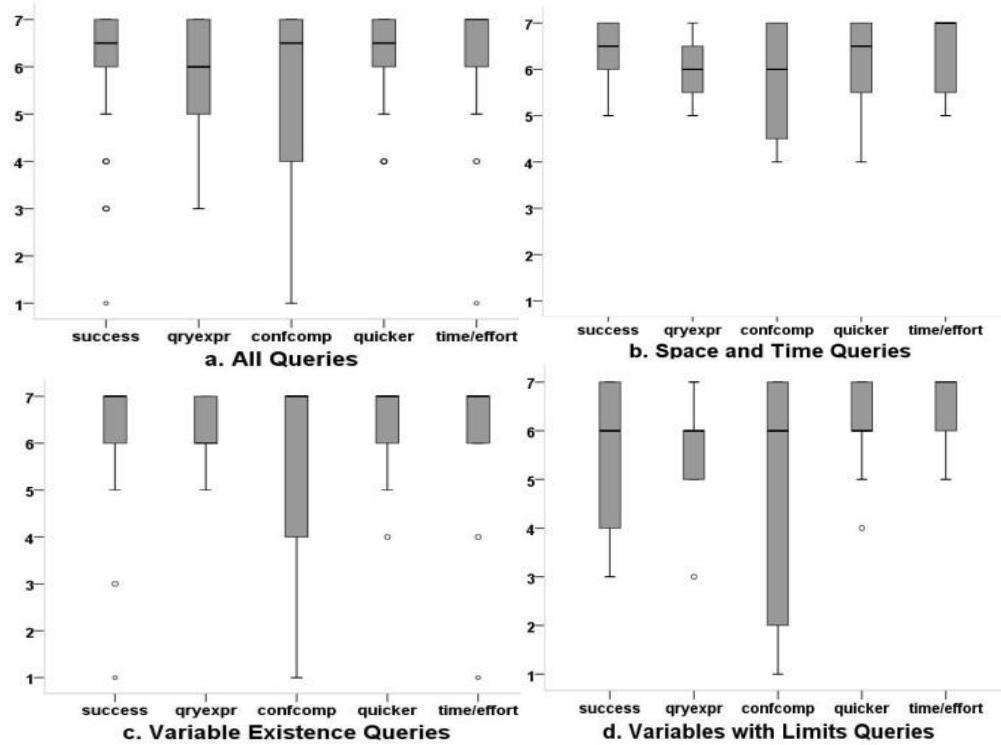


Figure 6.5. Summary results for user-satisfaction survey questions. The questions are shown in Table 6.2.

Table 6.3. Precision and Mean Reciprocal Rank (MRR) by Query Type and Relevance Judgment

	All (n=28)	Space/Time (n=8)	Variable Existence (n=12)	Variable with Limits (n=8)
P@10	0.96	1.00	0.91	1.00
2+3@10	0.82	0.96	0.74	0.81
3@10	0.55	0.69	0.58	0.38
MRR	0.95	1.00	0.88	1.00
MRR2+3	0.86	0.92	0.78	0.92
MRR3	0.72	0.76	0.73	0.67

In Table 6.3 we report precision measured at rank 10 (P@10) and mean reciprocal rank (MRR). Precision at rank 10 gives a measure of the number of relevant documents found in the top ten returned; MRR measures the average position of the first relevant document found [139]. We include in these measures all datasets judged to have any relevance; this choice is in line with the threshold of relevance used in binary evaluations [139]. We report measures for all relevance levels together (P@10, MRR). Overall mean precision at rank 10 was 0.96. Overall MRR was 0.95, and was 1.0 for two search types. In one variable-existence search all datasets were found to be not relevant; no dataset from any other top 10 was rated not relevant.

We also report separately precision and MRR for the combination of the “medium” (2) and “high” (3) relevance ratings (denoted as 2+3@10 and MRR2+3); likewise, we report separately precision at 10 and MRR for the “highly relevant” (3) ratings only (denoted as 3@10 and MRR3). As before, we report these measures for the full set of searches, then for each search type separately. Even excluding low-relevance datasets, precision at rank ten and MRR remain respectable, but highlight areas for possible exploration and improvement. Analysis of ratings below position 10 is presented below.

Recall is defined as the fraction of relevant items retrieved, and relies on knowing the total number of items relevant to the search in the archive. In our study, each respondent developed their own searches; with an archive of around 30,400 items and no constraints on a user's searches, it was not practical to identify all relevant items to each search in order to calculate recall. Collections such as TREC and INEX reduce the effort of relevance judgment by assuming documents not retrieved by any system in their test are not relevant, thus overstating recall [32, 139]. In the absence of multiple systems to provide alternate sets of documents, we approximate this approach by including items originally ranked 98-100 in the returned list, below the presumed attention span of the user, in the relevance-judgment subset. In Figure 6.6 we compare the ratings given to top 10 ranked datasets versus "bottom 3" ranked datasets (positions 98-100). The percentage of top 10 datasets rated as "highly relevant" is significantly higher than the percentage of "highly relevant" in the bottom 3 ( $z = 4.63$ ,  $p << 0.001$ ), despite several searches where all items received the same rating; thus, we posit that few false negatives exist and that recall, while not directly quantifiable, is likely to be high.

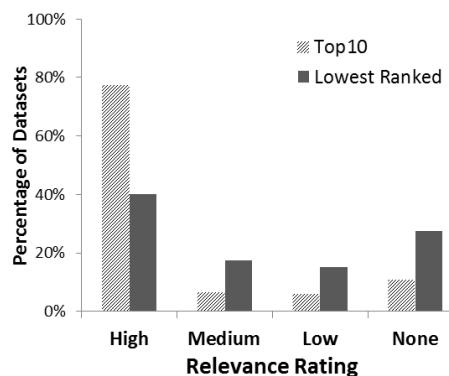


Figure 6.6. Proportion of datasets by rating

Taking all judged-relevant top 10 datasets as the true positives and all judged-relevant bottom-3 datasets as false negatives, we calculate an estimated recall figure of 0.821. However, even datasets in the bottom 3 may be true positives; for example, in the six searches in our study where all datasets were rated as having some (same) level of relevance. Thus, this recall figure probably understates the actual recall of our system.

#### **6.3.2.3 Dataset-Level Results**

A total of 685 datasets was rated. Of these, 351 (51%) were rated as highly relevant; 147 (21%) were rated medium; 118 (17%) were rated low, and 69 (10%) were rated as not relevant. Of the 685 datasets rated, 90 were rated more than once, with 26 of these rated three times. Of the datasets rated more than once, 53 received the same rating each time, while 37 received different ratings. In each of the 53 same-rating cases, the ratings came from the same respondent in the same search set; for example, a respondent rated a dataset as highly relevant for a location and time-based search, then added a variable to the search conditions and found the same dataset highly relevant when it was returned for the modified search. Eight datasets of the 37 were rated from “highly relevant” to “not relevant” across a set of searches; in each case, the different ratings came from a different search set (hence a different respondent). The original position in the returned list for any single dataset varied from 3 to 97 due to the differences in the search for which it was returned.

We saw no significant difference in the proportion of different ratings of datasets representing different geometries types (as listed in Table 6.1), that is, datasets

represented by points versus lines or polygons. Nor did we see significant differences in user responses between datasets from the lowest level of the hierarchy versus datasets from higher in a hierarchy. We conclude that summarizing diverse geographic locations by a geometry and representing subsets as though they were individual datasets themselves is well-accepted by our users.

We further explored rankings within each search and potential variations of our scoring formula.

We used two methods to explore rankings within each search. First, we applied a compressed version of Discounted Cumulative Gain (DCG) [67]. We have relevance judgments for rank positions one through 10, but we only have relevance judgments for 15 of the datasets in rank positions 11 through 100. Therefore, we condensed the results and treat the judged datasets as though they had been returned in positions 1 through 25, omitting the non-judged datasets. We compare the order of datasets returned with an ideal order for the rated datasets, with all highest-rated datasets returned first, followed by all medium, and so forth. In absolute terms, this approach gives arguable results, though without rating all intervening datasets, it is not clear in which direction the results will be slanted. However, since our primary interest is in exploring modifications to the current scoring formula in order to improve ranking results, including and discounting the unjudged items would reduce the differentiation between the reported curves without adding any counterbalancing diagnostic capability. Figure 6.7 shows results for all

searches and for variables with limits; the plots for space plus time and for variable existence are visually identical to Figure 6.7(a).

We tested the current distance measure against five variations, using the datasets judged by the study participants. (Our approach is in common with TREC approaches that evaluate alternate measures against previously judged documents, rather than the entire corpus.) The candidate measure, as described in Chapter 4, uses the center of a variable's range as the point to which distance is calculated for one-dimensional variables; for two- (or more) dimensional variables, the measure uses the average of the distance to the closest (mindist) and farthest (maxdist) points. The variations evaluated different weightings of the closest edge versus the center, per indications from the first user study and supporting informally expressed opinions by some study participants. In particular, we evaluated:

SN: mindist

S2:  $\text{mindist} * 0.875 + \text{maxdist} * 0.125$

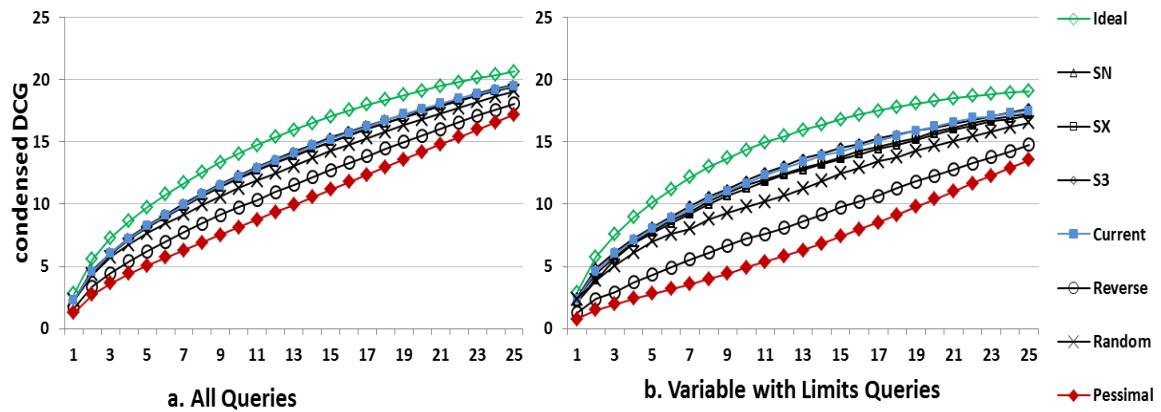


Figure 6.7. Condensed discounted cumulative gain (a) for all searches; (b) for variable-with-limits searches only.

S3:  $\text{mindist} * 0.75 + \text{maxdist} * 0.25$

S4:  $\text{mindist} * 0.625 + \text{maxdist} * 0.375$

Current:  $\text{mindist} * 0.5 + \text{maxdist} * 0.5$

SX:  $\text{maxdist}$

Based on the insight from the first user study that the closest edge should be a little more heavily weighted, we expected S2 or S3 to perform the best. In Figure 6.7, we plot the results from our current scoring formula and three variations: SN, S2 and S3. To contrast our measures with other orderings, we included four controls: the ideal (optimal) and “pessimal” (reverse of the optimal) curves, a randomized ordering (RAN), and an inverse ordering (REV) of the current ratings. Any possible performance curve will be bounded by the ideal and pessimal curves. As can be seen in Figure 6.7, there is insufficient differentiation to discern relative performance of the current and alternative scorings, as they overlay each other. We applied a one-way ANOVA to the condensed DCG at rank 25 against these and two additional alternatives. The results implied that alternatives S2 and S3 might perform 10-20% better than our current approach, but the results were not statistically significant under a Tukey’s HSD. The randomized order returned a mean score of around 0 and the reverse scoring returned a negative mean score, as expected. Secondly, we applied Rank Biased Precision (RBP) [97] with the extensions for non-binary relevance judgments and for missing judgments. RBP discounts each succeeding position in the ranking by a probability of examination,  $p$ . When there are missing judgments, RBP is reported as a range between the highest and lowest possible value, that

is, if all missing judgments had been of the highest or lowest possible relevance. Chapelle et al. [24] found in their analysis of Internet search engine click logs that RBP with  $p = 0.7$  closely models user behavior, while DCG over-estimates the likelihood of examination of lower-ranked documents. Moffat and Zobel [97] provide a calculation of the RBP accuracy for different result set sizes and values of  $p$ . Using 25 datasets gives an RBP accuracy to 4 decimal places assuming a “user persistence” factor of 0.7 (often used for Internet search audiences), and to 2 decimal places with a factor of 0.83; this greater persistence might be expected from a scientific audience. We also felt that asking our scientific users to rate relevance for 25 datasets was testing the limit of their patience. In order to accentuate possible differences under different scoring formulae, we removed the searches in which all ratings were the same, leaving 22 searches. Using  $p = 0.7$  and  $p = 0.83$ , we calculated the mean RBP range for the alternative scoring formulae against all searches. We assumed all unjudged documents were not relevant for the lower bound and assumed all were highly relevant for the upper bound. Results are shown in Table 6.4, and the average ideal and pessimal RBP ranges are also given. The upper bound is less than the theoretically achievable 1.0, reflecting that ratings below “highly relevant” were given to a substantial proportion of returned datasets. With the ideal RBP as our target, we see that scoring alternative S2 more closely approximates user rankings than the current formula. Formula S2 weights datasets even more heavily towards the closest edge than do either the current or S3 formula. As with condensed DCG, however, the results

Table 6.4. Comparison of Average RBP Ranges at Rank 25 for Ideal, Pessimal, Current and Alternative Scoring Formulae

Scoring Alternative	Average RBP Range at Rank 25, $p=0.7$	Average RBP Range at Rank 25, $p=0.83$
Ideal	0.90 – 0.93	0.79 – 0.92
Current	0.73 – 0.75	0.66 – 0.79
SN	0.72 – 0.74	0.64 – 0.77
S2	0.80 – 0.83	0.70 – 0.83
S3	0.74 – 0.76	0.67 – 0.79
S4	0.73 – 0.76	0.66 – 0.79
SX	0.72 – 0.75	0.65 – 0.78
Pessimal	0.36 – 0.38	0.36 – 0.49

were suggestive but not statistically significant. The results varied little across different search types.

#### 6.4 Related Work

There is a considerable body of research [32, 83, 87] into ranked relevance of unstructured text documents and XML against text searches; in contrast, our work focuses on numeric data ranges. Numbers in HTML tables are extracted and searched in Venetis et al. [138], but that work focuses on extracting additional semantics. Numbers are also matched by Agrawal and Srikant [4]. Both these approaches assume each “document” is small, by our standards. To our knowledge, ours is the first application of IR techniques to collections of diverse, potentially large, heterogeneous datasets.

Su, Al-Maskari [88, 126] and others have discussed the relationships between user satisfaction and IR measures; Su found that search efficiency, in terms of user time spent on the search, was the most highly ranked measure of search success. Chapelle et al. [24]

and others have applied DCG and RBP to evaluate systems' results for text retrieval. Our user studies were informed by their work, and we adapt their methods to dataset relevance evaluation and for validating the utility of our prototype.

To our knowledge, ours is the first application of these evaluation approaches to searching over collections of large, heterogeneous datasets.

## 6.5 Discussion of the User Studies

We were encouraged by the strong, positive response to the search style for expressing the respondents' information needs, especially given that none of the users had used the tool prior to the study. We did not hear of any difficulties or concerns with conflating geographic, temporal, variable existence and variable ranges into a single set of search conditions. Although we did not ask for comments in the study, several respondents approached us with unsolicited comments about their experiences. While we cannot tie respondents to specific searches, we presume that these same experiences flavored their responses to the survey questions.

The biggest frustration respondents expressed was with variable searches. The current prototype treats each column name as a variable name. In cases where different parts of the archive use different names for the same environmental variable (e.g., temperature, airtemp, air\_temperature) these are treated as separate variables. At present, only the variable name specified in the search is counted as a match; similar names are not. Multiple similar names in a search are treated as separate search conditions. We believe that multiple names for the same variable is one of the key causes of the lower

“completeness confidence” scores for searches involving variables. Future enhancements may allow multiple variables to be identified as “the same” for searching purposes. In addition, variable units are not currently standardized; we have experimented with unit translations and believe that this problem is tractable. These concerns are reflected in the wide range of responses for the question concerning confidence in complete results. Despite this spread, in six of the searches for variable existence, all but two datasets were judged highly relevant.

Several respondents commented that the tool did not return datasets that they knew existed and matched their search, leading to reduced confidence in completeness. In several cases the respondent demonstrated a search to us and identified supposedly missing datasets. In each case we investigated, the dataset was not similar to the search. This effect was most prevalent for searches with variables, where in several cases a long-running observation platform did not have the relevant sensor for that variable during the search time period, or the variable had a different name from that used in the search. The individual searches all focused on locations and time periods in which there were many potentially highly relevant datasets; thus, there are few low-scoring datasets in our judged sample. This effect is the result of two interacting processes: the center collects data in its area of interest, and their scientists are focused on that area of interest.

We found RBP and the condensed DCG useful in exploring the performance of variations of the scoring formula using the existing dataset ratings. We are encouraged by the consistent performance of the scoring approach across the different search types; the

range-based weighting of search terms across space, time and variable values seems to produce results relatively consistent with user expectations. Although applying RBP and condensed DCG did not result in statistically significant support for any one of the scoring alternatives over the others, they are suggestive that the weighting in the current formula could be improved, perhaps moving to formula S2. However, given the small differences reported, careful assessment of the effort invested versus the potential improvement is warranted; additional, larger user studies may lend support to one alternative over the others.

We do not yet know what level of false positives the scientists will tolerate, although clearly it is desirable that we minimize it. Different choices during metadata creation will change the precision, as will changing the scoring formulae.

Our scientists easily translated their experience with ranked document search into this new setting with nominal training. Despite their extensive previous experience with database-style Boolean retrieval of data, no concerns were raised about ranked retrieval, representing datasets by summaries, or the contents of the dataset summaries. The users accepted our similarity score and accepted without comment the combination of seemingly different distance units of space, time and variable values. Our overall success ratings were high. We attribute the positive response to the ease with which they can now perform a task they had been struggling with; this functionality, after all, is the goal of our research.

We found it relatively easy to adapt IR metrics to assessing ranked datasets and user ratings. User-study approaches from IR were also easily applied. The areas where we did encounter ambiguity tended to be ones that are also ambiguous in text document retrieval. For example, how should we account for the large number of unrated datasets in our test database? What should the relative weight be for a “highly relevant” rating versus a “medium” one? These issues are familiar to IR researchers.

We are encouraged by our experiences in applying IR measures to evaluating dataset ranked search. The strongest message we hear from our users is a request for additional data to be made available for searching, often from other archives. We take this as additional support of the utility of the tool itself, and of the existing similarity measure.

While our user studies necessarily reflect our current user base, we believe the concepts are generalizable to other scientific fields. Based on the similarities between the results for scientists and IT professionals in our first user study, we do not expect to see significant differences from these results when we expand tool usage to a non-research-scientist population.

## 7 Performance and Scalability

While most searches of the current CMOP archive take a few seconds, we are interested in understanding the effects of further growth on interactive response times. As described in Chapter 5, the exploratory scenarios for extreme growth create the greatest challenges for the current architecture and implementation approach. As noted there, we can handle a dramatic increase in the number of users (Scenario 7) via methods external to the prototype code. However, we expect Scenario 8, increasing catalog entries by a factor of 100, to cause a potentially large increase in search response times.

For the system to have utility, it must provide search results in a “reasonable” amount of time. We therefore focus on developing and assessing methods for mitigating the causes of search latency in our system identified in Section 5.2.1. As described there, the search engine’s tasks of performing the search and identifying the top-k results take the majority of the elapsed time seen by the user, and thus are our focus.

In this chapter we describe a set of techniques we developed to improve search performance and report on our performance evaluation of these techniques. Further, as we know that different hierarchies can substantially affect search response times, we evaluate the effect on response times of various different hierarchy designs.

We first give some background to our choices for performance evaluation and summarize related work in Section 7.1. Section 7.2 gives the basic algorithm we use to identify the top-k results for a search. We then describe the techniques we used to improve search performance: Section 7.3 describes the changes to the basic algorithm to add a *filter*, and

Section 7.4 adds *relaxation* – a technique that reduces the filter’s cutoff score if too few results are returned from the search and re-issues the search – to our algorithm. The combination of a filter and relaxation is called *filter-restart*.

We then shift our focus to performance evaluations. Section 7.5 describes the methods and data we used in the evaluations. Section 7.5.3 reports on the evaluation of our relaxation approach. Section 7.7 reports results for several different hierarchy designs. Section 7.8 describes some additional informal experiments performed, and Section 7.9 summarizes our findings across all the performance evaluations.

## **7.1 Background to Performance Evaluation**

We do not yet know what response times scientists consider to be “reasonable”; however, we begin with an assumption that, since scientists are accustomed to using web search engines such as Google or Microsoft’s Bing, they might expect or desire to receive search results within a few seconds. The negative impact on users of increasing system response times was recently reaffirmed by Schurman and Brutlag [117]; adding server delays to existing server response times decreased user search activity, satisfaction and, in their case, revenue. These findings support our focus on understanding the latency effects of changes in catalog size and hierarchy designs.

In all cases, our goal is to maintain search response in an “interactive response” range, while we grow the size or complexity of the underlying metadata collections. Further, we are curious about what techniques could compensate for any slowing, and the relative benefit or compensating value of these techniques. Of the various factors known to affect

performance of the search engine, we selected three for further exploration based on our expectation that they would provide the largest potential for improvement:

- Much research has been performed into query evaluation techniques to improve top-k query performance [65]. Can these techniques be applied to our metadata catalogs with hierarchical metadata, and if so, do they reduce overall latency sufficiently to obviate the need for other techniques?
- Organizing the datasets (and subsets thereof) in a hierarchy provides users with data in scales matching their research interests. It can also improve (or detract from) performance. Do alternative hierarchy design choices substantially affect latency, such that some designs should be avoided?
- We know that retrieving data from the database is a large component of our search engine latency. Our initial data design was chosen for ease of implementation, but constrains our performance. The current physical database model has a simple structure that allows new entries with new variable names to be easily added into the metadata catalog, as they are easily appended to the “variables” table. On the other hand, search performance is limited by this design due to the use of joins and pivot tables, as described in Chapter 5.

