Techniques for Structured Data Discovery

by

Lordique Fok

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Engineering in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

February 2019

Author ..... Signature redacted .....
Department of Electrical Engineering and Computer Science
January 20, 2019

Certified by..... Signature redacted .....
Eran Egozy
Professor of the Practice in Music Technology
Thesis Supervisor

Certified by..... Signature redacted .....
David Andrzejewski
Senior Manager of Engineering at Sumo Logic
Thesis Supervisor

Accepted by ..... Signature redacted .....
Katrina LaCurts
Chair, Master of Engineering Thesis Committee

Techniques for Structured Data Discovery

by

Lordique Fok

Submitted to the Department of Electrical Engineering and Computer Science
on January 20, 2019, in partial fulfillment of the
requirements for the degree of
Master of Engineering in Electrical Engineering and Computer Science

Abstract

The discovery of structured data, or data that is tagged by key-value pairs, is a problem that can be subdivided into two issues: how best to structure information architecture and user interaction for discovery; and how to intelligently display data in a way that that optimizes the discovery of "useful" (i.e. relevant and helpful for a user's current use case) data. In this thesis, I investigate multiple methods of addressing both issues, and the results of evaluating these methods qualitatively and quantitatively.
Specifically, I implement and evaluate: a novel interface design which combines different aspects of existing interfaces, two methods of diversifying data subsets given a search query, three methods of incorporating relevance in data subsets given a search query and information about the user's historic queries, a novel method of visualizing structured data, and two methods of inducing hierarchy on structured data in the presence of an partial data schema.
These implementations and evaluations are shown to be effective in structuring information architecture and user interaction for structured data discovery, but are only partially effective in intelligently displaying data to optimize discovery of useful structured data.

Thesis Supervisor: Eran Egozy
Title: Professor of the Practice in Music Technology

Thesis Supervisor: David Andrzejewski
Title: Senior Manager of Engineering at Sumo Logic

Acknowledgments

I would like to thank my advisers and coworkers who helped me through my MIT masters journey. I especially would like to acknowledge the following people who helped me with my thesis work:

David Andrzejewski, who helped expand my perspective when my thoughts couldn't break the circular path;

Eran Egozy, who provided much-needed wisdom at the interface of work and academia;

Martin Castellanos and Miguel Antochiw, who helped me forge my way into UI engineering;

Aona Yang, who proved an invaluable research partner;

Rohan Singh, who counseled my early steps in interaction design;

Wojciech Donderowicz, whose expertise saved me endless frustration;

and Christian Beedgen, whose farsightedness guides many projects at Sumo Logic, including mine.

Additionally, I would like to expressly thank my parents, Henry and Deanna; and a few of my dearest MIT friends, Allison, Katherine, Jennifer, Li, Cynthia, and Dabin.

Contents

List of Figures

Chapter 1

Introduction

Recent developments in the management software for business metrics (i.e. metrics associated with monitoring technology infrastructure for business purposes, such as the health of a computing cluster) have created the ability to monitor and troubleshoot business metrics in real time. One of the greatest challenges while working in this space, from a customer perspective, is being able to locate relevant data among hundreds or millions of such metrics.

This thesis focuses on the implementation of a system within Sumo Logic, the company with which this 6A thesis was completed, to address this problem.

1.1 Motivation

Sumo Logic is a monitoring and troubleshooting business-to-business platform that ingests and analyzes logs and metrics. It is a platform that collects a business's logs and metrics, and provides a comprehensive tool set to analyze that data through machine learning, visualizations, and a host of operations on the data. However, users of this platform and others like it have difficulty understanding and locating their structured data (i.e., data that is tagged with metadata in the form of key-value pairs, usually pertaining to the data's locality, identity, and purpose) within the product unless they have a very precise notion of what they are looking for and are experts in the query language. This usage difficulty is due to a two-pronged problem:

a lack of transparency in the user interface, as well as Sumo Logic's non-aware or "dumb" treatment of the customer's data. For example, when a customer is running a query, returning multiple trivially different autocomplete suggestions (e.g. typing "Au" returns a suggestions list of "Auto-1", "Auto-2", "Auto-3", etc. rather than "Auto-1", "Australia-2", "Augur").

A related but separate issue is the fact that the system's current format contains no representation of a user's mental model, despite knowledge that certain key-value pairs (KVPs) are of high relevance and interest to users in most, if not all of the use cases, while other KVPs are of negligible importance. For example, a common business metric is CPU load on a computing cluster; a highly relevant key-value pair for this metric would be cluster_name: aws_123, while a (contrived) example of a low-value key-value pair would be master_node: node_5, as the master node is transient and the individual nodes are managed by the cluster provider, not the user.

In order to address the problem of data discovery and navigation, other companies impose restrictions on the sources of structured data, only allowing structured data from common sources such as AWS or Graphite. By imposing these restrictions, they can create hand-curated views of this data that incorporate knowledge of the data source from which the data originated. For example, Graphite data is hierarchical, so there are optimized views for this hierarchy; and though AWS Cloudwatch data is not hierarchical, it has a known set of relevant keys that are useful for a majority of users. [18] However, Sumo Logic's view is data agnostic: they allow data imports from many different types of sources, both common and custom (user-designated, as when a user is collecting structured data from a system they personally maintain). While there is added benefit for the Sumo Logic user in being able to monitor their custom data, custom data makes it difficult for Sumo Logic to create curated data views which incorporate expert knowledge of the underlying data. Therefore, the challenge in this case is to create a general framework that will accommodate the quick location of relevant structured data regardless of