Another major performance aspect is the effect of varying the complexity of the queries themselves. We do not yet know what the “usual” number of search terms will be, but we know that this number will have a large effect on response times. User studies in Internet search have found 5 to be a common number for text retrieval [2, 7, 66]; however, we do

not know whether this number of search terms will apply to data search by scientists.

This question is an area for future research.

Section 7.1.1 contrasts our metadata catalog with database indexes. Section 7.1.2 describes common top-k query evaluation techniques, and how they relate to our situation. Section 7.1.3 describes in more detail a particular issue in our selected approach of filter-restart: the problem of identifying appropriate cutoff scores.

### **7.1.1 Metadata Catalog versus Index**

Classic Information Retrieval systems primarily achieve good performance by using inverted indexes, often implemented using key-value datastores [83]. Each (possibly stemmed) word is treated as a key, and the documents in which the word occurs, along with a count of occurrences, are stored as values. The search term (word) is used as a key in the lookup. In contrast, our datasets may contain real numbers and searches are generally for ranges rather than for a specific value, so an inverted index is not an appropriate indexing technology.

Our catalog is index-like; however, the hierarchical structure is not aimed primarily at performance and structure decisions are not mainly driven by performance considerations; instead, the hierarchy structure carries semantic meaning. Each entry is in itself an “object” – a set of related attributes that has a specific meaning external to the system. Our metadata catalog performs a similar function to a book catalog in a library: each entry provides a brief summary of the information to be found in the actual dataset to which it refers. We assume that each individual entry has some amount of semantic

meaning and internal coherence; we thereby differ from an index, where no semantic meaning is implied. Further, parents and children related via a hierarchy within the catalog have a semantic relationship to each other, and we can use those semantics to aid our search. For example, if even the closest edge of a parent is “far” from our search, we know that no child can be close; if a parent is completely inside our search area, then all children will be, too. While we may choose to segment a dataset into a set of children based on the parent’s size, and may even choose the number of children based on the size, we expect (or hope) this segmentation will be performed based on the semantics of this kind of dataset. This semantic relationship may be invalidated if, for example, index-tree balancing functions are applied to the hierarchy. However, where practical and effective, we would like to use indexing concepts to improve performance. Other fields have also combined indexes with semantic meaning; for example, data mining has long used indexes as an aggregation method [11].

We chose to implement our catalog using relational database technology, specifically PostgreSQL. We defined PostgreSQL indexes on the tables in which we store the data; in particular, we use B-trees for non-spatial data, and PostGIS uses R-tree indexes implemented on top of GiST indexes [56, 155] to index spatial data [149]. Our queries use the database indexes to navigate our catalog entries.

We expect the majority of catalog accesses to come from identifying the top-k similar entries to a search; we expect direct access to an individual catalog entry to be less common. Our similarity function does not easily translate into a query against an index:

rather than a request for a specific set or range of identified rows, we wish to retrieve the top-k entries subject to a function that considers the center and the end-points of the data range, in the light of another range known only at the time of the query.

Thus, while index algorithms inform our work, search performance over our catalog is not directly predictable from index-performance statistics. Also, we do not require the hierarchy relationships to be balanced in terms of numbers of levels or numbers of children per level, further undermining analytic analysis and formulaic prediction of performance.

There are similarities between our approach and, for example, Hellerstein and Pfeffer's RD-trees ("Russian Doll" trees) [57]; in our hierarchies the children are subject to the same containment relation they describe, and we are often selecting a subset of the set of children according to some criteria. Again, every entry in our catalog, at each level of a hierarchy, has semantic meaning; however, an implementation of our system using RD-tree indexes instead of the GiST indexes currently used might provide faster performance than that currently achieved with the combination of B-trees and R-trees.

Several other scientific fields have developed specialty indexes for scientific data, with the most advanced work in the field of astronomy [72, 74, 76, 121, 141]. Unlike our case, these systems generally assume they are dealing with homogeneous data.

### 7.1.2 Top-k Evaluation Techniques

In our implementation, search engine latency is primarily driven by the SQL query time (multiple queries may be performed) plus a (relatively) fixed time to process each row

returned from an SQL query, in order to identify the top-k results. The simplest method for identifying the top-k entries from a large collection of entries (according to some score or measure) is to process the entire collection, compute the score of each entry, and then sort the resulting list. This approach suffers from scalability issues as the number of entries increases [65]. In addition, the subsequent sort is a blocking operation. These issues have led to development of improved top-k techniques and algorithms. We wished to explore the use of such top-k techniques and adapt them to our setting.

Ilyas et al. [65] classify over 20 top-k approaches in a taxonomy based on dimensions that capture the capabilities and assumptions about the environment to which each approach applies. The section of their taxonomy relevant to our work is the following: in the query model dimension, we use a top-k selection query (we wish to return the results of a top-k selection query, with the scores based on the outcome of a scoring function); we have data and query certainty, that is, we apply exact methods over certain data; our infrastructure provides both sorted and random data-access methods; we desire an application-level technique (as opposed to a technique that modifies the underlying query engine); and we limit ourselves to monotone score-combining functions. We prefer component-scoring functions that reasonably reflect subjective distance. That is, if feature1 is considered by the majority of searchers to be closer to the matched search term than feature2, then it should have a higher score.

Six techniques identified in their paper have our desired combination of characteristics: Threshold Algorithm (TA) and Combined Algorithm (CA) [40], LPTA [29], Onion

indexes [23], PREFER [64], and Filter-Restart [18]. We describe each in more detail.

Fagin et al.’s Threshold Algorithm (TA) and Combined Algorithm (CA) [40] scan multiple lists, with each list representing a ranking of the same set of objects by a separate scoring component. An upper bound is maintained for the overall score of unprocessed items, and the algorithm terminates when  $k$  objects have been found and no unprocessed item can have a score higher than the  $k^{\text{th}}$  object. While TA assumes that random and sorted access have the same cost, CA assumes they have different costs and thus favors one or the other access based on the cost differential. Guntzer et al. develop a modification called Quick Combine that adds an efficient termination condition and operates efficiently over skewed data [10, 52]. Das et al. develop a variant they call LPTA [29], an algorithm that given a set of views and a top-k query uses the set of views to produce an answer to the query. The algorithm identifies the most promising views (in terms of performance) to use when multiple alternative views could be used to produce the answer, based on a cost-estimation framework. Other variants (e.g., Fagin’s No Random Access (NRA) [40] and Guntzer’s stream-combine [53]) assume that random access is not available, and operate solely on the ordered lists returned from the underlying access methods.

These techniques all operate over a set of ranked lists, one for each individual search term, and they combine the individual search terms according to some function. The techniques assume an underlying engine efficiently returns such a ranked list, and, for optimal performance, can do so in an incremental fashion. While it is possible for our

relational database to generate such a ranked list for each term while using our formula, it is less clear that we can produce these ranked lists efficiently; to do so would require an index of the data range’s center point on each possible search term, and another on each edge of the data range. These three indexes would need to be accessed for each component of the search. We do not believe this approach is practical, and therefore choose to rely on the underlying database engine’s capabilities to return a single set of entries to our application.

A different technique is to use specialized top-k indexes; one example of this technique is Onion indexes [23]. In this technique, each tuple is represented by a convex hull of points, with each point on the hull constructed from the application of a single scoring predicate. An onion index organizes the underlying objects into a series of layered convex hulls. However, as noted by Ilyas et al. [65], these indexes become inefficient for range predicates on attribute values, as the convex hull structure will be different for each set of constraints. This is precisely the situation in Data Near Here.

PREFER [64] answers preference queries by relying on knowing beforehand the attributes to be searched and the scoring function to be applied by the query, but allowing the weights for the attribute combination to be determined at query time. The overall scoring function (also called the preference function) is a linear combination of the attributes. The authors wish to avoid evaluating all records for each query. They do so by creating a (small) set of materialized views where the underlying tuples have been scored based on the pre-defined scoring predicate. At query time they apply an algorithm that

computes the top-k results of a query using a minimal prefix of a (selected) view. In our case, the scoring function changes its details based on the range specified in the search, and thus this approach does not apply to our situation.

Lastly, Filter-Restart techniques such as that of Bruno et al. [18] operate by applying range selection queries to limit the number of entries retrieved. A selection query is formulated that only returns entries above an estimated cutoff threshold (the *filter*), and the retrieved entries are ranked. If the cutoff threshold is incorrectly estimated, performance suffers. If it is underestimated, too many entries are retrieved and search time increases as a result. Conversely, it may be overestimated, in which case too few entries are retrieved, and the query must be re-formulated and re-issued with a lower threshold (the *restart*, *query expansion* or *query relaxation*), consuming extra time. Filter-restart techniques operate on a set of entries returned from an underlying engine; the techniques still block until all scoring and sorting operations are complete, and rely for their performance improvement on the filter reducing the number of entries returned without compromising the quality of the result.

Another oft-mentioned technique is to use a skyline or convex hull to identify the closest points to a given query. Despite the similarities, the top-k closest points do not necessarily lie on a skyline or convex hull [104], thus these techniques do not apply.

We selected filter-restart as the best approach for our situation.

### 7.1.3 Filter-Restart and Cutoff Scores

As the size of the table(s) over which a query is executed increases and the scoring metric is calculated for a larger number of entries, execution time increases. It has long been known that limiting the results returned from a query by using a filter can dramatically improve response times [21]. To reduce the number of rows returned from the database, we add a filter to the search to return a subset of the rows that is guaranteed to contain the top-k results, by applying a formula derived from our scoring formula and adding a cutoff threshold. To be useful, the filter must have sufficient selectivity and be fast enough that the total time taken to evaluate the search and process the results is less than the time that would be taken if we evaluated all rows without the filter.

A challenge for the filter-restart class of techniques is determining how to initially set the cutoff threshold for a given top-k search. As Chaudhuri et al. noted [26], there is as yet no satisfactory answer to estimating result cardinalities and value distributions to identify an optimal cutoff value, leaving this estimation as an open area for research. More recently (2010), Vartak notes that “obtaining exact cardinality assurance with query relaxation has been proven to be NP-Hard” [136, 137]. Researchers have explored a variety of approaches to cutoff selection. For example, Donjerkovic and Ramakrishnan note that a top-k query is equivalent to a simple selection query  $\sigma_{x>\kappa}$  on an attribute x, where  $\kappa$  is a cutoff score determined by N (the number of entries) and by the data distribution over which the selection query is run. Since the query optimizer’s knowledge of the data distribution is not perfect (even for a single attribute), the cutoff score must be estimated.

They propose a probabilistic optimization framework, using histograms in the database catalog to estimate the probability that a particular cutoff score will provide the desired selectivity without restarts, and assuming that the histograms represent variables whose values are independent from each other.

Bruno et al. translate a top-k query into a range query for use with a traditional relational database management system [18]; they perform this translation for three monotonic distance functions, SUM, EUCLIDEAN DISTANCE and MAX (of some subset of attributes). If too few results are obtained from the initial query, they restart the query with a lower cutoff score. Like Donjerkovic and Ramakrishnan, they use database-catalog statistics to estimate the initial starting score. In each test, they make an assumption about the relationship between the distributions of multiple attributes; commonly, the distributions are presumed to be uniform and independent [84]. However, Bruno et al.’s experiments show that applying the uniformity assumption within histogram buckets was computationally expensive and led to many restarts [18]. They also note that “no strategy is consistently the best across data distributions. Moreover, even over the same data sets, which strategy works best for a query  $q$  sometimes depends on the specifics of  $q$ ” [18].

Billerback and Zobel [15] focus on testing document ranking performed using the Okapi BM25 measure, which requires several parameters that they prime by using values determined in experiments on a particular test data set. The key parameters are then fixed and used during additional query-expansion experiments. They find that no fixed choice

is robust across different collections; entirely different values give the best result on different collections, and “worse, the best choices per query vary wildly.”

For our primary use case (observational data), it is more common for multiple attributes to be either positively or negatively correlated than for them to be independent. Our data may also be strongly clustered temporally, spatially, or on some observational variable. Thus, we can expect to see issues similar to those found by other researchers in identifying the best cutoff value to use.

Bruno et al. [18] attempt to adapt to the underlying data by sampling it and running a training workload on the sample to identify a cutoff score. Other queries from a similar workload can then use the same cutoff score. In our case, as the similarity function is part of the filter and is affected by the relative ranges of each search term, the appropriate cutoff score is greatly affected by the individual search. Thus, in order to use this technique, we also need to identify a “search similarity function” that can compare the current search to other searches, then use the cutoff score from the most similar search. It might be possible, over time, to use a history of past searches and corresponding final scores in this way.

#### 7.1.3.1 Restart and Relaxation

We have discussed the need to estimate an initial cutoff score. However, when an initial cutoff score fails to return the requisite number of matches, the search must be restarted, that is, reissued with a revised cutoff score designed to return a larger number of matches [18]. This technique is often called *relaxation*, which is defined by Gaasterland as

“generalizing a query to capture neighboring information” [41]. As with the initial cutoff score, we must identify a revised cutoff score to use for the revised search.

Koudas et al. [73] focus on relaxing queries (conditions) on numeric attributes, where the relaxations can be quantified as value differences. They compute a “relaxation skyline” of query results, and provide several algorithms based on which parts of the query the user is willing to relax. However, their top-k algorithm returns the top-k on the skyline, which may not be the actual top-k [104].

More recently, several researchers have focused on relaxing queries while limiting the semantic distance of the results. Elbassuoni et al. [37] develop techniques for automatically relaxing SPARQL queries performed over RDF triples. They use a knowledge repository to provide alternative terms to those used in the original query; terms are selected based on their distance from the original under a language model. Results are ranked based on a combination of their match to the query under any desired scoring function. Their approach focuses on semantic distance between RDF triples and between entities, whereas our work focuses on comparing the distances of a numeric range in a query to a numeric range in a dataset summary. Poulovassilis and Wood also search over semi-structured data modeled as a graph, such as RDF triples [108]. They relax the given query to return similar tuples, where similarity is assessed by identifying alternate terms that are structurally “close” to the desired term in a provided ontology. They differentiate relaxation from approximate queries, which they also address, wherein the desired graph path is “approximated” by alternate paths through the graph that are

ranked according to their graph-edit distance. These techniques may be useful in future research, such as in comparing named categories (“high pH” or “low pH”, for example) to a variable with numeric values (“pH 9.23”).

## 7.2 Basic Algorithm

In this section, we present the basic algorithm we use to identify the top-k entries across all levels of our hierarchy. The pseudo-code is shown in Figure 7.1. We first process the given search terms, and build the static sections of the SQL query we will use to retrieve entry information from the database tables that represent our metadata catalog. Using this query, we then retrieve all roots from the database and score them. Any entry we deem likely to have “interesting” children we add to a list of parents. We then modify the SQL query to retrieve the children of this list of parents, thus accessing the next level of the hierarchy. We repeat this process for each lower level of the hierarchy, with each subsequent query based on the results of the previous one. When we get no rows returned from the database or we find no entries with likely interesting children at some level of the hierarchy, we terminate. Thus, rather than a single top-k query we issue an SQL query for (potentially) each level of any hierarchy within our forest of hierarchies.

The entries are scored and sorted, and the top-k returned. Entries may have widely different numbers of children, with widely different scores. Entries in our top-k could be contributed from any level of the hierarchy, and our top-k is aggregated from the results of several SQL queries. The final selection of our top-k can only be determined after all relevant hierarchy levels have been processed.

At first glance it may appear that we will score every entry in the catalog, retrieving them in a series of queries (and therefore with longer latency than performing a table scan and processing every entry in sequence). Whether or not this is the case depends on the ranges of the search terms as compared to the entries in the metadata catalog. If all entries in the catalog are roots (and therefore have no children), then a table scan is, in fact, performed. On the other hand, if the ranges of the relevant variables in an entry are completely within the search term's range we deem the children not interesting, as each child will contain only a subset of this (already identified as interesting) entry's data; therefore, only the parent entry is returned. If the closest and furthest edges of the variable's range have the same score (within some delta), we deem the children not interesting, as the children's scores cannot be higher than the score of the closest edge of the variable. Again, only the parent entry is returned. In these cases, only a subset of the entire catalog's entries are scored.

Note that we do not ask the database to sort the returned entries (that is, we do not add a sort clause to the SQL query). Although we do currently perform some calculations in the SQL query, we chose to perform the majority of the scoring and sorting in our search engine code. We could equally have chosen to place the scoring functions in PostgreSQL functions or directly in the SQL query. Our experience with PostgreSQL functions is that their invocation is slow and their inclusion in an SQL query causes the optimizer to choose suboptimal plans. The complexity of these options makes customizing the SQL query while accounting for the potential variety of search terms fragile and error-prone.

---

```

// Set starting parameters
// "highest" is the score we'll return for an item that's a complete match
// for all given search terms
highest = 100
// note that distance is unbounded - so lowest possible score is not limited
// but, we can choose to specify a score below which we don't return results
lowest = -100
// limit items returned to some maximum number
maximumMatches = 50

// array to save matches found, from any level of the (chosen) hierarchy
matches = array()

Parse_search_parameters()
// incorporate search parameters into the body of the SQL query
create_sql_body()
// For Basic Algorithm, set SQL 'where' clause to null
sql_where = ''
// start traversing the hierarchy with the forest's roots
sql_where_end = ' parent isnull'

// Main retrieval loop
do { //till we get no rows returned from sqlQuery,
    // or run out of hierarchy we want to traverse
    sqlQuery = sql_body + sql_where + sql_where_end
    rows = query_result(sqlQuery)
    get_kids_list = null
    for each row do {
        // process each entry from this level of the hierarchy
        tmprowScore = 0                                // initialize the score
        for each search_parm do {
            // calc score for this search parm and add it
            tmprowScore = tmprowScore + score(row, search_parm)
            if (search_parm is a span or overlap or
                (inner and outer edges have 'different' scores))
                // We want to check this parent's children
                get_kids_list = get_kids_list + ', ' + row['id']
        }
        row.score = tmprowScore / count(search_parm)
        // append this row as a "potential match" (all rows, for now)
        matches[] = row;
    }
    sql_where_end = ' where parent in ( ' + get_kids_list + ') '
} while count(rows)>0 and get_kids_list != null
// 'get_kids_list == null' tells us we can stop:
// we have finished traversing the hierarchy
// Now: sort the matches; output the highest-scoring entries
Create empty XMLdoc (header and root)
Reverse_sort matches[] on row_score
For (each match while score>lowest and fewer than maximumMatches) do {
    compose result_snippet
    add result_snippet to XMLdoc
}
Return XMLdoc

```

---

Figure 7.1. Pseudo-code for basic algorithm

### 7.3 Adding a Filter

In the design of our filter, we take as a design constraint that we operate over interval summaries, and that we wish to avoid false negatives; that is, we wish to avoid excluding any entry that should have been included in the top  $k$ . Therefore, for a particular cutoff score (which we'll call  $lowestScore$ ), our filter must return (at least) all entries that would achieve a higher score using our similarity measure (in this case, our distance formula). If insufficient matches are found from an application of the filter, we must restart the query with a “looser” filter.

To implement the filter, we make the following changes to the basic algorithm. We add a filter (“where” clause) to the SQL query, that limits entries returned to:

- Those we estimate will have a final score for the entry that is higher than the cutoff score; that is, entries where the center of the data range has a score of  $lowestScore$  or higher. We may increase a summary's score if the data range overlaps the search term; however, we never decrease a score below that of the center of the data range.

This filter term excludes entries that will not be adjusted upwards and that cannot have a score higher than  $lowestScore$ .

- Those whose children may have higher scores than the cutoff score; that is, entries that are not wholly within the search range, that have children and where the closest edge to the search term<sup>5</sup> has a score of  $lowestScore$  or higher. These parents may have a child with scores higher than the parent (if, for example, a child's bounds are near

---

<sup>5</sup> In the one-dimensional case, the closest edge for a variable is the endpoint of the variable's range that is within the search range or that has the smallest delta from either end of the search range.

- the parent’s closest edge). If the closest edge has a score lower than *lowestScore*, not only will the parent not have a possible score higher than *lowestScore*, but none of its children will either; thus, “far” parents and their children are not retrieved.
- Entries which span at least one search term; or, where the closest edge has a score higher than *lowestScore*. These entries’ scores will be adjusted upwards based on the amount of overlap with the search term. They also may have children whose bounds are completely within the search term, and thus are perfect matches for that term.

Given the confounding effects described in Section 7.1.3, our initial cutoff score is set somewhat arbitrarily, based on experience (currently 95). Even if the cutoff score is incorrect, issuing the queries with that cutoff score gives us information about the distribution of relevant table entries, which we can choose to exploit in adjusting the cutoff score. Markl et al. use multivariate statistics to estimate the joint frequencies of data distributions for use in the RDBMS’ optimizer [84]; techniques such as this could be adapted to provide a better estimate of the initial cutoff score, especially as continued growth in underlying data warrants additional complexity to maintain the desired interactive response times.

For geospatial search terms, we developed a slightly different filter, in order to reduce the number of heavyweight spatial calculations. In addition to storing each entry’s shape, we add two columns to our data model. When building the metadata entry, one column is populated with the shape’s centroid, and the other with the maximum radius (*maxradius*) of the shape, allowing us to quickly calculate a lower bound on *mindist*.

---

```

// Set starting parameters
// "highest" is the score we'll return for an item that's a complete match
highest = 100
lowestScore = 95      // starting cutoff score for filter: naive
...
create_sql_body()    // incorporate search parameters into the SQL query

// Calculate the relative distance contributions from each of the searchParms
// For each search parameter: since our distance measure is in "number of radii",
// mult is a factor that acts to apply our current (linear) distance-to-score
// conversion function
f = 10              // number of radii past the edge that makes something "too far"
mult = searchTermsCnt * (highest - lowestScore + f)/f;

// set up the geo, time and depth radii with defaults if necessary
// [this section varies according to the types of search terms;
// the example is for a search with time, location, depth, 1 variable]
factors = qdr * qgr * qvr * qtr;
// qdr: radius of distance search parm; qgr: radius of geo s.parm;
// qtr: radius of time s.parm; qvr: distance of variable s.parm
locDistFactor = factors / qgr;
depthDistFactor = factors / qdr;
timeDistFactor = factors / qtr;
varDistFactor = factors / qvr;
estDistTooFar = mult * factors;

// Set up filter: composed of terms that:
// a) filter for the current hierarchy level
sql_where_thislvl = ' true ' // create, initialize variable
// and b) terms that filter for parents likely to have eligible children
// create, initialize variable (left as "false", if not reset)
sql_where_kids = ' false '

For each search_parm do {
    // add to the filter terms specific to this search parm:
    // first, to the "this hierarchy level" filter term
    sql_where_thislvl = sql_where_thislvl +
        ' + least(abs(minvar - qvcenter), abs(maxvar - qvcenter))*varDistFactor '
    // or, locDistFactor, depthDistFactor etc., as appropriate for the variable
    // in question.
    // And then, add to the filter terms looking for children that may be
    // relevant (even if parent is not)
    sql_where_kids = sql_where_kids + ' or ((minvar < qvmin and maxvar > qvmax )
        or least(abs(minvar - qvcenter),abs(maxvar - qvcenter)) <=
        (mult * qvradius) ) '
}
// Complete composing the where clause:
sql_where = sql_where_thislvl + ' < estDistTooFar or (' + sql_where_kids + ')'
...

```

---

Figure 7.2. Pseudo-code changes for basic algorithm with filter

During a search, the filter calculates the distance from the query to the centroid and subtracts *maxradius* from the distance; this distance is used only in the filter’s WHERE clause to limit the number of entries retrieved by the SQL. The real distance is only calculated by the SQL query for those entries that pass this filter and will be returned to the calling code. We found that the number of spatial calculations avoided for entries that did not pass the filter far outweighed the cost of performing a spatial calculation in the filter that was not directly used in calculating the query results.

Since performance (in terms of elapsed time and resources used) is driven largely by the number of rows returned from the SQL query, applying the filter should improve performance. The first SQL query in any execution only evaluates the root entries in the forest of trees. We access successive levels of the hierarchy (i.e., children) by *parent\_id*, and the filter acts to reduce the number of child entries returned from each level. (We employ an index on *parent\_id*.)

As noted, we currently arbitrarily set a fixed cutoff score (later development may replace this setting with a calculated or estimated starting score per search). However, if the cutoff score (*lowestScore*, in Figure 7.2) is set too high to find  $k$  entries with equal or higher scores, insufficient (or no) rows will be returned from the initial walk through the hierarchy, and a restart will be required. In some cases, we may discover the too-high cutoff score at the root level (for example, if no roots are returned by the initial query); in other cases, we may find that “enough” entries pass the filter but not enough receive high scores (for example, of many entries under a single root only one leaf may receive a high

score). During restarts, previous results are discarded. The effect of the restarts is shown in the performance experiments in Section 7.6.

Our current filter design is simplistic. The number of entries returned could be substantially reduced with a more sophisticated filter; for example, by taking into account interactions between scores on different search dimensions. Note also that as the number of terms in the search increases, our filter becomes less effective, since we retrieve and process any entry that could be “close enough” in any one dimension (since a high score in one dimension, or high-scoring children in one dimension, may sufficiently increase the overall score to balance low scores in other dimensions). It is possible that for high-dimension queries, alternate techniques that combine filter terms may be more effective.

This subject remains an opportunity for future research.

#### 7.4 Relaxation

If fewer than  $k$  entries are found with sufficiently high scores, we restart the search with a lower starting score, that is, we *relax* our filter.

We use the term *iteration* to refer to a sequence of SQL queries and associated processing to walk the hierarchy from root to leaf. Thus, a search that takes two iterations is one in which the entire hierarchy was processed once without finding sufficient entries above the desired cutoff score, the cutoff score was then reduced (*relaxed*), and the entire hierarchy was processed again (equivalently, we may say the search was *restarted*) with the revised cutoff score, this time finding sufficient entries. We use the term *step* to refer

to the processing of a single hierarchy level. Thus, an iteration consists of a series of steps.

We developed and experimented with three relaxation techniques:

- Naïve: In our simplest technique, whenever we do not find sufficient results at our starting filter score, we iterate after having reduced the filter score by a fixed amount (currently set at 15).
- Adaptive: Our second technique uses information calculated while processing the entries returned from one iteration to adjust (in our case, lower) the starting score for the next iteration.
- Contraction: This technique modifies adaptive relaxation by reversing relaxation within an iteration. That is, after each step we check the number of results found so far. If we have found more than  $k$  results with scores higher than our filter score, we increase the filter score for subsequent steps within the iteration.

Clearly, many more variations and extensions on these techniques are possible.

In all cases, our restart design was rudimentary; we discarded the results and restarted the entire evaluation sequence again with a new, lower cutoff score. This restart design is clearly a worst-case; a more intelligent, incremental restart design would be expected to improve performance. The number of “re-found” results will, however, necessarily be fewer than the total number of matches in the iteration.

We now describe each of the three techniques separately. The performance of the techniques will be compared in Section 7.6.

### 7.4.1 Naïve Relaxation

Our first relaxation approach, “Naïve”, takes a simple approach to restarting a query. We make the following additions to the basic algorithm, as shown in Figure 7.3:

- Begin with a *lowestScore* of a given *starting\_score* (e.g., our current default of “95”)
- If no entries were returned from the root-level SQL query: drop the score by some suitably large amount (e.g., we use the empirically-chosen amount of twice *naïve\_reduction*), and iterate.
- If fewer than *minimumMatches* are found, subtract *naïve\_reduction* from *lowestScore* (e.g., our current setting is “15”), and iterate.

---

```
...
// Relaxation loop: includes filter calculations + main retrieval loop
do {    // while less than the desired minimumMatches do
    // Calculate the relative distance contribution from each of the searchParms
    ...
    // Main retrieval loop
    do {        // till we get no rows returned from db,
                // or run out of hierarchy we want to traverse
    ...
    // get_kids_list == null tells us we can stop:
    // we've either finished traversing the hierarchy, or no rows returned
    // from this level of the hierarchy (no (qualified) kids found)
    } while count(rows)>0 and get_kids_list != null

    if (count(matches) < minimumMatches)
        { lowestScore = relaxScoreNaïve() }

} while (count(matches) < minimumMatches)      // end relaxation loop
// Now: we have "enough" matches; output the highest-scoring items
...
relaxScoreNaïve(): {
    // when "relaxing", relax the score by (at least) this amount
    naïve_reduction = 15
    // when nothing returned from db, relax by this amount:
    evenmore = naïve_reduction x 2           // empirically chosen
    // if the SQL query returned rows, but we didn't find matches in them
    if count(rows) > 0    // reduce by "naïve_reduction" amount
        { return lowestScore - naïve_reduction }
    else        // No rows were returned from the SQL query:
                // will need to cast the net "a lot" wider
        { return lowestScore - evenmore }
}
```

---

Figure 7.3. Pseudo-code changes for naive relaxation

The effect is that with each restart, the next “ring” of items (in terms of distance from the query) is retrieved and scored. Previously scored parents and children are re-scored; no attempt is currently made to optimize for rework, and this remains an opportunity for future refinement.

This relaxation approach is simple to apply. It can be used even when no other information is available about data distributions or good starting scores to use.

#### 7.4.2 Adaptive Relaxation

“Adaptive” relaxation refines “Naïve” relaxation by using items scored during the current iteration (which did not return the minimum desired matches, *minimumMatches*) as a sample from which to estimate a revised *lowestScore* to use on the restart.

The changes to the algorithm, shown in Figure 7.4, are as follows: If an iteration results in fewer than the minimum desired matches (*minimumMatches*), we choose as a new *lowestScore* the score from an entry within the sample. We currently use the score of the

---

```

...
    if (count(matches) < minimumMatches)
        { lowestScore = relaxScoreAdaptive() }
} while (count(matches) < minimumMatches) // end relaxation loop
// Now: we have "enough" matches; output the highest-scoring items
...

relaxScoreAdaptive(): {
    sort matches on score from highest
    if matches.length > minimumMatches/2      // heuristic
        { newLowestScore = matches[ floor(minimumMatches/2) ].score }
    else // take score of last entry
        { newLowestScore = matches[matches.length-1].score }
    if abs(lowestScore - newLowestScore) < naïve_reduction
        { return lowestScore - naïve_reduction }
    else
        { return newLowestScore }
}

```

---

Figure 7.4. Pseudo-code changes (from naïve relaxation) for adaptive relaxation

last item in `scoredItems` if fewer than  $minimumMatches$  items have been scored, else we use the score of the middle item of `scoredItems`, with empirically good results. If the amount of reduction of the  $lowestScore$  is less than the  $naive\_reduction$  amount (from Naïve), we reduce by  $naive\_reduction$ . Initial tests showed that if the score of the entry at  $minimumMatches/2$  is greater than the revised score that naïve relaxation would have chosen, performance of the adaptive algorithm was poorer than naïve, because more iterations were sometimes required; thus we added the second condition to ensure that the score reduction is greater than the reduction the naïve algorithm would have applied.

This refinement is effective because in most cases, we get at least some items (perhaps from high in the hierarchy, that is, an aggregate entry with wide ranges) back, so we get some sense of the score distribution near the currently applied filter's boundaries. Note that items returned by the filter may receive scores lower than  $lowestScore$ , since the filter passes on parents that may have high-scoring children. Since reducing the filter's  $lowestScore$  returns more items at each hierarchy level than were returned in the last iteration, we arbitrarily use the score of the entry at  $minimumMatches/2$ . Additional experience with a particular archive or additional research may result in a better estimate. This relaxation refinement is effective when the initial cutoff returns an insufficient number of high-scoring entries, and one or more restarts of the naïve algorithm are required to return the minimum desired number of entries. The expected effect is to reduce the number of restarts required as compared to Naïve by providing a better estimate of the new lowest score for the next iteration.

### 7.4.3 Contraction

This technique modifies “Adaptive” relaxation by adding the concept of *contraction*, that is, the opposite of relaxation. As we process the results for each successive step within an iteration, we check the number of *matches* we have discovered so far, that is, the number of results with scores higher than the current cutoff score. If we already have more than *minimumMatches*, we increase the cutoff score that we use for subsequent queries (at lower levels of the hierarchy). Contraction is applied repeatedly within a single iteration at each step (each successive hierarchy level) to raise the *lowestScore* for subsequent SQL queries, whereas adaptive relaxation lowers the *lowestScore* between iterations. Note that contraction applies once *minimumMatches* have been found, while adaptive relaxation applies when fewer have been found; thus the two methods are complementary and can be applied together.

The algorithm changes, shown in Figure 7.5, are as follows: We modify the inner

---

```
...
    // Main retrieval loop
    do {      // till we get no rows returned from db,
        // or run out of hierarchy we want to traverse
        ...
        if (count(matches) > minimumMatches) {
            // If we already have enough matches: Contract
            sort matches on score, from highest to lowest
            if ( matches[minimumMatches].score > lowestScore )
                {   lowestScore = matches[minimumMatches].score   }
            }
        ...
        } while count(rows)>0 and get_kids_list != null
    // If we don't have enough matches, then relax, using Adaptive Relaxation
    if (count(matches) < minimumMatches)
        { lowestScore = relaxScoreAdaptive() }
} while (count(matches) < minimumMatches) // end relaxation loop
// Now: we have "enough" matches; output the highest-scoring items
...
```

---

Figure 7.5. Pseudo-code changes for contraction

retrieval loop only. At each hierarchy level, if *minimumMatches* have already been found, we set the *lowestScore* to be the score of the entry at *minimumMatches*.

The higher cutoff score potentially reduces the number of entries retrieved from database while processing the next level of the hierarchy, thus potentially reducing the total records processed. Only not-perfect matches are eligible to have their children retrieved (since the children of perfect matches will also be perfect matches, but will contain less data). If there are a number of children that have scores near *lowestScore* and we increase the *lowestScore*, we retrieve only the “closer” subset of them, thus reducing the number to be scored. This approach is most effective when there is a cluster of close-but-not-perfect matches’ children near the current filter edge. In other cases, the increase in the cutoff score may not change the total entries processed.

## 7.5 Performance Tests: Methods and Data

This section describes the methods and data used in the performance comparison of the three types of relaxation we tested: Naïve (N), Adaptive (AD), and Contraction (CN).

We wish to understand the performance of the following:

- How response times change as we increase the collection size over which we search
- Whether, and how much, our query evaluation techniques improve search performance
- The effect of different hierarchy “shapes” on performance

We focus on the performance of a single server. If single-server performance is sufficient for the expected workload, no additional scaling approaches are needed, while if more

servers are needed, the number will be defined largely by the single-server performance. Performance of queries executed by a database engine generally deteriorates rapidly as soon as the database indexes no longer fit into memory, and most highly-used applications retain frequently accessed indexes and data in memory to ensure minimum latencies (e.g., [120]). In order to understand and differentiate the performance characteristics of our approach from the (potentially confounding) impact of underlying disk hardware, we restrict ourselves to data sizes where the metadata catalog can fit its selected working set into our available memory (2GB of buffer pools). We believe this choice is reasonable since during search our metadata is read-only and thus is not likely to become stale; further, as usage grows we would expect the database engine to reside on its own server. While implementing a different data model or different datastore technology may change the format of the memory contents, if similar entries and attributes are used, the amount of memory required should be of a similar order of magnitude.

We created three metadata collections of increasing size over which to run our tests. All three collections summarize real-world data from a non-CMOP archive of interest to our scientists. The smallest collection is approximately five times the size of CMOP’s current collection, while the largest is 200 times the size. The collections are further described in Section 7.5.1.

We implemented the search-evaluation techniques that we describe earlier in this chapter, while retaining the current design of Data Near Here. The current design and

implementation is known to be a limiting factor on performance; however, we feel it provides a useful base for exploration of a variety of techniques to improve performance, the results of which can then be used to inform a more scalable redesign. Portions of such a redesign are described in Chapter 5.

To test different hierarchy “shapes”, we developed eight hierarchy structures, each with different aggregation characteristics. In addition, we keep a ninth “no hierarchy”, with every entry treated as a root. The structures of the hierarchies we developed are described in detail in Section 7.5.2. We instantiated each hierarchy structure over the same leaf entries, for each of the three data collections.

We developed three test suites of searches, comprising three sets of search terms that we know from experience to have very different performance characteristics: time-only, space-only, and time-and-space. The suites are described in Section 7.5.3.

Each query in a test suite was run five times against each target hierarchy. For each search, the maximum and minimum response times were discarded and the response times for the remaining three queries were averaged, as is common in database performance measurement. Whenever a different metadata table or hierarchy was accessed, the first search was discarded, to force the relevant indexes (and, possibly, the metadata tables, if PostgreSQL chooses to) to be loaded into memory; this action simulates the effect of a “warm cache”, or of using the same set of tables repeatedly, as we would expect in a production environment.

All tests were run on a 2 quad-core 2.13 GHz Intel Xeon system with 64 GB main memory, running Ubuntu 3.2.0 with Apache 2.2.22, PHP 5, PostgreSQL 9.1 and PostGIS 2.0. The PHP space limit per request was increased to 2 GB, and the time limit to 1,000 seconds. PostgreSQL 9.1 only uses a single core per user request and the flow through the application is sequential; as a result, a search uses only one processor.

### 7.5.1 Test Collections

We scanned three sets of data from NOAA for use in the performance tests.<sup>6</sup> The first two are chlorophyll-a concentration data from NASA's Aqua Spacecraft<sup>7</sup>, in two different formats: Science Quality (herein abbreviated to collection *s*) and Experimental (herein referred to as collection *e*). The data is gathered by the Moderate Resolution Imaging Spectroradiometer (MODIS) carried aboard the spacecraft, and reported as a grid with values removed for clouds, cover, or technical reasons. The data scanned differs in density ( $0.05^\circ$  for Science Quality versus  $0.0125^\circ$  for Experimental), and has differences in the algorithms used to produce the data. The third collection (herein referred to as collection *t*) contains nighttime surface temperature readings gathered by the Advanced Very High Resolution Radiometer (AVHRR) instrument, a multiband radiance sensor carried aboard the NOAA's Polar Operational Environmental Satellites (POES). The data is provided at high resolution ( $0.0125^\circ$ ) and is not cloud-masked; there is a data point reported at every possible location, resulting in greater data density than the chlorophyll collection.

---

<sup>6</sup> To reduce confusion, we will refer to the results of scanning each of these sets of data as a “catalog”.

<sup>7</sup> <http://coastwatch.pfeg.noaa.gov/coastwatch/CWBrowser.jsp>

We created a configurable scanner that reads the subset of data for our geographic area of interest (the region between latitude 40 to 50, longitude -122 to -127), and creates a leaf dataset record for each (configurable) block. The blocks were configured to be  $0.25^\circ$  latitude by  $0.25^\circ$  longitude; each block is treated as a separate “dataset” from the perspective of the metadata collection. Where there are missing values for cloud cover and other reasons, the actual area of each block represented in the data may be substantially smaller than the nominal area. The scanner checks the physical locations of the data points reported, excludes any “no data” points, and represents the physical extent of the valid data by a convex hull of the valid data points.

The three resulting collections have different catalog sizes. The leaf data counts are: for  $s$  (small), 192,554 leaf records, with 962,770 entries in the variables table; this collection is 5.5 times the size of our current catalog. The medium collection,  $e$ , has 930,261 leaf records, with 4,653,105 entries in the variables table, or 4.8 times the size of  $s$ . The large collection,  $t$ , has 7,066,501 leaf records, and is 201 times the size of our current catalog.

### 7.5.2 Test Hierarchies

We developed eight different hierarchy structures (hereafter abbreviated as “hierarchies”), and a configurable hierarchy creation program. The hierarchies represent different patterns of aggregation of space and time, with the number of levels varying from one (all entries are leaves) to 5 (root, three intermediate levels, and leaves). Each hierarchy was built over the  $s$  and the  $e$  collections; further, for testing convenience, each different hierarchy was stored in a different set of database tables.

Table 7.1 describes each hierarchy used in our tests. Each level defines the entity (time or space) aggregated on, followed by the aggregation characteristics. One or more aggregation entity can be used at each level. For time, we specify the aggregation in terms of a time unit and how many of those units; in the test hierarchies we use 15 days, 1 year, and all. For space, we specify the number of blocks to be aggregated in the  $x$  and the  $y$  direction, starting from grid coordinate (0,0); for example, an aggregation of “space: [1,4]” means that strips 1x4 blocks in size will be aggregated to form the next level of the hierarchy making. “All” means that all items at the next level of hierarchy (meeting other aggregation criteria) are treated as children of a single root.

For four hierarchies, the root level contains the entire spatial coverage (for some segment of time); the other four contain all times at the root level (for some aggregation of spatial blocks). The intermediate levels are aggregated on various combinations of time and space. In Hierarchy “None”, every entry is simultaneously a root and a leaf.

In the performance tables below, a designation “5e” means the  $e$  data with Hierarchy 5; similarly, “5s” means the  $s$  data with Hierarchy 5. For ease of explication in this chapter, we will often refer to the combination of a specific collection and hierarchy as a hierarchy (even for the “no hierarchy” structure). Thus, we may refer to “Hierarchy 5e” or “Hierarchy 5s”. We anticipate that a mixture of hierarchies may coexist in a single set of production database tables, as is currently the case at CMOP; our algorithm is not changed by this intermixing, although performance characteristics will likely become more difficult to characterize and predict.

Table 7.1. Summary of Test Hierarchy Structures. For each hierarchy we describe each level, beginning from the root, to the leaf. Each level defines the entity (time or space) aggregated on, followed by the aggregation characteristics. For time, we aggregate on [unit, number of units], for space, we aggregate together the number of blocks in each direction [x, y]. “All” means that all items at the next level of hierarchy (meeting other aggregation criteria) are treated as children of a single root.

#	Description (from leaf to root)	#Levels	Aggregated on
None	No hierarchy	1	1. leaf
1	Aggregate on time, then space	5	1. space: [8, 4], time: [all], 2. space: [1, 1], time: [year, 10], 3. space: [1, 1], time: [month, 6], 4. space: [1, 1], time: [day, 15], 5. leaf
2	Aggregate on space, then time (to 6 months only)	5	1. space: [all], time: [month, 6], 2. space: [all], time: [day, 15], 3. space: [all], time: [day, 1], 4. space: [8, 4], 5. leaf
3	Aggregate on space, then time	4	1. space: [8, 4], time: [all], 2. space: [8, 4], time: [year, 1], 3. space: [8, 4], time: [day, 15], 4. leaf
4	Aggregate on time, then space, then time	5	1. space: [8, 4], time: [all] 2. space: [8, 4], time: [year, 1], 3. space: [8, 4], time: [day, 15], 4. space: [1, 1], time: [day, 15], 5. leaf
5	Aggregate on space and time [1]	3	1. space: [8, 8], time: [all], 2. space: [8, 8], time: [day, 15], 3. leaf
6	Aggregate on space and time [2]	3	1. space: [all], time: [day, 15], 2. space: [8, 8], time: [day, 15], 3. leaf
7	Interleave aggregations on space and time [1]	5	1. space: [all], time: [year, 1], 2. space: [all], time: [day, 15], 3. space: [8, 8], time: [day, 15], 4. space: [2, 2], time: [day, 15], 5. leaf
8	Interleave aggregations on space and time [2]	5	1. space: [all], time: [year, 1], 2. space: [all], time: [day, 15], 3. space: [8, 8], time: [day, 15], 4. space: [1, 4], time: [day, 15], 5. leaf

### 7.5.3 Search Suites

Since almost all searches include time, space or both search terms, we developed three search suites: one with queries containing only a time search term; one with queries with a space search term only; and one with queries containing both a time term and a space term. Given the performance limitations of the search for variables (due to the current design, which pivots and joins two large tables for every query containing a variable search term) we did not develop a search suite for these queries.

**Time search suite.** This search suite consists of 77 searches. The searches range in duration from 1 day (the smallest duration accepted in this implementation of DNH) to a search spanning 44 years (1/1/1970 to 1/1/2014), which is longer than our entire temporal coverage. We defined searches that match exactly the various levels of the hierarchy; others are offset from and overlap hierarchy groupings, but cover the same durations; and some are both longer and shorter than various hierarchy grouping levels.

**Space search suite.** This suite consists of 20 searches. The smallest search's spatial extent is  $4.4 \text{ km}^2$  (45.07187, -124.532 to 45.1, -124.55), while the largest search spans 2 degrees latitude by 2 degrees longitude (44, -126 to 47, -124) for an area of approximately  $52,000 \text{ km}^2$ , which is a bit more than 8% of our entire geographic coverage area (of roughly  $10^\circ$  latitude by  $5^\circ$  longitude). As with the time searches, we created spatial searches that align with hierarchy groupings; some that are offset from or overlap hierarchy groupings; and others that are both larger and smaller than hierarchy grouping levels.

**Space-Time Search Suite.** This suite consists of 70 searches. The searches were created by selecting several of the fastest, several of the slowest and several of the most-varying time-only searches (in terms of response times, from the results of the tests against all hierarchies); selecting several of the fastest, several of the slowest and several of the most-varying space-only searches; and combining these terms. The resulting list was reduced to 70 searches by removing combinations that appeared very similar or that, by inspecting the results from the individual runs, we expected to have very similar performance characteristics.

In all cases, we requested  $10 \leq k \leq 50$ , that is, a minimum number of 10 and a maximum of 50 entries to be returned from a search.

## 7.6 Relaxation Performance Test Results

This section describes the results from our performance tests for each of our three suites of searches: the time search suite (in Section 7.6.1), the space search suite (in Section 7.6.2), and the space-time search suite (in Section 7.6.3).

### 7.6.1 Time Search Suite Results

We first ran a set of tests for the time suite to ensure the filter alone (without restart) provides a performance benefit. Adding a filter without adding restart capabilities means that a search may not result in the minimum requested entries (*minimumMatches*).

The geometric mean<sup>8</sup> response time for the time suite as run against each hierarchy is shown in Table 7.2, along with the number of searches in the suite that completed. We

---

<sup>8</sup> Since each search's response time is the average of the middle 3 of 5 runs, geometric mean is used to

Table 7.2. Time Search Suite Response Times, No Filter vs. Filter (in seconds)

Hier- archy	<i>s</i> Collection						<i>e</i> Collection					
	No Filter			Filter			No Filter			Filter		
	Mean	Std. Dev.	Count	Mean	Std. Dev.	Count	Mean	Std. Dev.	Count	Mean	Std. Dev.	Count
1	2.245	3.279	77	0.596	0.395	77	3.812	5.893	77	0.876	0.547	77
2	0.253	0.171	77	0.108	0.038	77	0.232	0.098	77	0.119	0.021	77
3	0.376	0.314	77	0.146	0.055	77	1.063	1.323	77	0.276	0.134	77
4	1.250	2.002	77	0.277	0.152	77	1.651	2.304	76	0.381	0.203	77
5	1.118	1.264	77	0.235	0.096	77	1.363	1.543	77	0.299	0.142	76
6	0.829	1.227	77	0.169	0.072	77	1.102	1.485	77	0.245	0.115	77
7	0.968	1.473	77	0.197	0.091	77	1.278	1.792	77	0.279	0.133	77
8	0.978	1.497	77	0.200	0.093	77	1.290	1.795	77	0.283	0.134	77
None	18.554	0.696	77	1.543	5.462	76			0	4.568	23.193	61

show results for both the *s* (smaller) and the *e* (medium-sized) collections. The fastest response times are colored in green, while the slowest are colored red. Hierarchy 2 gave the fastest response times for both collections, while “None” (that is, all entries are roots and are processed for each search) gave the slowest.

For most of the hierarchies, adding the filter reduces the response time to one-fifth or less of the response time without the filter.

Note that for “None” and no filter, every search failed to complete (either exhausting PHP memory or time limits) for the *e* collection; when the filter was added, nearly four-fifths of the searches completed, although the mean response time for those queries was very high compared to the other hierarchies. For the other hierarchies, all searches for the *e* collection completed except in two cases, where one search failed to complete.

We then ran the suite using Naïve (N), Adaptive relaxation (AD) and Contraction (CN) (which builds on Adaptive relaxation) against the  $e$  and  $s$  hierarchies. The majority of the time searches completed in one iteration and did not use relaxation. Hierarchies 1e, 3e and 5e had no searches with two or more iterations. Hierarchies 2e and 6e each had one search with a second iteration; in both cases, the drop in the score recommended by relaxation was small enough that the *naïve\_reduction* was used instead of the adjusted *lowestScore*, and so the results were the same as for Naïve relaxation.

The only searches noticeably affected by Adaptive relaxation were against Hierarchy 2s.

Table 7.3. Time Search Suite, N vs. AD, for the Five Searches with More Than Two Iterations

Hierarchy	Naïve (N)				Adaptive (AD)			
	Total # of Iterations	Avg. SQL Rows	Avg. Matches Found	Mean Response (s)	Total # of Iterations	Avg. SQL Rows	Avg. Matches Found	Mean Response (s)
2s	21	845	192	0.16±0.07	14	334	192	0.12±0.03

Table 7.3 reports, for the five searches in Hierarchy 2s with more than two iterations: the total number of iterations across all the searches, the average rows retrieved by the filter per search (“Avg. SQL Rows”), the average number of matches found (that is, number of items with scores higher than *lowestScore* on the last iteration performed) and the mean response time. (Note that the number of iterations, rows retrieved and number of matches are the same across runs of the same query with the same method, although the response time may vary.) Across these five searches AD reduced the total number of iterations by 7, while reducing the number of rows returned from the database by 60% and reducing the mean response by a quarter, with lower variability.

Since many of the time searches completed in well under a second against the smaller  $s$  collection and noise in the system was greater than the differences between the algorithms we wanted to measure, we report results in Table 7.4 for the time search suite against the larger  $e$  collection, while including one  $s$  collection (2s) for comparison. Table 7.4 reports the average rows retrieved from the database, the average number of entries returned from the search (Avg. Matches Found) and the mean response time for the subset of searches affected by CN, when run against each of selected hierarchies. We also show the number out of the 77 searches in the suite for which the low score was increased as a result of contraction.

As can be seen, contraction increased the final cutoff scores for the majority of the 77 time searches. However, contraction can only be considered to be beneficial if the reduction in total processing time from reducing the number of rows returned from the database and processed more than compensates for the additional work required to calculate new cutoff scores to use (i.e., storing and sorting current matches).

Table 7.4. Time Search Suite, AD Only vs. CN, for searches affected by CN only

Hierarchy	# of Searches with Increased Low Scores	Adaptive (AD)			Contraction (CN)		
		Avg. SQL Rows	Avg. Matches Found	Mean Response (s)	Avg. SQL Rows	Avg. Matches Found	Mean Response (s)
1e	71	12,757	11,387	1.056±0.580	8,883	7,033	0.879±0.431
2s	58	790	704	0.221±0.071	700	223	0.159±0.045
2e	73	409	388	0.292±0.147	385	89	0.194±0.095
3e	63	5,249	5,126	0.590±0.388	3,088	2,676	0.307±0.094
5e	63	6,037	5,948	0.621±0.307	3,927	3,576	0.319±0.096
6e	63	4,843	4,773	0.500±0.252	2,920	2,604	0.361±0.210

For the five searches with exactly two iterations, the final cutoff score for AD was the same as the Naïve cutoff; this means that the revised cutoff calculated by AD based on results of the first iteration was either less than or the same as the *naïve\_reduction* value. For these five searches, the final cutoff for these searches when using CN was higher than N and AD's final cutoff, indicating that the ideal cutoff was between the cutoff scores used in the first two iterations (that is, between the initial setting of *lowestScore* and *lowestScore* minus *naïve\_reduction*).

As can be seen in Table 7.4, CN results in lower mean response times and lower variability (lower standard deviations) than AD for affected searches for every hierarchy. The effect of CN on the number of rows processed varies from a reduction of 6% for hierarchy 2e with a reduction in average response time of approximately a third, to a reduction in rows of 42% for hierarchy 3e, with a reduction in average response time of nearly one half.

Note that while many of the searches that AD and CN benefit were among the longest running, other long-running searches do not benefit from them. It is possible for long-running searches to complete in a single iteration and without changing the cutoff threshold; for example, if many entries are returned at each hierarchy level but very few of them have higher scores than the current cutoff score. Note also that despite an increase in the number of records by a factor of 4.8 from the 2s to the 2e collection (from 192,554 leaf records and 205,163 total records to 930,621 leaf records and 993,535 total records), the response time increased by only around 20%.

We conclude that for this search suite, the combination of the filter with adaptive relaxation and contraction provides the best results for all hierarchies. While these techniques do not help all searches, they tend to help long-running searches and they do not negatively impact the searches that they do not improve.

### 7.6.2 Space Search Suite Results

For consistency, we tested the space search suite against the same collections and hierarchies as used for the time search suite. As many of the “no filter” searches did not complete, we do not include those results. Again, we report results for the  $e$  collections while including one  $s$  collection (2s) for comparison.

Of the 20 searches in the space suite, seven resulted in more than one iteration using naïve relaxation; four searches had two iterations only. Three searches had more than two iterations, and one search iterated eight times before exiting and returning fewer than the minimum desired results.

For the searches with two iterations (that is, the searches that might be affected by AD), AD used the same final cutoff score as Naïve relaxation; this means that the revised cutoff calculated by AD after the first iteration was either higher than the “minimum cutoff adjustment”, or the same as Naïve’s cutoff score. (In fact, for all tests run except 1e, CN used the same final cutoff score as N for these searches, indicating that this effect is the result of interaction between the search and the data distribution. As such, this effect may not recur with other data and provides an opportunity for further experimentation.)

Table 7.5 summarizes the total number of iterations, average rows returned from SQL, the average number of matches returned from the searches, and response times for the three searches with more than two iterations. All response times are significantly longer than for the time search suite, as the spatial comparisons used in the distance measure take more time to calculate than the numeric calculations used to calculate “time distance”. In every case, AD reduced the number of iterations and reduced the response time by more than 50%, even in the cases where the number of rows returned from the database (Avg. SQL Rows) were almost identical to those returned for N. This reduction is because the *lowestScore* was adjusted down by more than the *naïve\_reduction* value for all affected searches, and in most cases the search found *minimumMatches* during the first restart.

Table 7.5. Space Search Suite, N vs. AD: Summary of (Three) Searches with More Than Two Iterations

Hierarchy	Naive (N)				Adaptive (AD)			
	Total # of Iterations	Avg. SQL Rows	Avg. Matches Found	Mean Response (s)	Total # of Iterations	Avg. SQL Rows	Avg. Matches Found	Mean Response (s)
1e	17	20,419	36	33.83±6.22	7	20,229	24	14.40±2.19
2s	19	7,688	33	14.24±3.52	6	7,444	15	4.45±0.21
2e	17	33,302	36	61.57±7.35	8	33,074	31	29.92±2.35
3e	17	22,505	35	31.51±5.29	7	21,754	24	13.17±2.88
5e	17	22,484	35	32.89±4.34	7	21,732	24	13.76±3.24
6e	17	22,984	35	32.89±4.34	7	22,232	24	13.76±3.24

Table 7.6 reports the average rows retrieved by SQL and response time for the subset of searches affected by CN, when run against each of the selected hierarchies. As shown in Table 7.6, CN increased the final cutoff scores for 13 of the 20 space queries for

Table 7.6. Space Search Suite, AD vs. CN, only for searches affected by CN

Hierarchy	# of Searches with Increased Low Scores	Adaptive (AD)			Contraction (CN)		
		Avg. SQL Rows	Avg. Matches Found	Mean Response (s)	Avg. SQL Rows	Avg. Matches Found	Mean Response (s)
1e	13	115,191	39,171	35.02±58.37	82,115	3,549	29.58±24.78
2s	5	65,361	33,256	21.31±6.33	44,117	9,043	14.69±3.60
2e	7	239,470	114,544	78.24±69.33	158,978	30,461	54.31±34.33
3e	5	286,736	147,625	93.89±44.18	212,842	91,195	66.35±25.40
5e	4	333,281	181,754	110.57±48.38	272,107	128,743	86.97±23.80
6e	4	333,678	181,751	110.57±48.38	272,499	128,741	86.97±23.80

hierarchy 1e, but only for between four and seven searches over the other hierarchies.

As Table 7.6 shows, CN results in lower mean response times and lower variability (lower standard deviations) for affected searches in every hierarchy tested. CN reduces the number of rows processed by approximately 20% to nearly 50%, while reducing response time by approximately 50% and reducing variability in all cases.

Recall that the 2e collection contains 4.8 times as many records as the 2s collection, and the average response time nearly tripled. This result is in contrast to a response time increase seen in the time searches of only around 20%.

We conclude that for this search suite (as with the time suite), the combination of the filter with adaptive relaxation and contraction provides the best results across all hierarchies. While these techniques do not help all searches, they also do not negatively impact the searches that they do not improve. As multiple iterations tend to be associated with long response times, the searches with the longest response times tend to be the ones most improved.

### 7.6.3 Space-Time Search Suite Results

For consistency, we tested the combined space-time queries against the same data and hierarchies we used for the time search suite and the space time search suite. Again, we report results for the *e* collections while including one *s* collection (2s) for comparison. Of the 70 searches in the space-time search suite, 22 resulted in more than one iteration using naïve relaxation for the *e* collections; of these, 10 searches had two iterations only, 11 searches had between 4 and 7 iterations, and one search iterated 8 times before returning fewer than the minimum desired results. For Hierarchy 2s, 28 searches resulted in more than one iteration; of these, 8 searches had two iterations only. Table 7.7 summarizes the average rows returned from SQL and response times for the twelve searches with more than two iterations. As shown in Table 7.7, for this search suite, some searches (# Failed Searches) exhausted either memory or 1,000 seconds of processing time and failed to return results; the other columns exclude the failed searches.

CN modifies AD by “contracting”, or raising the final cutoff score as soon as the desired

Table 7.7. Space-Time Search Suite, N vs. AD

Hierarchy	# Failed Searches	Naïve (N)				Adaptive (AD)			
		Total # of Iterations	Avg. SQL Rows	Avg. Matches Found	Mean Response (s)	Total # of Iterations	Avg. SQL Rows	Avg. Matches Found	Mean Response (s)
1e	4	61	129,951	35	45.16±40.72	30	35,819	32	22.50±22.54
2s	0	107	48,965	44	19.70±13.18	44	11,963	62	8.30±5.61
2e	6	61	182,293	34	69.45±32.08	33	49,245	94	38.91±27.10
3e	3	61	41,806	33	14.23±10.16	29	13,832	30	7.15±5.32
5e	4	61	42,646	33	15.35±10.05	29	13,997	30	7.73±5.29
6e	4	61	45,485	33	17.05±10.47	29	14,626	30	8.47±5.50

number of matches has been found. Table 7.8 reports the average rows retrieved by SQL and response time for the subset of searches affected by CN, when run against each of the selected hierarchies. As shown in Table 7.8, CN increased the ending scores for between 19 and 38 of the 70 space-time searches, depending on the hierarchy. CN is only beneficial if the reduction in processing time by reducing the number of rows returned from the database and processed more than compensates for the additional time required to calculate new cutoff scores to use (i.e., storing and sorting current matches). CN gives lower mean response times and lower variability (lower standard deviations) than AD for every hierarchy tested. CN reduces the number of rows returned by SQL and the response time in all cases. In all hierarchies except 1e, the variability is reduced. Thus the extra costs of CN are more than offset by the improvements.

Note that the 2e collection contains 4.8 times as many records as the 2s collection and that the average response time increased by a factor of 3.8. This increase is in contrast to a response time increase of only around 20% for the time-only searches and around 3 for the space-only searches.

Table 7.8. Space-Time Search Suite: AD vs. CN, only for searches affected by CN

Hierarchy	# Searches with Increased Low Scores	Adaptive (AD)			Contraction (CN)		
		Avg. SQL Rows	Avg. Matches Found	Mean Response (s)	Avg. SQL Rows	Avg. Matches Found	Mean Response (s)
1e	38; 4 fail	170,718	40,288	198.93±177.50	120,687	2,190	178.40±208.32
2s	25; 0 fail	58,004	22,428	19.15±14.43	37,607	5,170	14.53±8.09
2e	30; 6 fail	149,515	37,909	78.24±69.33	103,047	6,743	54.31±34.33
3e	22; 3 fail	185,032	77,107	45.11±55.06	112,464	26,329	31.37±34.94
5e	19; 4 fail	189,814	75,990	47.71±50.96	140,513	44,250	39.03±38.76
6e	19; 4 fail	190,152	75,993	48.06±50.78	140,850	44,253	39.30±38.87

We conclude that for this search suite (as with the other two suites), the combination of the filter with adaptive relaxation and contraction provides the best results across all hierarchies, although the improvement is not as great as seen in the other two cases. As multiple iterations tend to be associated with the longest response times, the searches with the longest response times tend to be the ones most improved. While these techniques do not help all searches, they also do not negatively impact the searches that they do not improve.

## 7.7 Hierarchy Performance Test Results

The differences seen in the previous section between the performance of the different hierarchies is much larger than the effects of different styles of relaxation, and thus deserves exploration. The primary purpose of the segmentation of datasets and grouping into containment hierarchies is improved usability by allowing us to match scientists' information needs to the best available segment of a dataset. However, we recognize that there are often several equally valid-seeming ways of grouping and segmenting a large set of data, and that segmentation decisions must sometimes be made with limited knowledge or with assumptions about how the data will be used; in these cases, knowing the performance impact of the choices might be helpful in choosing amongst the options. We sought to compare the different hierarchies across the different search suites with the goal of providing guidance to archive curators on the response time effects of their hierarchy choices. Based on the results of the tests in Section 7.6 we used Adaptation plus Contraction as our default for the hierarchy comparisons. As noted, we expect that

Table 7.9. Time Search Suite: Hierarchy Comparisons. (Response time in seconds)

Hierarchy	<i>s</i> Collection		<i>e</i> Collection		<i>t</i> Collection	
	Mean	Std. Dev.	Mean	Std. Dev.	Mean	Std. Dev.
1	0.832	0.746	1.165	0.947	4.661	6.271
2	0.217	0.125	0.440	0.294	0.263	0.878
3	0.246	0.171	0.773	0.846	2.950	5.274
4	0.256	0.103	0.760	0.766		
5	0.204	0.075	0.641	0.560		
6	0.140	0.033	0.480	0.370		
7	0.175	0.048	0.484	0.354		
8	0.177	0.051	0.395	0.277		
None	2.490	5.835	4.947	23.981		

multiple hierarchies may coexist within a single metadata collection. However, for the purpose of our performance tests we split each different hierarchy into a separate set of physical tables in our relational database, so that during a single test run the search suite only encountered hierarchies of a single pattern.

### 7.7.1 Time Search Suite Results

We ran the time search suite against the *s* and *e* collections for all hierarchies. In addition, we ran the time suite against the larger *t* collection for three hierarchies only. Table 7.9 compares the (geometric) mean response time for each hierarchy across the entire time search suite. The hierarchy with the lowest response time for each collection is highlighted in green, and the worst in red. Mean response time increased in all cases from the *s* to the *e* collections for the same hierarchy, but by less than the increase in collection size (a factor of 4.8); Hierarchy 1 mean response time increased by the least, a factor of 1.4, while Hierarchy 6 increased by the most, a factor of 3.4.

Table 7.10. Space Search Suite: Response Times by Hierarchy (in seconds)

Hierarchy	<i>s</i> Collection		<i>e</i> Collection		<i>e</i> Collection, excluding 5 “large” Searches	
	Mean	Std. Dev.	Mean	Std. Dev.	Mean	Std. Dev.
1	12.035	10.640	28.659	22.853	16.620	4.039
2	7.365	4.904	38.312	27.119	24.952	5.425
3	6.669	5.174	28.994	27.267	15.256	3.975
4	32.408	41.698	33.552	31.188	17.397	4.434
5	38.393	54.760	31.829	32.155	15.150	3.954
6	39.017	55.451	32.987	33.941	15.672	3.931
7	27.334	32.000	27.752	21.972	16.408	3.592
8	29.393	37.223	29.631	27.204	16.365	3.827
None	7.968	8.255	38.304	43.354	16.972	4.170

Figure 7.6 and Figure 7.7 show the level of variability in response times across the search suite. Each figure shows the searches in the time search suite ordered by response time within that hierarchy’s searches. For example, the points at the 10 mark are the 10<sup>th</sup> fastest searches for each hierarchy respectively, and thus may not represent the same search within the suite.

Because of the dramatic differences in graph scale, we show the same data at three levels of detail. Figure 7.6a shows all hierarchies for the *e* collection, showing that all other hierarchies provide reduced and more consistent response times than “None.” Figure 7.6a also shows the dramatic increase in response times for some searches in “None”; the remaining searches did not complete within the resource limits. Figure 7.6b removes “None” in order to better compare the other hierarchies. Figure 7.7a removes hierarchy 1e. Figure 7.7b shows the same set of hierarchies run over the *s* data, which is a much smaller collection and has a somewhat different data distribution. The results are similar

in pattern for most hierarchies, but the response times are faster.

The other  $s$  graphs are almost identical and have been omitted. All hierarchies give improved response over ‘‘None’’; in fact, the improved response holds for every search within the suite.

### 7.7.2 Space Search Suite Results

Table 7.10 compares the geometric mean response time for each hierarchy, across the entire space search suite. The hierarchy with the lowest response time for each collection is highlighted in green, and the worst is highlighted in red. Mean response time varied widely from the  $s$  to the  $e$  collections; in two cases (Hierarchies 5 and 6) it dropped by 20% despite the increase in collection size; in three cases it was close to the same (Hierarchies 4, 7 and 8), and in three cases (Hierarchies 2, 3 and None) it increased by a factor close to the increase in the collection size.

There is far more variability in the mean response times across hierarchies for the  $s$  collection than for the  $e$  collection. We believe this variability is because the data distribution and aggregation approach are the same for each collection, but the  $s$  collection contains only one-fifth as many entries. Thus, each tree in the  $s$  collection contains approximately one-fifth as many entries as the tree in the  $e$  collection with the same bounds. An iteration in an  $s$  hierarchy with a given cut-off score will have fewer parents returned at each level, thus is less likely to find *minimumMatches* while traversing the hierarchy, and hence more likely to require a restart.

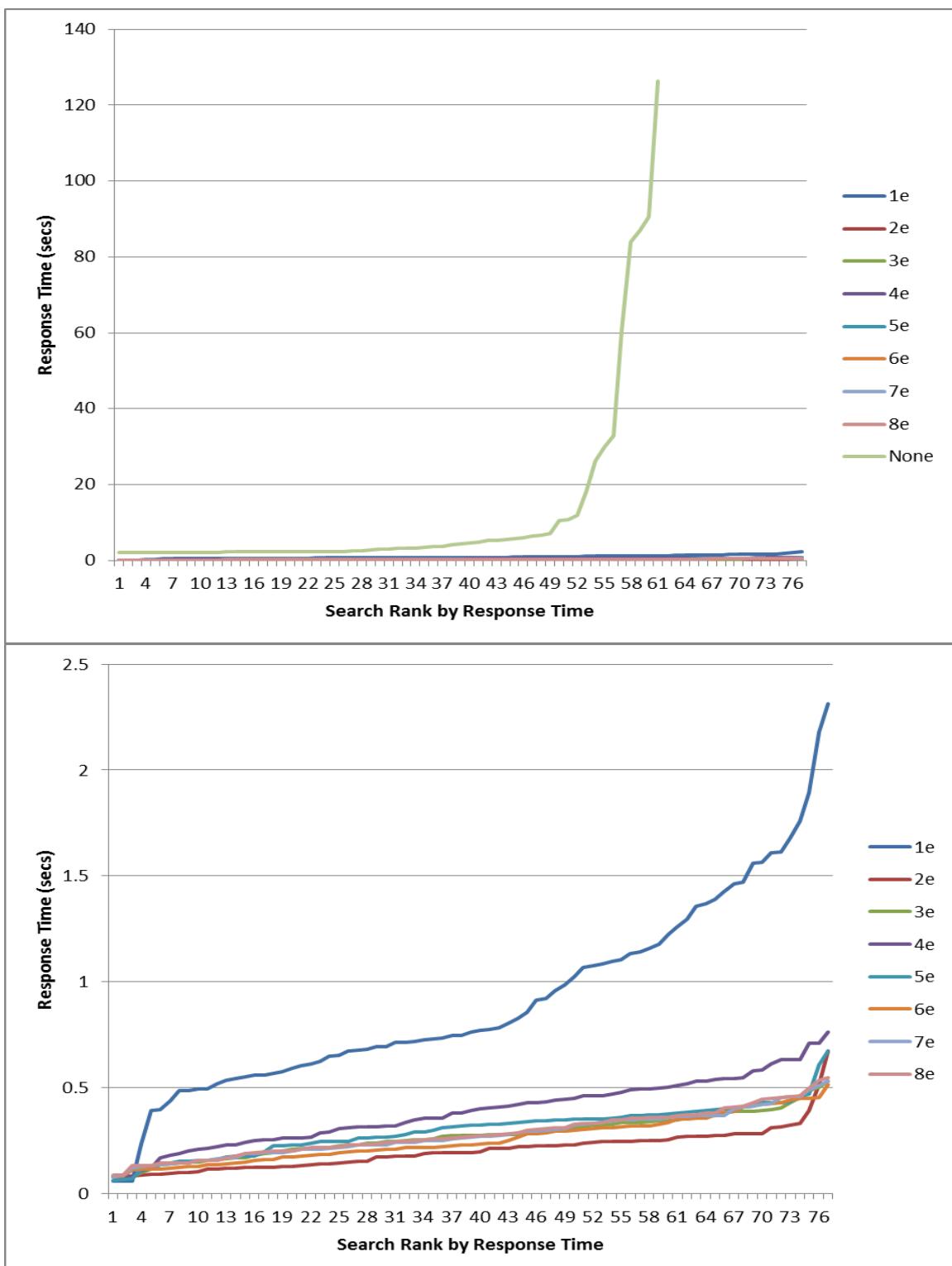


Figure 7.6. Time search suite response times,  $e$  hierarchies: a. All Hierarchies, b. “None” removed

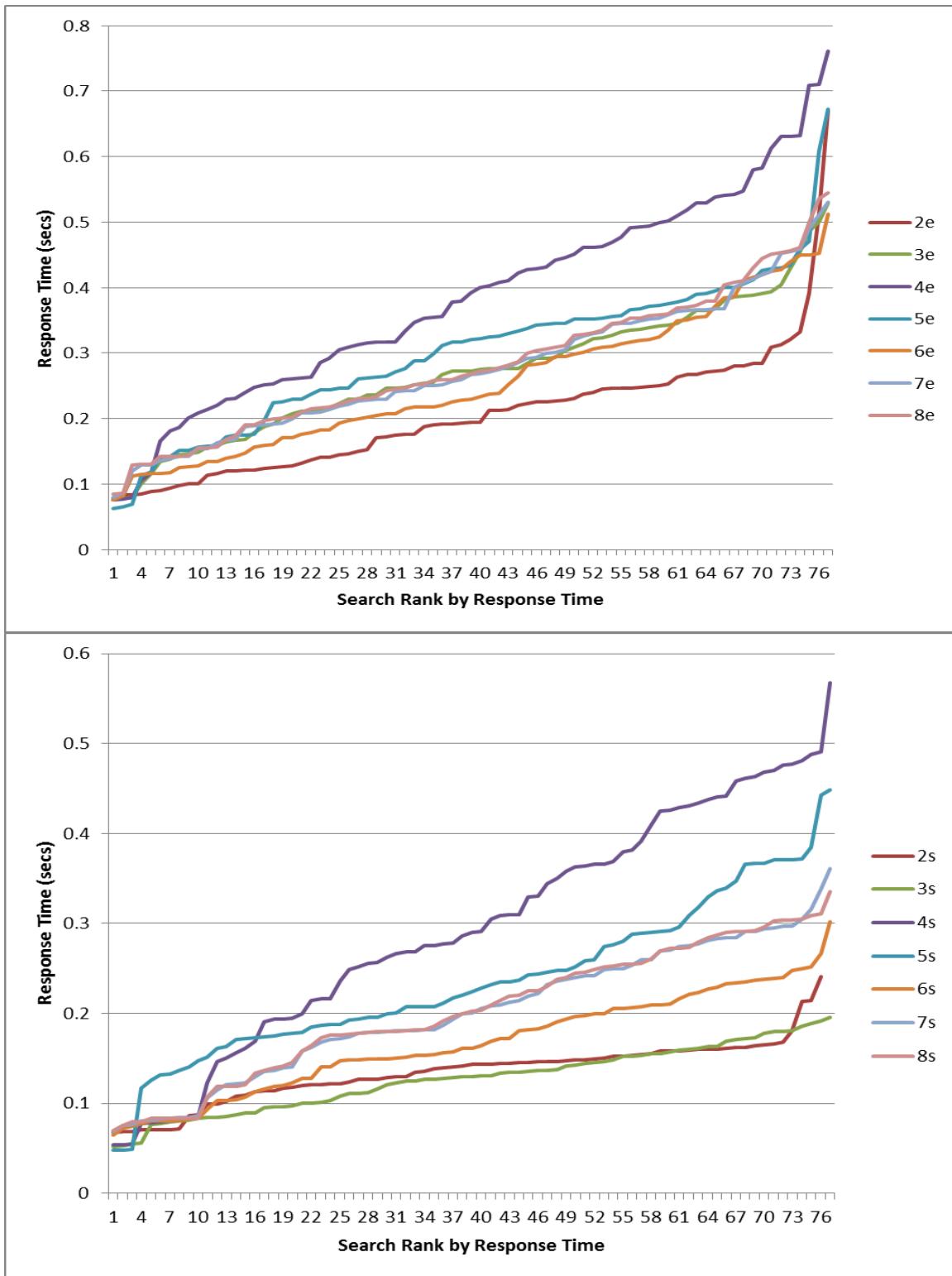


Figure 7.7. Time search suite response times: a. Detail, *e* collection; b. Detail, *s* collection

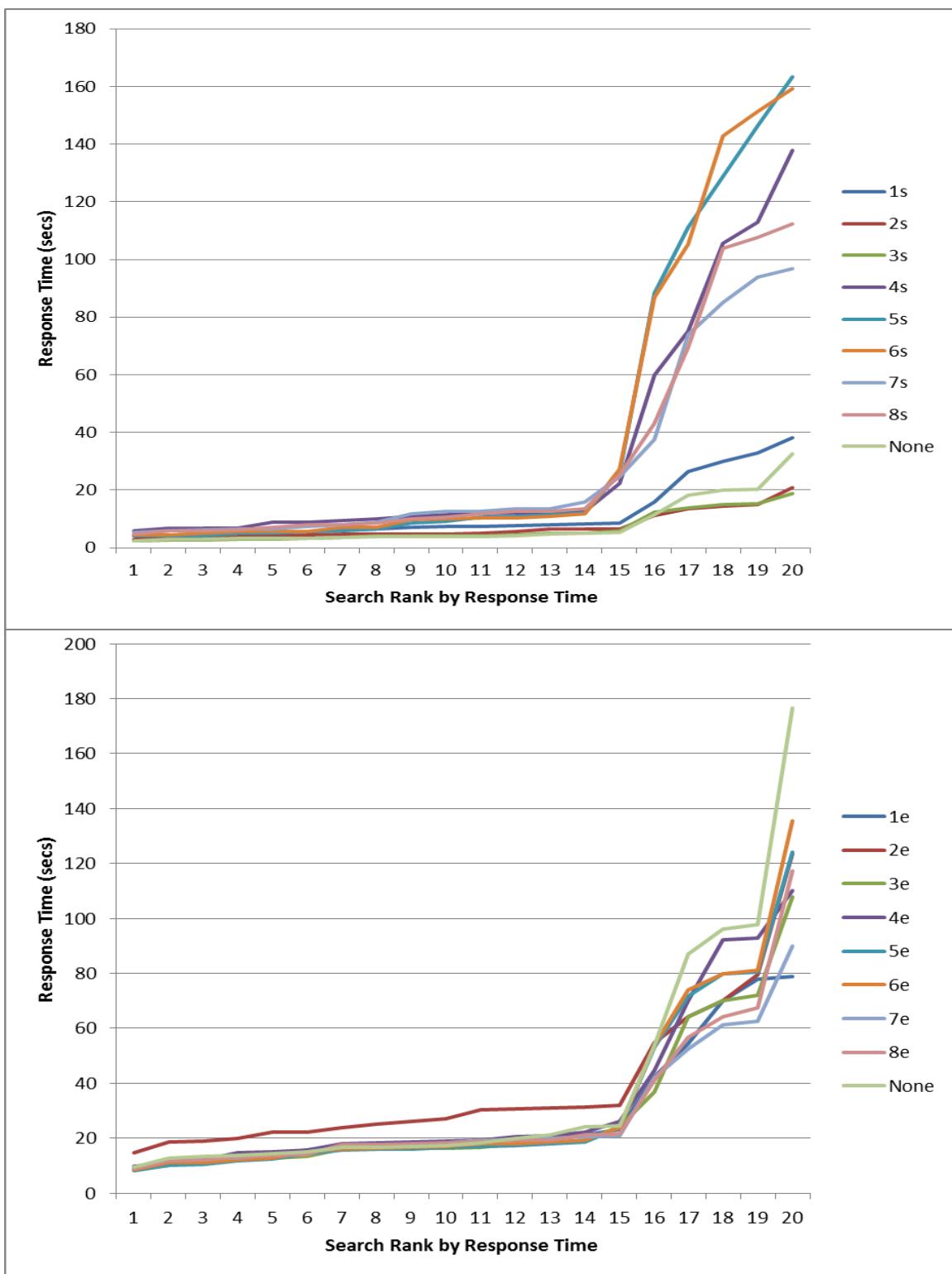


Figure 7.8. Space search suite response times: a. *s* collection; b. *e* collection

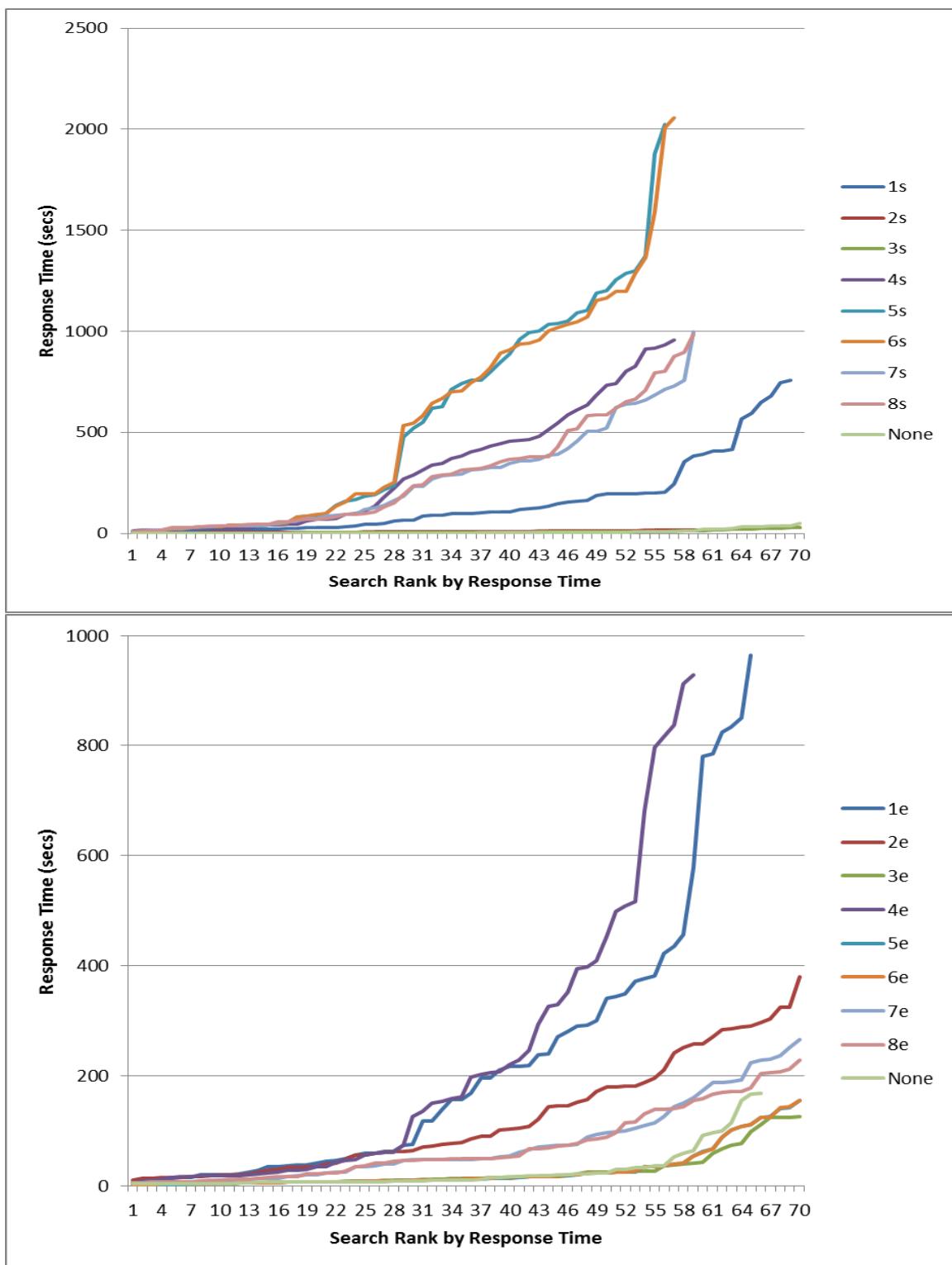


Figure 7.9. Space-time search suite response times: a. *s* collection; b. *e* collection

Figure 7.8 shows the individual search responses, sorted by rank; Figure 7.8a shows the results for the  $s$  collections and Figure 7.8b for the larger  $e$  collections. Unlike the time searches, all hierarchies here exhibit very similar behavior: relatively consistent performance for 15 searches in the suite, and response times increasing by a factor of 3 or more for the other 5 searches.

There are 5 searches that have extremely high response times as compared to the other searches for all hierarchies. All 5 of these searches have geographic areas as large or larger than the block size used in the hierarchy, and span more than one block. (These 5 searches do not appear in the same response time order across the different hierarchies.) The cause of high response times for these searches is the high number of entries processed (from 125,000 to over 350,000); none of these searches have more than one iteration. Removing these 5 searches for the  $e$  collection gives the revised response times shown in the right of Table 7.10; the mean response times excluding these 5 searches are quite consistent across the hierarchies, and also exhibit greatly reduced deviations.

For this search suite, “None” mirrors the performance of the other hierarchies. Hierarchy 2e gives worse performance (approximately double the response time) for the fastest 15 searches; because this hierarchy only splits the space dimension at the leaf level, every entry at every level of the hierarchy must be processed, down to the leaves. This processing overhead increases the response times for 2e over that of “None”, since it visits the same number of leaves, plus all the intermediate nodes in the hierarchy.

For the smaller  $s$  collection, only two hierarchies gave improved mean response over “None”; however, for the larger  $e$  collection, all provided close to the same or faster response times. We attribute this change to the larger collection size; the reduction in number of entries returning from the filter in the  $e$  hierarchies offsets the effect of repeatedly querying the database.

### 7.7.3 Space-Time Search Suite Results

Table 7.11 compares the (geometric) mean response time for each hierarchy, for the space-time search suite. These comparisons show widely differing response times between the  $e$  and  $s$  collections. For four of the hierarchies (5, 6, 7 and 8), response times for the larger  $e$  collection dropped to a small fraction of the response time for the smaller  $s$  collection. This is because with the greater density of data, a subset of searches were able to find sufficient matches in the first or second iteration, while in the smaller collection these searches restarted additional times to find sufficient matches. For the  $s$  collection, in most hierarchies only a subset of searches finished successfully (the number of Successful Searches is smaller than the size of the search suite, which contained 70 searches). Hierarchy 2 had nearly 11 times the response time for the  $e$  collection compared to the  $s$  collection, while “None” and hierarchy 3 had increased response times by a factor of around 3.5; this is less than the increase in collection size of around 4.5. Overall, the response times for this search suite were much longer than for the space suite because many more entries had to be scored for each search in order to reach *minimumMatches*; the response time is dominated by the cost of the spatial calculations.

Table 7.11. Space-Time Search Suite: Response Times by Hierarchy (in seconds)

Hierarchy	<i>s</i> Collection			<i>e</i> Collection		
	Successful Searches	Mean	Std. Dev.	Successful Searches	Mean	Std. Dev.
1	69	160.537	191.850	65	217.896	244.354
2	70	11.046	7.395	70	120.594	101.549
3	70	7.796	7.186	70	26.004	32.828
4	57	314.169	297.273	59	215.340	255.911
5	56	540.728	535.155	70	28.794	39.088
6	57	557.902	546.000	70	29.031	38.942
7	59	268.292	245.729	70	75.721	71.894
8	59	286.295	270.883	70	72.371	63.889
None	70	7.526	10.193	66	26.513	38.738

Figure 7.9 shows the individual search responses sorted by rank; Figure 7.9a shows the results for the *s* collections and Figure 7.9b for the larger *e* collections. In both cases, “None” provides the best (or close to the best) results, although in the case of the *e* collection, 4 searches fail. Hierarchy 3 had the lowest maximum response time for both collections (28.5 seconds for 3s and 126.4 seconds for 3e); hierarchy 4 had the fewest searches that completed successfully, and reported very high mean response times.

For this search suite, individual searches have widely varying response times between the collections; for example, one search has response times varying from 6.6 seconds to 1,285 seconds across hierarchies. The searches that are to the right of the bend on the graph, signifying dramatically increasing response times, disproportionately consist of a time term combined with one of the 5 poorly-performing space terms identified in the space-search-suite results.

## 7.8 Other Informal Performance Tests

The existing data model consists of two primary tables, one containing dataset-level information and the other containing information about variables. This data model makes it easy to add new variables during dataset scanning. The variables table is pivoted (using a contributed PostgreSQL module) during searches that include variable search terms, quickly leading to unacceptable performance as the table increases in size.

We performed some informal experiments of alternative data models using several example searches. In one test, we pivoted the variables table into additional columns added to the files table (for three variables only), thus avoiding the join between the files and variables tables. This approach provided much faster response times than the current data model, as one would expect. As we gain more experience, we may be able to identify commonly-searched variables to include in the files table.

We also experimented with a Bloom-filter-inspired structure [6, 25]. We pivoted the variable table on variable name into a materialized view, and added a binary column to the “files” table for each variable name. The binary value was set to TRUE for an individual entry if the named variable existed for that dataset. We modified the filter to include entries where the variable’s existence would give a total entry score above the cutoff score (assuming a complete match on variable values). Informal tests with several searches showed significant improvements in response times. Since only 1 bit is added to the width of the table for each new variable, this approach captures much of the

performance improvement of avoiding a full pivot of the variables table, without causing the same column explosion as the previous option.

For any substantial set of variable names and using a relational datastore, both the approaches above lead to extremely wide tables. Any additions to the set of variable names require a schema change to add columns for the new variable names. As a result, despite the performance gains, we feel that these data model alternatives are not appropriate replacements for our current design.

The primary driver of overall response time is the level of hierarchy that has the most entries returned by the filter to process. Depending on the search, this level may not, in fact, be the leaf level; there were a number of queries where the level above the leaf level had the most entries returned. In experimenting with the hierarchy designs, we tested the impact of removing the root level of the hierarchy, reducing tree height by one level. For the combinations of search and hierarchy we tried, removing the root reduced response time by around 0.025 seconds for the search. The root level has the smallest number of entries; thus, it appears that reducing the number of levels has far less effect than reducing the number of entries returned from any one level's query.

Our current spatial filter identifies “close enough” entries by taking the distance from the center of the search area to the entry’s spatial centroid and subtracting the maxradius of the entry’s shape (the centroid and maxradius having been statically calculated and stored as part of the dataset summary during summary creation). This filter does not use a spatial index because our geospatial software (PostGIS 2.0 and Postgresql 9.1) only uses

a spatial index for queries involving certain bounding box comparisons not used in the current filter design. We experimented with a different spatial filter design on several long-running spatial queries. In this alternate design, we created an additional spatial column and set it to the bounding box of each dataset’s shape. We modified the filter in the following way to ensure that a spatial index was used: we first calculate an expanded spatial query term “bounding box” representing the area where all points falling within that area would have a score higher than our current cutoff. We then look for intersections between the expanded spatial query bounding box and each dataset’s spatial bounding box. We confirmed that the spatial index was in fact selected by the Postgresql optimizer by using Postgresql query “explain” function. This rewrite reduced the number of rows returned from the database in the test queries by up to 50%, since a maxradius-sized circle around the centroid will often cover a larger area than a bounding box. The result reduced the runtime of the SQL query by approximately one-quarter; we attribute the limited gain from using the spatial index to the resource-consuming calculation required to compare bounding boxes. We estimate that moving to this filter design would reduce response time for the spatial query terms by approximately one-third.

## 7.9 Discussion

Our overall goal in this performance work is to understand, improve and predict the performance characteristics of our approach. In this section we summarize our progress towards this goal, and comment on additional research required.

### 7.9.1 Effect of Filter-Restart Techniques

We showed that as the metadata collection size increases, our filter allows us to complete searches that would otherwise time-out or exhaust memory. Further refining the filter and starting cutoff score will provide additional benefits; it may be possible to remove the need for most restarts through improved theory or heuristics. We experimented with a modified filter for space searches that may provide additional benefits by allowing use of spatial indexes; however, the improvement of 50% achieved is less than the order-of-magnitude improvement needed to make a substantial difference to the scalability.

The implementation of the filter, in turn, requires firstly selecting a starting score, and secondly adding a restart approach for the times when the initial cutoff score is too high to return the desired number of matches. We chose a default starting score for our work. Bruno et al. use database catalog statistics to estimate the initial starting score. In each test, they make an assumption about the relationship between the distributions of multiple attributes; commonly, the distributions are presumed to be uniform and independent [84]. However, Bruno et al.'s experiments show that applying the uniformity assumption within histogram buckets was computationally expensive and led to many restarts [18]. In all three search suites, less than one third of the searches required restart. For the time search suite, restarts only occurred in one hierarchy. For searches that required restart, in all three suites Adaptive (which estimates a revised cutoff score based on the matches found so far) reduced the number of restarts required by half or more over Naïve (which uses a fixed revision to the cutoff score). For affected searches, Adaptive reduced

response time by around half, and reduced the variability of response times across the searches within the search suite by around half as well. Thus, Adaptive relaxation achieves its goal of improving performance over Naïve.

We also experimented with “Contraction”; that is, successively increasing the cutoff threshold within a single iteration as the number of desired matches is reached. This technique can be applied to the initial filter as well as during restarts. For all three suites, adding Contraction reduced response time and variability; the improvement was around 20-25% for the affected searches. However, the improvement gained by adding Contraction was lower than the improvement of Adaptive over Naïve.

### 7.9.2 Effect of Hierarchies

We showed that hierarchies can, depending on their organization and on the specific search, further improve performance compared to having no hierarchy. For most hierarchies compared, the increase in response times between the  $s$  and  $e$  collections (and  $t$ , for the tests run there) was, for most searches, smaller than the ratio of collection sizes. This reduction supports the added value of taking advantage of hierarchies to improve response times as collection size grows.

For the subset of searches where low-density data was encountered (and therefore, a smaller number of children at each hierarchy level using the chosen hierarchy structures) in the smaller collection, the response time was higher than for the same search against the larger collection. For the subset of searches where high-density data was encountered in the larger collection, response times were much larger than the relative increase in

overall data sizes. These comparisons show the sensitivity of search performance to local data density per search. It is challenging to design a filter-restart method that can handle both high-density data (leading to too many matches, causing poor performance) and low-density data (leading to too few matches, causing poor performance from many restarts).

Failed searches (exhausting either memory or PHP processing time of 1,000 seconds) were experienced only in the space-time suite and in the *e* collections; the specific failed searches varied according to the hierarchy. Further investigation showed that for each of the failed searches, there was a level in the hierarchy where the SQL query returned a large number of rows after applying the filter. The SQL query itself took several seconds to perform the query; the number of results returned from SQL to PHP sometimes exhausted the PHP memory available to the search engine task; and in some cases processing the results of the query took up to 10 times the query time for a space search term. While we keep a count of the number of children each entry has, we cannot predict the percentage of these children that will survive application of the filter. Applying heuristics based on the number of children to which the SQL query will apply the filter (which the current data model allows us to calculate) would allow us to know that we are likely to have a long-running query, and to adjust accordingly. For example, we could combine information about the parent's score, bounds and number of children, and adjust the threshold based on their distribution (if a high scoring parent has many children, we

could further raise the cutoff threshold). We also suspect that additional refinements to our algorithm could avoid processing some of the returned rows.

The mean response time was greatly affected by the difference in the types of search terms the search contains and by the hierarchy used, as can be seen in Table 7.12. While all time-only searches in the suite performed within normal user expectations (sub-second or a few seconds at most), space-only and space-time searches did not.

Table 7.12. Hierarchy Comparisons Across Search Terms: Best and Worst for *e* Collection

	Best (s)	Worst (s)
Time Only	0.34±0.28 (8e)	4.95±23.98 (None/e)
Space Only	27.75±21.97 (3e)	38.30±43.35 (None/e)
Space-Time	26.00±32.83 (3e)	217.90±244.35 (1e)

The variability across the hierarchies was far greater than the effect of the relaxation techniques tested. For space-only searches, Hierarchy 3e provided the best average response, with around two-thirds the response time of the worst, “None.” For the space-time searches, the worst hierarchy, 1e, had 8.4 times the average response time of the best hierarchy, 3e. These comparisons show that in many cases, the hierarchy can improve performance for some queries, providing lower response times than a linear scan of the entire metadata collection.

An important question, given the widely different performance results from the search suites, is how to predict which hierarchy will provide the best – or at least acceptable – performance for the range of searches that users may perform. These hierarchy design choices could then inform the archive curator when adding new data; for example if two data segmentation strategies equally met the users’ expected needs but had very different

performance characteristics, the curator could select the one that would provide faster response times.

We explored a range of hypotheses for how to predict the best hierarchy to use, but without clear success. The best performing hierarchy varies according to the type of search terms and the density of data close to the combination of terms. Given the absence so far of a good method for choosing an appropriate hierarchy (when several options meet the users' needs), several options are possible. For example:

1. Have different types of hierarchy, in different sets of tables, as we did in the performance study. Start a search on each hierarchy and provide the response from the hierarchy that comes back first, canceling the others.
2. Choose a single “good enough” hierarchy approach as a default, and continue to develop other performance methods (such as the filter-restart approaches) to maintain performance in acceptable ranges.
3. Keep every set of metadata tables small enough to give fast response from each table and add a search integrator to combine the top-k from across the various tables. This approach might allow one to mix different kinds of hierarchies in any one of the tables.
4. Provide partial results back early once a search with long response times is recognized (i.e., many results are returned from an SQL query, or a restart is required), and continue to refine them as more results become available.
5. Perform additional research to identify methods to predict hierarchy performance.

In summary, our research echoes the findings of Bruno et al.’s experiments for setting starting scores: “no strategy is consistently the best across data distributions. Moreover, even over the same data sets, which strategy works best for a query  $q$  sometimes depends on the specifics of  $q$ ” [18]. Similarly, our hierarchy tests mirror the results of Billerbeck and Zobel [15], who focus on testing document ranking performed using the Okapi BM25 measure, which requires several parameters that they initialize by using values determined in experiments on a particular test data set. The key parameters are then fixed and used during additional query expansion experiments. They find that no fixed choice is robust across different collections; entirely different values give the best result on different collections, and “worse, the best choices per query vary wildly.”

### 7.9.3 Scaling Beyond a Single Server

As noted, we wish to understand the performance of a single server; if single-server performance is sufficient for the expected workload, no additional scaling approaches are needed. When single-server performance is found insufficient even after optimization, throughput can be further scaled by adding servers. The scaling design must consider how the workload is to be distributed across these servers. Tomasic and Garcia-Molina [130] summarize the two basic strategies for distributing an inverted index over a collection of servers: so-called *local inverted files*, where each server is responsible for a disjoint partition of the documents in a collection, and so-called *global inverted files*, where each server is responsible for a disjoint partition of the terms (but across all documents). They simulate performance of different configurations of global and local

indexes and conclude that local inverted files provide the best performance. The performance of the servers containing the disjoint index partitions therefore limits overall system performance. While we are not using inverted files, the same two segmentation approaches apply (segment on entries or on attributes); we would expect to see the same result, that segmentation on entries provides better performance.

Building on their work, Cacheda et al. [19] and Long and Suel [80] describe the use and performance of “query integrators” to distribute queries to and combine results from multiple servers with local inverted files. Cacheda et al. [19] compare the performance of searching a terabyte of text on three different multi-server architectures: distributed, replicated and clustered. Depending on the search characteristics, either the replicated or the clustered configurations performed better. Latency is increased relative to single server architectures by the need to route the searches to or aggregate the results from the multiple servers (depending on the approach taken), while throughput of the entire system, that is, the number of searches that can be handled simultaneously, is increased. We would expect to see the same effect of increasing response time (but with added throughput) from adding search integrators to our approach, relative to a single system.

The recent move to cloud-computing infrastructures has led to new approaches that perform well at larger scales. For example, Wang et al. [140] describe a multi-dimensional indexing scheme, RT-CAN, that efficiently supports multi-dimensional query processing in a cloud. RT-CAN creates a distributed global index that is used to identify the local nodes containing relevant data. The distribution approach may be based

on geographic location or on data ranges. They demonstrate their system using point, range and nearest-neighbor queries, using a 128-node network. We believe this scaling approach applies to our work.

#### **7.9.4 Current Deployment**

In our current archive, we use different hierarchies for different types of data, mixing all of them in a single metadata collection. Thus, a single search may be traversing many different hierarchy shapes simultaneously. The majority of searches occur in areas with the greatest data density, since more data is collected in the areas with the greatest research interest; thus, relatively few searches suffer from the “data scarcity” issue that causes slow response times for some searches over the *s* collection.

In the currently deployed version of the search engine, we use the filter with Adaptive relaxation and Contraction, and the centroid-and-*maxradius*-based spatial filter. All search types perform well for our current catalog size, and for the expected organic growth of our catalog over the next 5 years.

To support larger archives, we will need to modify the data model we use for variables as we described in Section 7.8. We must also learn to distinguish and accommodate hierarchies that lead to long response times, using techniques such as those identified in Section 7.9. Again, there is the opportunity for further improvements identified via additional research.

## **8 Future Research and Conclusions**

The research described in this dissertation consists of a concept of how large archives of data can be searched; a search model; a prototype showing the ideas and model are feasible; user studies supporting the utility of the ideas to the target user community; and initial performance work that identifies some successful approaches and areas for additional investigation.

At the same time, the work we present is a limited proof-point; there are several research issues that we identified during this work but have not addressed. As we continue to work with the scientists and with the initial prototype, we develop additional understanding of the use cases, requirements, opportunities and challenges in this area. In this chapter, we discuss some of these potential areas of research.

We describe needs and ideas in the areas of feature extraction (Section 8.1); managing metadata (Section 8.2); the issue of variable-name diversity (Section 8.3); increasing the sophistication of search capabilities (Section 8.4); and the goal of universal data search (Section 8.6). We present conclusions in Section 8.7.

### **8.1 Feature Extraction**

Within Data Near Here, we focused on developing scanners for a small but representative set of scientific dataset formats (CSV files, NetCDF files, relational database tables, text webpages used to report data for several specific instrument types). We are working towards complete coverage of CMOP's current data holdings, while keeping up with new observation capabilities. In some instances, new sources are incorporated with no extra

human action, such as for a deployment of an additional sensor gathering data of a previously seen type. We are currently testing a scanner for CMOP’s analytic models, the largest remaining segment of the archive not currently included in DNH’s metadata collection.

We are beginning to incorporate datasets from other sources into the collection, allowing users to search for data across multiple organizations’ archives. We do not need to host the data to provide our search service – we only access it with an appropriate scanner and build summaries for our collection. Some of these sources are easy to incorporate – for example, when an archive collects similar kinds of measurements, such as temperature and salinity, but at another location, or uses similar data formats. For other sources, we have to create new feature extractors, but the data is comparable enough that we do not need to change similarity functions or search terms; for example, some observation points provide their data in columnar text format on a web page, which we read and summarize. At present, we treat a numeric environmental variable’s readings (as represented by the values in the dataset) as being equally distributed between the bounds (or, alternately, the bounds after having eliminated the highest and lowest observation), as is common practice in database indexing [84]. Some other distribution assumptions may be equally valid; alternatively, it may be appropriate to identify different distributions for each variable for at least some types of data. For each distribution assumed, a matching similarity function is required; further, the *Match* function must be modified to select an appropriate similarity function to use with each distribution. If different distributions and

similarity functions are used in a single search, interactions between the distributions and functions may produce a variety of rankings. User studies should be performed to validate that the result of rankings using such similarity functions and combinations match user expectations.

As we expand into working with other kinds of data, we will wish to aggregate different kinds of features, depending on the type of data involved. Features for gene sequences will need to be handled differently from features for species names. These different types of features will need to be associated with appropriate similarity functions.

We believe we could build an archive crawler that could take, for example, a base URL for an archive, then build metadata entries for all (recognizably formatted) datasets accessible from that base URL. A longer-range possibility is to embed the ideas from these scanners into a modified web crawler that could identify and crawl scientific archives, similar to the way that the Internet is crawled today.

## **8.2 The Metadata Mess**

Some archive owners are sensitive to having their archive's metadata exposed to the external world. As we found with CMOP [90], once we provide an easily-readable list of the variable names and units in use within an archive, we expect that archive curators will wish to normalize or repair at least some of the contents in their archive. We wish to provide them with the ability to request a rescan of those sections, and then provide tools so they can review and validate the results.

We also expect that there will be sections of the archive for which repair or normalization will be not possible, for technical or resource reasons. For these sections, we believe curators desire the ability to create a “gloss” over the variables and units; that is, they would like to provide a mapping from the actual names (as they exist in the datasets) to corrected names (that they wish existed in the datasets). We developed a proof-of-concept of such a mapping capability [90]. In some cases, more than one name may even be desired, if different metadata or naming standards are in use. Based on our proof-of-concept, we believe that it is possible, with some additional development, to provide such a capability.

Some of the metadata standards commonly in use contain a mix of *contextual metadata* (ownership, terms and conditions, etc.) and *inherent* (derived from automated analysis of the data) metadata items [61]. In future work, we would like to provide a method for an archive to identify a relevant metadata standard, and then identify which fields can be automatically generated from the dataset collection. Contextual metadata, which is often the same or substantially the same for entire groups of datasets, could then be requested from the archive curator to fill out the required data. Archive curators should also be able to add to or modify information gained via the feature-extraction process. For example, a curator might add collection-level information to a set or collection of datasets within the archive, such as contact details for the responsible party, quality annotation, or usage restrictions.

As part of these capabilities, we would like to automatically segment datasets and create metadata hierarchies over them. We believe it is possible to develop a set of heuristics that can produce “good enough” results for many situations; for example, segmenting by time and space, by segmenting highly variable data into finer granules, or by adapting tools such as principal component analysis. In addition, we would expose an interface for archive curators to use to specify an alternate segmentation or hierarchy if desired – possibly at the cost of having to rescan the relevant datasets in order to implement the new parameters.

We have also considered the possibility of using crowd-sourcing to assist with cleaning archive metadata; for example, allowing scientists to tag a variable in an archive as being “the same as” some other variable or concept; and then using the tags to enhance search results or search quality.

We envisage moving to a two-catalog approach, with a “work” and a “production” catalog. The work catalog would contain the results of scanning and allow additional transformations to be applied to the results, if desired. Individual archive curators may apply different workflows to the data representing their own archives. Summaries would be promoted to the production catalog after moving through the appropriate workflow. For some summaries or for some archives, the workflow could be a “null” workflow, allowing summaries to move directly to the production catalog. Thus, archive curators would be afforded the opportunity to clean their metadata, but it would not be required.

### **8.3 The Variability of Variable Names**

We have discussed several times the challenge of variable-name diversity. The work described in this dissertation makes the simplifying assumption that a scientist can accurately name (or recognize, if presented with a list) a desired variable, and we address the issue of comparing the similarity of that variable's data range to a desired data range (potentially including translating units).

At a coarser level of detail is the problem of identifying whether the variable in a dataset is, in fact, the desired variable. In practice, diversity in variable names is a significant problem. This issue is in itself an unsolved research problem; we believe the issue of matching variable names can also be seen as an information-retrieval problem.

While this challenge exists within a single archive, it compounds when considered across archives. Even in research fields for which metadata and naming standards exist, the challenge of updating variable names in an archive to one or more standards – or even to a single standard that changes over successive versions – is difficult for archives, most of which have extremely limited resources.

We believe that there are several research problems to address within this topic of variable-name matching.

Firstly, there is a set of translations or normalizations that an archive curator would be aware of, if given the opportunity to see in a central place the diversity of variable names in their archive. We have experienced this process at CMOP, where a scan of the archive brought to light the numerous ways that, for example, “water\_temperature” was spelled

across his archive; we also experienced the effort involved in trying to reduce and normalize these variables. Work is underway to address the problems of normalizing the variable names in the archive [90]. Our approach is to distinguish the raw harvested metadata from that shown to the user, and to provide various methods for the archive curator to map from raw names to the desired variable names for their archive. There are multiple places in the overall harvest-and-search process where the adjustments may be made [90].

However, we believe that even if every archive curator were able to present the metadata she wishes she had, we would still be left with a variable diversity problem on the side of the searchers. That is: searchers themselves are not consistent in naming the variables they search for. We believe that this problem can be segmented into several sub-problems, and that each of these has different sources and must be addressed using different techniques.

One source is the problem of synonymy, that is, using different terms for the same thing. The search engine would ideally be able to recognize and return data stored under a synonym. For data, synonymy can be extended to include unit translations: recognizing that the desired data range is in fact present in a dataset but stored using a different measurement unit. There are multiple approaches to semantic similarity that might apply here. Amongst other possibilities, Schwering distinguishes transformational distance (where distance is measured via the number or complexity of transformations required to transform one concept or object to another) from path distance (such as the length of the

shortest path between two nodes in a tree such as an ontology relating terms in a domain) [118]; the most effective (from a search user perspective) semantic similarity approach for addressing synonymy for units is likely to be different from the best approach for variable synonymy.

A second source is the difference in research fields amongst scientists; an oceanographer may use a term differently from a microbiologist, say. This problem is also found in text retrieval, in the problem of polysemy (multiple meanings for the same syntactic term). This difference may be amenable to searchers giving some hints as to their source context, perhaps as part of an advanced search interface; latent semantic indexing techniques [31, 55] may be useful here. For data-search engines to be useful, both the archive and the user must be able to specify which context they intend, and the search engine must be enhanced to estimate similarity across contexts. Thus the matching of search terms to variables would not be fixed, but would depend on the searcher (and perhaps on the search). Ideally we would like to figure out the matching with the minimum of effort on the user side, avoiding profiles or additional dialogues with the search interface; as with our work so far, we are curious to see how far we can go using simple approximations, and perhaps handing the filtered result to a more sophisticated tool if desired by the searcher.

Another source of diversity is the use of multi-level concepts. For example, fluorescence may be measured at different wavelengths and stored as separate variables in a dataset: *fluores375*, *fluores400*, etc. For a microbiologist studying the data, each of these

wavelengths is a separate variable. For the oceanographer, all wavelengths may be thought of as a single variable called “fluorescence.” Likewise, ocean modelers often regard *surface\_temperature* as a variable distinct from *water\_temperature*, since it represents a boundary condition of inputs from external influences (wind, sun). In essence, such situations are manifestations of property precedence, as described by Parsons and Wand [106], where attributes that appear different at one level can be regarded as the same at a more abstract level. We also note that a scientist may move through several phases of concept detail when searching for data. She may begin with a more general search, while trying to assess what data is available: is there any fluorescence information available, and if so, what kinds? On finding some, she becomes progressively more selective. This challenge exists in other fields; in our examination of a sensor archive of vehicular traffic data in an urban setting we identified the same issue. During a search, multi-level concepts present a number of challenges. Identifying what kinds of data might match a search is itself a matching problem, subject to some kind of (as yet undefined) similarity model and (presumably) estimable via some function. This variable-similarity function then overlays the data-similarity function developed in this dissertation. However, we posit that some rough simplifications could be applied here as well; for example, higher-level searches could be treated as existence searches only; or, a detailed search that only identifies low-similarity data could propose some alternate, similar terms that the search engine identifies as resulting in higher-similarity results.

For each dataset, we first match each search term to a feature of this dataset, or to no feature (performed by function *Match*). Our initial matching function, used in the user study and reflected in the pseudo-code herein, simply looks for an exact match between the variable named in the search term and a named column in the dataset. (This approach is naïve and is not mandated by our model.)

In addition, variable units are not currently standardized. Here, again, we have a diversity problem. Often, a variable can be reported using one of several different measures (such as, distance in millimeters, meters, or even feet), or with a single unit named in different ways (C, °C, Centigrade). In other cases, different measures are not directly translatable. In some cases, a variable may be misspelled (c instead of C), or unrecognizable. We believe that, as with variable diversity, it is possible to characterize kinds or sources of unit diversity and identify techniques that apply to each. Within Data Near Here we have experimented with several approaches, with promising results. We initially worked with the archive curator to correct and regenerate affected portions of the archive, with good results but with an increased understanding that this approach is only a partial solution and is not sustainable given the data growth and the commonly-experienced personnel resource constraints for such archives. In the search interface, we provide a list of the units found for each (identically) named variable. We have experimented with unit translations for some units. Where the units of the variable are unknown, we assume the variable is in the desired units, apply the distance measure, and then discount the result by a factor representing some level of uncertainty. As with variable-name diversity, some

of the cases may also be amenable to the “gloss” or hiding by applying rules in the metadata collection, while leaving the underlying data as is.

Based on an initial analysis of sources of the diversity, it is likely that a variety of approaches will need to be applied to address the different aspects of matching variable names found in datasets to variable terms [90]. Ordering the search results will involve balancing similarity estimated at different levels of detail: for example, the similarity of the variable to the desired variable, along with the similarity of the data ranges.

#### **8.4 Increasing Search Sophistication**

Our initial research focused on numeric data search, as that form of data currently represents the overwhelming majority of our archive’s contents. As we expand our coverage, we will wish to handle more combinations of numeric and textual data and searches. Combining and weighting textual, numeric and other types of search terms (such as, for example, DNA sequences) remains an area for future research. While it is mathematically possible to combine scores from these two methods – for example, by including the score for each textual term in the final score normalization – we do not yet have a model for how scientists perceive these combinations. In particular, unlike the continuous numeric and the existence measures, we have not validated these additional approaches with formal user studies. There are technical issues to be addressed, but we believe the most pressing issue to be more user studies exploring how users expect these systems to operate.

For some searches, a large number of entries are found that are “perfect matches”, or have the same score (within some very small delta). Currently, entries with the same (rounded to integer) score are ordered from the entry containing the most observations to the entry with the least, on the assumption that more matching observations are better. However, we have no evidence to support this assumption; it may be that scientists consider entries with observations covering the entire search range as being the best fit for their information need. For example, when specifying a search area, a grid covering a large proportion of the search area may be a better fit than a single point with many observations. Again, we suggest user studies to discern whether there is a general preference amongst scientists that one could emulate (recognizing that no option will be the best choice in all situations).

The current search interface design is naïve; alternatives should be explored. Additional desirable capabilities include the ability to remember a user’s searches for later repetition, and to register “standing searches” that will notify the user when new matching (above some score) datasets are added to the catalog.

There is also the opportunity to provide more expressive search capabilities. However, research on Internet search has shown that even advanced users rarely use advanced search interfaces when they are available, preferring to continue using brief lists of search terms [66, 143]. Therefore we believe that research is required into what search capabilities are in fact useful before effort is invested here.

Recently, sophisticated search engines are taking word co-occurrences and context into account in their similarity scoring functions [78]; identifying ways to apply these concepts to dataset similarity is left to future research.

Longer-range, we would like to suggest combinations of datasets to meet a scientist's information need; for example, a search for oxygen and nitrogen data in a given location and time-period could suggest that two datasets, one containing oxygen data and one containing nitrogen data, could perhaps be combined; the combination might achieve a higher score than any individual dataset currently available. Ideally, we would be able to suggest data from multiple archives that could be correlated or integrated to produce new insights [49]; or, more generally, to support virtual datasets for which we could construct metadata in advance, then synthesize the dataset contents on demand.

## 8.5 Scalability

As noted in Chapter 7, more scalability research is required should we need to support much larger metadata catalogs. Alternate architectural choices and data models provide one possible avenue of research. We clearly showed that some hierarchy and search combinations lead to long response times; we also noted that, in the search suites and hierarchy combinations tested, there were a small number of searches that had the longest response times, and these searches tended to be the same across all the search suites. We believe additional research into identifying the characteristics of these searches and identifying methods targeted at improving their response times could provide substantial benefit.

## 8.6 Towards Universal Data Search

We see little difficulty in deploying another instance of DNH at a different ocean observatory. The main additional work would be creating or adapting metadata extractors for datasets unlike those at CMOP, and making decisions about how to hierarchically decompose dataset collections. But what about using DNH for other scientific disciplines? In its efforts to ease data sharing, the oceanographic community differentiated between oceanography-specific issues and discipline-neutral problems, such as data access [27]. In the same way, we have differentiated between oceanography-specific aspects of our implementation (the names of the environmental variables, the details of the hierarchy, the exact similarity function) and the discipline-neutral approaches and ideas, and have favored neutrality where possible. Thus, we believe that the overall architecture – and significant portions of the implementation – of DNH would readily carry over to other scientific repositories with a size and scope similar to CMOP. Other domains are likely to require new or modified similarity functions, and possibly additional ways to specify search terms and display results.

Ideally, we would like to push our approach further, to be a universal search engine for locating relevant datasets across all scientific disciplines – the dataset equivalent of web search. There are significant challenges to realizing this vision. While CMOP is multidisciplinary, we have had success with a more-or-less-fixed mapping from search terms to dataset features. A general scientific search engine might have to do this mapping dynamically, on a per-search, per-variable or per-domain basis. The issue is not

just which physical phenomenon (temperature, pressure, velocity) is being measured or modeled, but also what entity is manifesting it. (Does “temperature” mean “water temperature” or “air temperature” or “star-surface temperature”?)

We are curious whether our approach can be extended to all data search. Our user study results are very positive; however, it is possible that these positive results were related to some particularities of our use case. For example, our use case differs substantially from the case of tables extracted from HTML, as explored by several researchers [20, 138, 144]. We characterize these differences as follows:

- We work with datasets of observations, with each individual dataset having a relatively simple and consistent internal structure, and with each row having a similar semantic meaning as other rows in the dataset. As a result, it is possible to summarize a large dataset by a relatively small and simple summary. This simplicity and regularity is not necessarily present in datasets found in HTML, where HTML may be being used as for presentation formatting in addition to (or instead of) representing the schema of the contained data.
- Our scientists have a simpler pattern of information needs than the wide variety of searches posed to web search engines. We suspect that the scientist’s current formulations of their information needs may be constrained by the capabilities of the data extraction tools they currently use (such as using a Python script or Matlab expression to extract data from a large array). We exploit this relative regularity.

Searches on the web potentially address a much wider set of topics and variety of information needs.

- As noted in Chapter 4, the summaries we create match quite closely the way the scientists describe the datasets they have or are looking for. As a result, there is a natural and direct translation from the scientist's information need into a dataset summary that represents that dataset. We then apply a similarity measure to that ideal summary and the summaries stored in our metadata catalog. In the case of web search for data, the user's information need may not be expressed in a way that is addressed by the way the data is formatted or presented. For example, a user may search for data in the Pacific Northwest, but a table that lists the desired data for each individual state may not have a summary for the Pacific Northwest.
- Scientists don't have a "better" alternative for finding datasets that match their information needs. Two of our key questions in the second user study were: "Was using this tool quicker than finding the most relevant results by other means?" and "How valuable are the search results versus time expended?" In both cases, we received overwhelmingly positive results. In contrast, many web search engine users can expect to find answers to their specific questions in a text sentence in a document ("what is the capital of Tanzania?" ... "The capital of Tanzania is ..."), and these documents act as a competitor to the information in web tables. Thus, the lack of viable alternatives for the scientists positively influenced our results.

- In our user studies, our users were already working with their archive of interest (CMOP), since CMOP tends to collect data that is relevant to the studies that CMOP scientists are interested in performing. Therefore it was very likely that our archive contained data relevant to their research. As Data Near Here is broadened to cover more archives and its reach extended to scientists less familiar with what data might be available in the covered archives, it is likely that issues such as semantic matching and appropriate similarity measures (as described elsewhere in this chapter) will become a much greater issue.

We believe that exploration of these differences could provide additional insight into improving data search in scientific and non-scientific fields. The continued increase in the quantity and complexity of multidisciplinary scientific research is making it difficult for scientists to find data by navigating individual portals for each type of data they desire. We believe there is the need for a general approach such as ours to scientific data search. We believe that many of the research issues that we have identified were probably raised at the beginning of text search. Research to resolve these doubts have led to the plethora of text search techniques in effective use today.

## **8.7 Conclusion**

Just as Internet search engines began with simple functions and have become progressively more sophisticated and complex as we better understand and can automatically replicate user search behavior, we believe that much more progress can be made in helping searchers locate relevant data. Some big-data projects store output from

just a single instrument (telescope, collider), and scientists can fairly readily locate the data they need. But in other cases, such as ocean observatories, big data means many datasets, of multiple types, possibly in more than one archive, and navigating among them to find relevant data is a challenge.

This dissertation presents a novel approach to this problem by adapting ideas from Internet search. We scan datasets, potentially partitioning them, and create a catalog entry summarizing the contents of each. We allow compound geospatial, temporal and variable searches across a collection of datasets containing numeric data; search terms are scaled to adjust for differences in the search ranges, data ranges and units. Search results consist of datasets ranked by relevance and presented in real time. The approach combines hierarchical metadata extracted from the datasets with a method for comparing similarity between datasets and a scientist's search. This approach complements existing visualization techniques by allowing scientists to quickly identify which subset of a large collection of datasets they should review or analyze. The combination of dataset summaries, using the metadata for search, the hierarchical metadata design and overall loosely coupled architecture allows for scalability and growth across large, complex data repositories.

We believe these concepts complement rather than replace other, domain- or task-specific tools. Our end goal is two-fold: to allow a scientist to locate within the “data deluge” the best-fit, most-promising datasets to analyze in the limited time she has available; and to

provide tools so that any interested archive owner could present to the world the metadata he wishes he had, instead of the metadata he currently has.

We are encouraged by our experiences in applying IR techniques to dataset ranked search, and by the enthusiasm of the scientists for our work. We believe our techniques have broad applicability, and that as data volumes and heterogeneity grow, the need to apply the concepts such as those we have developed here and for tools such as that prototyped in “Data Near Here” will only increase. We believe that scientists really are looking for a “Google for data”, and that our work is a step in that direction.

Large archives of data only have value commensurate with the use and reuse that can be made of their contents; and data cannot be used if it cannot be found. The harder it is to find data, the fewer questions are asked [49, 142]. If relevant data cannot be found in the archive even when it is stored there – if the data is “lost” – the archive’s value is diminished. With the constant need to achieve more with fewer resources, tools such as ours are required to reduce the overhead associated with locating, downloading and segmenting data that is experienced in current scientific research.

By adapting techniques first developed for similar challenges in the world of text documents, we believe that – at least for scientific data – what was lost, can still be found.

## References

1. Abraham, R., Erwig, M.: UCheck: A spreadsheet type checker for end users. *Journal of Visual Languages & Computing.* 18, 1, 71–95 (2007).
2. Ageev, M. et al.: Find it if you can: A game for modeling different types of web search success using interaction data. *Proceedings of SIGIR.* (2011).
3. Agrawal, R.: Data externality. Presented at the CIDR , Asilomar, CA (2011).
4. Agrawal, R., Srikant, R.: Searching with numbers. *IEEE Transactions on Knowledge and Data Engineering.* 15, 4, 855 – 870 (2003).
5. Ahrens, J.P. et al.: Data-intensive science in the US DOE: Case studies and future challenges. *Computing in Science Engineering.* 13, 6, 14 –24 (2011).
6. Almeida, P.S. et al.: Scalable Bloom filters. *Information Processing Letters.* 101, 6, 255–261 (2007).
7. Aula, A. et al.: How does search behavior change as search becomes more difficult? *Proc. of the 28th International Conference on Human Factors in Computing Systems.* pp. 35–44 (2010).
8. Baeza-Yates, R., Ribeiro-Neto, B.: *Modern Information Retrieval.* ACM Press New York (1999).
9. Baker, K.S., Chandler, C.L.: Enabling long-term oceanographic research: Changing data practices, information management strategies and informatics. *Deep Sea Research Part II: Topical Studies in Oceanography.* 55, 18, 2132–2142 (2008).
10. Balke, W.-T. et al.: Applications of quick-combine for ranked query models. *DELOS Workshop: Information Seeking, Searching and Querying in Digital Libraries.* (2000).
11. Barbar'a, D. et al.: The New Jersey data reduction report. *IEEE Data Engineering Bulletin.* 20, 3–45 (1997).
12. Barcelos, R.F., Travassos, G.H.: Evaluation approaches for software architectural documents: A systematic review. *Ibero-American Workshop on Requirements Engineering and Software Environments (IDEAS), La Plata, Argentina.* (2006).
13. Barros, E.G. et al.: A digital library environment for integrating, disseminating and exploring ecological data. *Ecological Informatics.* 3, 4, 295–308 (2008).
14. Batcheller, J.K.: Automating geospatial metadata generation – An integrated data management and documentation approach. *Computers & Geosciences.* 34, 4, 387–398 (2008).
15. Billerbeck, B., Zobel, J.: When query expansion fails. *Proceedings of SIGIR.* pp. 387–388 (2003).
16. Blackburn, K. et al.: XSIL: Extensible Scientific Interchange Language. *Center for Advanced Computing Research, Space Radiation Laboratory, CalTech* (1999).
17. Brin, S., Page, L.: The anatomy of a large-scale hypertextual web search engine. *Computer Networks and ISDN Systems.* 30, 1-7, 107–117 (1998).

18. Bruno, N. et al.: Top-k selection queries over relational databases: Mapping strategies and performance evaluation. *ACM Transactions on Database Systems (TODS)*. 27, 2, 153–187 (2002).
19. Cacheda, F. et al.: A case study of distributed information retrieval architectures to index one terabyte of text. *Information Processing & Management*. 41, 5, 1141–1161 (2005).
20. Cafarella, M.J. et al.: Webtables: exploring the power of tables on the web. *Proceedings of the VLDB Endowment*. 1, 1, 538–549 (2008).
21. Carey, M.J., Kossmann, D.: On saying “enough already!” in SQL. *ACM SIGMOD Record*. 26, 2, 219–230 (1997).
22. Carlson, S.: Lost in a sea of science data. *The Chronicle of Higher Education*. 52, 42, A35 (2006).
23. Chang, Y.C. et al.: The onion technique: Indexing for linear optimization queries. *ACM SIGMOD Record*. pp. 391–402 (2000).
24. Chapelle, O. et al.: Expected reciprocal rank for graded relevance. *Proceedings of CIKM*. pp. 621–630 (2009).
25. Charles, D., Chellapilla, K.: Bloomier filters: A second look. *Algorithms-ESA 2008*. pp. 259–270 Springer (2008).
26. Chaudhuri, S. et al.: Integrating DB and IR technologies: What is the sound of one hand clapping. *CIDR’05*. 1–12 (2005).
27. Cornillon, P. et al.: OPeNDAP: Accessing data in a distributed, heterogeneous environment. *Data Science Journal*. 2, 0, 164–174 (2003).
28. D’Ulizia, A. et al.: Approximating geographical queries. *Journal of Computer Science and Technology*. 24, 6, 1109–1124 (2009).
29. Das, G. et al.: Answering top-k queries using views. *Proceedings of VLDB*. pp. 451–462 (2006).
30. Datta, R. et al.: Content-based image retrieval: Approaches and trends of the new age. *Proceedings of ACM SIGMM International Workshop on Multimedia Information Retrieval*. pp. 253–262 (2005).
31. Deerwester, S.C. et al.: Indexing by latent semantic analysis. *Journal of the American Society for Information Science*. 41, 6, 391–407 (1990).
32. Demartini, G. et al.: Overview of the INEX 2009 entity ranking track. *Focused Retrieval and Evaluation*. 254–264 (2010).
33. Domenico, B. et al.: Thematic Real-time Environmental Distributed Data Services (THREDDS): Incorporating interactive analysis tools into NSDL. *Journal of Digital Information*. 2, 4, (2006).
34. Drosou, M., Pitoura, E.: Search result diversification. *ACM SIGMOD Record*. 39, 1, 41–47 (2010).
35. Dubin, D.: The most influential paper Gerard Salton never wrote. *Graduate School of Library and Information Science, University of Illinois at Urbana-Champaign* (2004).

36. Egenhofer, M.J.: Toward the semantic geospatial web. Proc. of the 10th ACM International Symposium on Advances in Geographic Information Systems. pp. 1–4 (2002).
37. Elbassuoni, S. et al.: Query relaxation for entity-relationship search. The Semantic Web: Research and Applications. pp. 62–76 Springer (2011).
38. Evans, M.P.: Analysing Google rankings through search engine optimization data. Internet Research. 17, 1, 21–37 (2007).
39. Fabrikant, S.I. et al.: The distance-similarity metaphor in network-display spatializations. Cartography and Geographic Information Science. 31, 4, 237–252 (2004).
40. Fagin, R. et al.: Optimal aggregation algorithms for middleware. Proceedings of ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems. pp. 102–113 (2001).
41. Gaasterland, T.: Cooperative answering through controlled query relaxation. IEEE Expert. 12, 5, 48–59 (1997).
42. Gartner, Inc.: Gartner says solving “Big Data” challenge involves more than just managing volumes of data, <http://www.gartner.com/it/page.jsp?id=1731916>, Accessed June 28 2012.
43. Gey, F.C. et al.: Advanced search technologies for unfamiliar metadata. California Univ. Berkeley School of Information Management and Systems (2001).
44. Gonzalez, H. et al.: Google Fusion Tables: Data management, integration and collaboration in the cloud. Proceedings of ACM Symposium on Cloud Computing. pp. 175–180 ACM, New York, NY, USA (2010).
45. Gonzalez, H. et al.: Google fusion tables: web-centered data management and collaboration. Proceedings of the 2010 International Conference on Management of Data. pp. 1061–1066 (2010).
46. Goodchild, M.F. et al.: Sharing geographic information: An assessment of the Geospatial One-Stop. Annals of the AAG. 97, 2, 250–266 (2007).
47. Goodchild, M.F., Zhou, J.: Finding geographic information: Collection-level metadata. GeoInformatica. 7, 2, 95–112 (2003).
48. Gray, J. et al.: Scientific data management in the coming decade. ACM SIGMOD Record. 34, 4, 34–41 (2005).
49. Gray, J., Szalay, A.: The world-wide telescope. Communications of the ACM. 45, 11, 50–55 (2002).
50. Green, J.L. et al.: Complexity in ecology and conservation: Mathematical, statistical, and computational challenges. BioScience. 55, 6, 501–510 (2005).
51. Grossner, K.E. et al.: Defining a digital earth system. Transactions in GIS. 12, 1, 145–160 (2008).
52. Guntzer, J. et al.: Optimizing multi-feature queries for image databases. Proceedings of VLDB. 10–14 (2000).

53. Guntzer, J. et al.: Towards efficient multi-feature queries in heterogeneous environments. Proceedings of the International Conference on Information Technology: Coding and Computing. pp. 622–628 (2001).
54. Harter, S.P.: Variations in relevance assessments and the measurement of retrieval effectiveness. *Journal of the American Society for Information Science*. 47, 1, 37–49 (1996).
55. Heisterkamp, D.R.: Building a latent semantic index of an image database from patterns of relevance feedback. Proceedings of 16th International Conference on Pattern Recognition. pp. 134–137 (2002).
56. Hellerstein, J.M. et al.: Generalized search trees for database systems. *Readings in Database Systems*. 101 (1998).
57. Hellerstein, J.M., Pfeffer, A.: The RD-tree: An index structure for sets. University of Wisconsin-Madison, TR-1252 (1994).
58. Herring, J.R. ed: OpenGIS® Implementation Standard for Geographic Information - Simple Feature Access - Part 1: Common Architecture. Open Geospatial Consortium (2010).
59. Hey, T., Trefethen, A.: E-Science and its implications. *Philosophical Transactions of the Royal Society of London. Series A: Mathematical, Physical and Engineering Sciences*. 361, 1809, 1809 (2003).
60. Hey, T., Trefethen, A.E.: The data deluge: An e-science perspective. *Grid Computing: Making the Global Infrastructure a Reality* (eds F. Berman, G. Fox and T. Hey). pp. 809–824 John Wiley & Sons, Ltd, Chichester, UK (2003).
61. Hill, L.L. et al.: Collection metadata solutions for digital library applications. *Journal of the American Society for Information Science*. 50, 13, 1169–1181 (1999).
62. Howe, B. et al.: Emergent semantics: Towards self-organizing scientific metadata. *Semantics of a Networked World*. pp. 177–198 Springer Berlin / Heidelberg (2004).
63. Howe, B. et al.: Scientific mashups: Runtime-configurable data product ensembles. In: Winslett, M. (ed.) *Scientific and Statistical Database Management*. pp. 19–36 Springer Berlin / Heidelberg (2009).
64. Hristidis, V. et al.: PREFER: A system for the efficient execution of multi-parametric ranked queries. *ACM SIGMOD Record*. 30, 2, 259–270 (2001).
65. Ilyas, I.F. et al.: A survey of top-k query processing techniques in relational database systems. *ACM Computing Surveys (CSUR)*. 40, 4, 11 (2008).
66. Jansen, B.J. et al.: Real life, real users, and real needs: A study and analysis of user queries on the web. *Information Processing & Management*. 36, 2, 207–227 (2000).
67. Järvelin, K., Kekäläinen, J.: Cumulated gain-based evaluation of IR techniques. *ACM Transactions on Information Systems (TOIS)*. 20, 4, 422–446 (2002).
68. Kazman, R. et al.: ATAM: Method for architecture evaluation. Defense Technical Information Center (2000).
69. Kazman, R. et al.: The essential components of software architecture design and analysis. *Journal of Systems and Software*. 79, 8, 1207–1216 (2006).

70. Klein, M.H. et al.: Attribute-based architecture styles. Proceedings of the TC2 First Working IFIP Conference on Software Architecture (WICSA1). pp. 225–244 (1999).
71. Kobayashi, M., Takeda, K.: Information retrieval on the web. ACM Comput. Surv. 32, 144–173 (2000).
72. Koposov, S., Bartunov, O.: Q3C, Quad Tree Cube: The new sky-indexing concept for huge astronomical catalogues and its realization for main astronomical queries (cone search and Xmatch) in open source database PostgreSQL. Astronomical Data Analysis Software and Systems XV. pp. 735–738 (2006).
73. Koudas, N. et al.: Relaxing join and selection queries. Proceedings of VLDB. pp. 199–210 (2006).
74. Kunszt, P. et al.: The indexing of the SDSS science archive. Astronomical Data Analysis Software and Systems. Vol. 216, (2000).
75. Lakoff, G.: Where Mathematics Comes From: How the Embodied Mind Brings Mathematics Into Being. Basic Books, New York NY (2000).
76. Lemson, G. et al.: Implementing a general spatial indexing library for relational databases of large numerical simulations. Scientific and Statistical Database Management. pp. 509–526 (2011).
77. Leser, U.: A query language for biological networks. Bioinformatics. 21, Suppl. 2, ii33–ii39 (2005).
78. Levy, S.: How Google's algorithm rules the web, [http://www.wired.com/magazine/2010/02/ff\\_google\\_algorithm/](http://www.wired.com/magazine/2010/02/ff_google_algorithm/), Accessed June 8 2012.
79. Lewandowski, D.: Web searching, search engines and information retrieval. Information Services and Use. 25, 3, 137–147 (2005).
80. Long, X., Suel, T.: Optimized query execution in large search engines with global page ordering. Proceedings of VLDB. pp. 129–140 (2003).
81. Lord, P., Macdonald, A.: E-Science curation report, [http://www.jisc.ac.uk/uploaded\\_documents/e-ScienceReportFinal.pdf](http://www.jisc.ac.uk/uploaded_documents/e-ScienceReportFinal.pdf), (2003).
82. Maier, D. et al.: Navigating oceans of data. In: Ailamaki, A. and Bowers, S. (eds.) Scientific and Statistical Database Management. pp. 1–19 Springer, Heidelberg (2012).
83. Manning, C.D. et al.: Introduction to Information Retrieval. Cambridge University Press (2008).
84. Markl, V. et al.: MAXENT: Consistent cardinality estimation in action. Proceedings of ACM SIGMOD International Conference on Management of Data. pp. 775–777 (2006).
85. Markowetz, A. et al.: Design and implementation of a geographic search engine. 8th Int. Workshop on the Web and Databases (WebDB). (2005).
86. Markowetz, A. et al.: Design and implementation of a geographic search engine. Technical Report TR-CIS-2005-03. Polytechnic University, Brooklyn, NY, CIS Department (2005).

87. Maron, M.E., Kuhns, J.L.: On relevance, probabilistic indexing and information retrieval. *Journal of the ACM*. 7, 3, 216–244 (1960).
88. Al-Maskari, A. et al.: The relationship between IR effectiveness measures and user satisfaction. *Proceedings of SIGIR*. pp. 773–774 (2007).
89. Mattsson, M. et al.: Software architecture evaluation methods for performance, maintainability, testability, and portability. *Second International Conference on the Quality of Software Architectures*. (2006).
90. Megler, V.M.: Taming the metadata mess. *IEEE 29th International Conference on Data Engineering Workshops (ICDEW)*. pp. 286–289, IEEE Computer Society, Brisbane, Australia (2013).
91. Megler, V.M., Maier, D.: Finding haystacks with needles: Ranked search for data using geospatial and temporal characteristics. In: Bayard Cushing, J. et al. (eds.) *Scientific and Statistical Database Management*. pp. 55–72 Springer, Heidelberg (2011).
92. Megler, V.M., Maier, D.: When big data leads to lost data. *Proceedings of the 5th Ph.D. workshop on Information and knowledge (PIKM)*. pp. 1–8 (2012).
93. Michener, W.K.: Meta-information concepts for ecological data management. *Ecological Informatics*. 1, 1, 3–7 (2006).
94. Miller, C.C.: A Beast in the field: The Google Maps mashup as GIS/2. *Cartographica*. 41, 3, 187–199 (2006).
95. Miller, H.J., Wentz, E.A.: Representation and spatial analysis in geographic information systems. *Annals of the AAG*. 93, 3, 574–594 (2003).
96. Moellering, H.: World Spatial Metadata Standards: Scientific and Technical Descriptions, and Full Descriptions with Crosstable. Elsevier, Amsterdam (2005).
97. Moffat, A., Zobel, J.: Rank-biased precision for measurement of retrieval effectiveness. *ACM Transactions on Information Systems (TOIS)*. 27, 1, 2 (2008).
98. Montello, D. et al.: Testing the first law of cognitive geography on point-display spatializations. *Spatial Information Theory*. 316–331 (2003).
99. Montello, D.: The geometry of environmental knowledge. *Theories and Methods of Spatio-Temporal Reasoning in Geographic Space*. 136–152 (1992).
100. Montello, D.R.: The measurement of cognitive distance: Methods and construct validity. *Journal of Environmental Psychology*. 11, 2, 101–122 (1991).
101. Mork, P. et al.: PQL: A declarative query language over dynamic biological schemata. *Proceedings of the AMIA Symposium*. pp. 533–537 (2002).
102. Murray-Rust, P., Rzepa, H.S.: Chemical Markup, XML, and the Worldwide Web. 1. Basic Principles. *J. Chem. Inf. Comput. Sci.* 39, 6, 928–942 (1999).
103. Pallickara, S.L. et al.: Efficient metadata generation to enable interactive data discovery over large-scale scientific data collections. *2nd IEEE Intnl. Conf. on Cloud Computing, Technology and Science*. pp. 573–580 (2010).
104. Papadias, D. et al.: Progressive skyline computation in database systems. *ACM Transactions on Database Systems (TODS)*. 30, 1, 41–82 (2005).

105. Parent, C. et al.: Conceptual modeling for traditional and spatio-temporal applications: The MADS approach. Springer Heidelberg (2006).
106. Parsons, J., Wand, Y.: Attribute-based semantic reconciliation of multiple data sources. Lecture Notes in Computer Science. 2800, 21–47 (2003).
107. Perlman, E. et al.: Data exploration of turbulence simulations using a database cluster. Proceedings of the ACM/IEEE Conference on Supercomputing. pp. 1–11 (2007).
108. Poulovassilis, A., Wood, P.T.: Combining approximation and relaxation in semantic web path queries. The Semantic Web—ISWC 2010. pp. 631–646 Springer (2010).
109. Rajasekar, A., Moore, R.: Data and metadata collections for scientific applications. High-Performance Computing and Networking. pp. 72–80 (2010).
110. Reichman, O.J. et al.: Challenges and opportunities of open data in ecology. Science (Washington). 331, 6018, 703–705 (2011).
111. Rew, R., Davis, G.: NetCDF: An interface for scientific data access. Computer Graphics and Applications, IEEE. 10, 4, 76–82 (1990).
112. Roddick, J.F. et al.: A unifying semantic distance model for determining the similarity of attribute values. Proceedings of the 26th Australasian Computer Science Conference. pp. 111–118 (2003).
113. Salton, G.: Automatic Information Organization and Retrieval. McGraw-Hill New York (1968).
114. Sanderson, M. et al.: Do user preferences and evaluation measures line up? Proceedings of SIGIR. pp. 555–562 (2010).
115. Saracevic, T.: Relevance: A review of the literature and a framework for thinking on the notion in information science. Part II: nature and manifestations of relevance. Journal of the American Society for Information Science and Technology. 58, 13, 1915–1933 (2007).
116. Saracevic, T.: Relevance: A review of the literature and a framework for thinking on the notion in information science. Part III: Behavior and effects of relevance. Journal of the American Society for Information Science and Technology. 58, 13, 2126–2144 (2007).
117. Schurman, E., Brutlag, J.: The user and business impact of server delays, additional bytes, and HTTP chunking in web search. Proceedings of Velocity: Web Performance and Operations Conference. , San Jose, Calif. (2009).
118. Schwering, A.: Approaches to semantic similarity measurement for geo-spatial data: A survey. Transactions in GIS. 12, 1, 5–29 (2008).
119. Sharifzadeh, M., Shahabi, C.: The spatial skyline queries. Proceedings of VLDB. pp. 751–762 (2006).
120. Shvachko, K. et al.: The Hadoop distributed file system. 2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST). pp. 1–10 (2010).
121. Singh, G. et al.: A metadata catalog service for data intensive applications. Proceedings of the 2003 ACM/IEEE Conference on Supercomputing. p. 33 (2003).

122. Skupin, A., Buttenfield, B.P.: Spatial metaphors for visualizing very large data archives. *Proceedings of GIS/LIS '96*. pp. 607–617 (1996).
123. Sormunen, E.: Liberal relevance criteria of TREC: Counting on negligible documents? *Proceedings of SIGIR*. pp. 324–330 (2002).
124. Stolte, E. et al.: Scientific data repositories: designing for a moving target. *Proceedings of ACM SIGMOD International Conference on Management of Data*. pp. 349–360 (2003).
125. Stolte, E., Alonso, G.: Efficient exploration of large scientific databases. *Proceedings of VLDB*. p. 633 (2002).
126. Su, L.T.: The relevance of recall and precision in user evaluation. *Journal of the American Society for Information Science*. 45, 3, 207–217 (1994).
127. Tata, S. et al.: Declarative querying for biological sequences. *Proceedings of the International Conference on Data Engineering*. pp. 87–87 (2006).
128. Teevan, J. et al.: Beyond the commons: Investigating the value of personalizing web search. *Proceedings of the Workshop on New Technologies for Personalized Information Access (PIA)*. pp. 84–92 (2005).
129. Thomson, J. et al.: Metadata's role in a scientific archive. *Computer*. 36, 12, 27–34 (2003).
130. Tomasic, A., Garcia-Molina, H.: Performance of inverted indices in shared-nothing distributed text document information retrieval systems. *Proceedings of the Conference on Parallel and Distributed Information Systems*. pp. 8–17 (1993).
131. Tversky, A.: Features of similarity. *Psychological Review*. 84, 4, 327 (1977).
132. Tversky, A., Gati, I.: Studies of similarity. *Cognition and Categorization*. 1, 79–98 (1978).
133. U.S. Federal Geographic Data Committee: Content Standard for Digital Geospatial Metadata. (1998).
134. Urbano, J.: Information retrieval meta-evaluation: Challenges and opportunities in the music domain. *International Society for Music Information Retrieval Conference*. pp. 609–614 (2011).
135. Vanea, A., Potolea, R.: A hierarchical semantically enhanced multimedia data warehouse. *Proceedings of the IEEE International Conference on Intelligent Computer Communication and Processing*. pp. 3–9 (2010).
136. Vartak, M. et al.: QRelX: Generating meaningful queries that provide cardinality assurance. *Proceedings of the ACM SIGMOD International Conference on Management of Data*. pp. 1215–1218 (2010).
137. Vartak, M.: Query-oriented relaxation for cardinality assurance, <http://src.acm.org/2010/ManasiVartak/ACM-SRC-Manasi-Vartak/ACM-SRC.html>.
138. Venetis, P. et al.: Recovering semantics of tables on the web. *Proceedings of VLDB*. 4, 9, 528–538 (2011).
139. Voorhees, E., Tice, D.M.: The TREC-8 question answering track evaluation. *Text Retrieval Conference TREC*. pp. 83–105 (1999).

140. Wang, J. et al.: Indexing multi-dimensional data in a cloud system. Proceedings of the ACM SIGMOD International Conference on Management of Data. pp. 591–602 (2010).
141. Wang, X. et al.: Liferaft: Data-driven, batch processing for the exploration of scientific databases. CIDR. (2009).
142. Weidman, S., Arrison, T.: Steps toward large-scale data integration in the sciences: Summary of a workshop. National Research Council of the National Academies (2009).
143. White, R.W., Morris, D.: Investigating the querying and browsing behavior of advanced search engine users. Proceedings of SIGIR. pp. 255–262 (2007).
144. Yakout, M. et al.: Infogather: entity augmentation and attribute discovery by holistic matching with web tables. Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data. pp. 97–108 ACM (2012).
145. Zhang, L. et al.: A fast indexing approach for protein structure comparison. BMC Bioinformatics. 11, Suppl 1, S46 (2010).
146. Zhu, J. et al.: DYNIQX: A novel meta-search engine for metadata based cross search. International Conference on Applications of Digital Information and Web Technologies (ICADIWT). pp. 204–209 (2008).
147. Center for Coastal Margin Observation & Prediction (CMOP), <http://www.stccmop.org/>, Accessed April 17 2011.
148. Center for Coastal Margin Observation and Prediction (CMOP) RIG Meeting, July 15 2010.
149. Chapter 4. Using PostGIS: Data Management and Queries, [http://postgis.net/docs/manual-2.0/using\\_postgis\\_dbmanagement.html#id437760](http://postgis.net/docs/manual-2.0/using_postgis_dbmanagement.html#id437760), Accessed September 4 2013.
150. Content Standard References: Marine Metadata Interoperability, <https://marinemetadata.org/conventions/content-standards>, Accessed July 26 2013.
151. Geospatial One Stop (GOS), <http://gos2.geodata.gov/wps/portal/gos>, Accessed January 19 2011.
152. Global Change Master Directory Web Site, <http://gcmd.nasa.gov/>, Accessed January 19 2011.
153. Initiative for the Evaluation of XML Retrieval (INEX), <https://inex.mmc.uni-saarland.de/>, Accessed June 4 2012.
154. PostGIS — Spatial and Geographic Objects for PostgreSQL, <http://postgis.net/>, Accessed November 13 2013.
155. The Gist of the GiST, <http://gist.cs.berkeley.edu/gist1.html>, Accessed June 20 2011.
154. The Google Maps Javascript API V3, <http://code.google.com/apis/maps/documentation/javascript/>, Accessed January 20 2011.

## **Appendix A: DNH PostgreSQL Database Indexes**

```
CREATE INDEX metadata_files_parent ON dnh.metadata_files
    USING btree (parent, kids , mintime , maxtime , id, count );
CREATE INDEX metadata_files_parentnull ON dnh.metadata_files
    USING btree WHERE parent IS NULL;
CREATE INDEX metadata_vars_idvar ON dnh.metadata_vars
    USING btree (variable, varmin , varmax , varunits, id);
CREATE INDEX metadata_vars_varid ON dnh.metadata_vars
    USING btree (variable, id, varunits);
```

## **Appendix B: Variable List**

Table B.1 lists the variables available from the user interface during the time of the user study. Research not described in this dissertation (but described in Megler[90]) added an additional layer of organization (“concept”, denoting the higher-level concept that many of these variables are measures of) and a translation of the column name within the dataset to the actual variable represented. The list of concepts and variables chosen by CMOP, and the column names mapped to them, is shown in Table B.2. An additional set of 222 column names were designated as column names that should be “not visible” to the user until the actual dataset was downloaded; these columns often represent calculated, statistical or quality assurance measures.

**Table B.1. Variable Names at the Time of User Study**

airtemp	avg_nh4
alongvel	avg_nn
altimeter	avg_no2
altimeter_std	avg_oxygen
apna_mode	avg_par
A_Tideheight	avg_ph
atmpres	avg_phaeophytin
avg_atmosphericpressure	avg_po4
avg_atmospherictemp	avg_pressure
avg_bottle_depth	avg_quality
avg_cdом_voltage	avg_salinity
avg_chl	avg_shortwave_radiation
avg_chla_fluorescence	avg_silicate
avg_conductivity	avg_temperature
avg_ct_water_temperature	avg_transmiss
avg_fluorescein	avg_turbidity
avg_fluorescence	avg_winddirection
avg_ft_conductivity	avg_windspeed
avg_ft_par	bindepth
avg_ft_salinity	bindist
avg_ft_temp	bottom
avg_longwave_radiation	bp

bp_fl	fluorescence
bp_leu	fluorescence_std
bp_leu_fl	flux
bs_avg	fm
capo4	fo
cast_dataset	height
cdom	humidity
clay	isus_inter
co2	isus_slope
coastal_upwelling_index_45N	leak
coastal_upwelling_index_48N	nh3
cond	nitrate
conductivity	no2
conductivity_std	northwind
count	nox
cr_flow_at_bonneville	numbottles
crossvel	num_bottles
cr_residence_time	oxygen
cumulative_flow	oxygensat
cv_bottle_depth	oxygen_saturation
cv_chl	oxygen_saturation_std
cv_ft_salinity	oxygen_std
cv_ft_temp	pacific_decadal_oscillation_in_month
cv_nh4	par
cv_nn	par_std
cv_no2	ph
cv_phaeophytin	pH
cv_po4	ph_std
cv_salinity	phycoeryth
cv_silicate	po13c
cv_temperature	po15n
dataset	po4
deploymentid	poc
depth	pon
Dist_from_Astoria	pres
doc	pressure
elevation	pressure_std
estocant	pump
fl_fluores	quality
fluores	q_yield
fluores375	salinity
fluores400	salinity_std
fluores420	salt
fluores435	sand
fluores470	scan
fluores505	scattcoef
fluores525	sdocnoxy
fluores570	sdocntemp
fluores590	secondary_conductivity
fluorescein	secondary_conductivity_std
fluorescein_std	secondary_fluorescein

secondary_fluorescein_std	vel
secondary_salinity	vel_e
secondary_salinity_std	vel_mag
secondary_temperature	vel_n
secondary_temperature_std	vel_vert
si	voltage_channel_0
silt	voltage_channel_0_std
so_slope	voltage_channel_1_std
status	voltage_channel_2_std
st_slope	voltage_channel_3_std
sum	voltage_channel_4_std
sumscat	voltage_channel_5_std
tau	voltage_channel_6_std
tdn	voltage_channel_7_std
tdp	water_electrical_conductivity
temp	water_pressure
temperature	water_salinity
temperature_std	water_temperature
time	winddir
transcount	windgust
transmiss	windspeed
transmiss_std	xwind
turbidity	ywind
turbidity_std	
vapo4	

**Table B.2. Concepts, Variable and Column Names (as of March, 2014)**

Concept	Variable	Column
calculated_data	estoceant	estoceant
calculated_data	estoxxygen	estoxxygen
calculated_data	sdocnoxy	sdocnoxy
calculated_data	sdocntemp	sdocntemp
calculated_data	so_slope	so_slope
calculated_data	st_slope	st_slope
cDOM	cdom	cdom
cDOM	cdom	cdom_voltage
cDOM	cdom_voltage	avg_cdom_voltage
conductivity	cond	avg_conductivity
conductivity	cond	avg_ft_conductivity
conductivity	cond	conductivity
conductivity	cond	secondary_conductivity
conductivity	water_electrical_conductivity	water_electrical_conductivity
metadata	apna_mode	apna_mode
metadata	cumulative_flow	cumulative_flow
metadata	doc	doc

<b>Concept</b>	<b>Variable</b>	<b>Column</b>
metadata	filename	bpsourcefilename
metadata	isus_inter	isus_inter
metadata	isus_slope	isus_slope
metadata	leak	leak
metadata	notes	bp_notes
metadata	notes	chl_a_notes
metadata	notes	dna_test_notes
metadata	notes	nutrients_notes
metadata	notes	ws_notes
metadata	pump	pump
metadata	sampleid	sampleid
metadata	sum	sum
metadata	sumscat	sumscat
metadata	tau	tau
metadata	transcount	transcount
metadata	vessel	vessel
microbiology	bp	bp
microbiology	bp_fl	bp_fl
microbiology	bp_leu	bp_leu
microbiology	bp_leu_fl	bp_leu_fl
microbiology	dna_tests	dna_tests
Murrays_dye_experiment	fl_fluores	fl_fluores
Murrays_dye_experiment	fl_fluores	fluorescein
Murrays_dye_experiment	fluorescein	avg_fluorescein
nutrient	capo4	capo4
nutrient	nh3	nh3
nutrient	nh4	avg_nh4
nutrient	nitrate	nitrate
nutrient	nitrate	voltage_channel_6
nutrient	nitrate	voltage_channel_7
nutrient	nn	avg_nn
nutrient	no2	avg_no2
nutrient	no2	no2
nutrient	nox	nox
nutrient	po13c	po13c
nutrient	po15n	po15n
nutrient	po4	avg_po4
nutrient	po4	po4
nutrient	POC	poc
nutrient	PON	pon
nutrient	si	si
nutrient	silicate	avg_silicate
nutrient	tdn	tdn

Concept	Variable	Column
nutrient	tdp	tdp
oxygen	oxygen	avg_oxygen
oxygen	oxygen	oxygen
oxygen	oxygen	voltage_channel_2
oxygen	oxygensat	oxygensat
oxygen	oxygensat	oxygen_saturation
pH-CO2	co2	co2
pH-CO2	ph	ph
pH-CO2	pH	avg_ph
pH-CO2	pH	ph
pigment	chloro	avg_chl
pigment	chloro_fl	avg_chla_fluorescence
pigment	fluores	avg_fluorescence
pigment	fluores	chla_fluorescence
pigment	fluores	fluorescence
pigment	fluores	voltage_channel_0
pigment	fluores	voltage_channel_1
pigment	fluorescence	fluorescence
pigment	fm	fm
pigment	fo	fo
pigment	phaeophytin	avg_phaeophytin
pigment	phycoeryth	phycoeryth
pigment	phycoeryth	phycoerythrin
pigment	q_yield	q_yield
river discharge	flux	flux
salinity	salt	avg_ft_salinity
salinity	salt	avg_salinity
salinity	salt	salinity
salinity	salt	secondary_salinity
salinity	water_salinity	water_salinity
temperature	temp	atmospheric_temperature
temperature	temp	avg_ct_water_temperature
temperature	temp	avg_ft_temp
temperature	temp	avg_temperature
temperature	temp	ct_water_temperature
temperature	temp	secondary_temperature
temperature	temp	temperature
temperature	water_temperature	water_temperature
tides	elevation	elevation
tides	water_pressure	water_pressure
time-space	altitude	altitude
time-space	bindepth	bindepth
time-space	bottom	avg_beddepth

<b>Concept</b>	<b>Variable</b>	<b>Column</b>
time-space	bottom	voltage_channel_4
time-space	depth	avg_bottle_depth
time-space	depth	depth
time-space	depth	samplingdepth
time-space	height	height
time-space	latitude	latitude
time-space	longitude	longitude
time-space	pres	atmosphericpressure
time-space	pres	avg_pressure
time-space	pres	pressure
time-space	site	site
time-space	time	time
turbidity	backscat	backscatter
turbidity	bs_avg	bs_avg
turbidity	clay	clay
turbidity	sand	sand
turbidity	scattcoef	scattcoef
turbidity	silt	silt
turbidity	transmiss	avg_transmiss
turbidity	transmiss	transmiss
turbidity	transmiss	voltage_channel_3
turbidity	turbidity	avg_turbidity
turbidity	turbidity	turbidity
turbidity	turbidity	voltage_channel_0
water_current	alongvel	alongvel
water_current	crossvel	crossvel
water_current	vel	vel
water_current	vel_e	vel_e
water_current	vel_mag	vel_mag
water_current	vel_n	vel_n
water_current	vel_vert	vel_vert
weather	airtemp	airtemp
weather	airtemp	avg_atmospherictemp
weather	atmpres	atmpres
weather	atmpres	avg_atmosphericpressure
weather	humidity	humidity
weather	longwaver	avg_longwave_radiation
weather	longwaver	longwave_radiation
weather	northwind	northwind
weather	par	avg_ft_par
weather	par	avg_par
weather	par	par
weather	par	voltage_channel_5

<b>Concept</b>	<b>Variable</b>	<b>Column</b>
weather	shortwaver	avg_shortwave_radiation
weather	shortwaver	shortwave_radiation
weather	winddir	avg_winddirection
weather	winddir	winddir
weather	winddir	winddirection
weather	windgust	windgust
weather	windspeed	avg_windspeed
weather	windspeed	windspeed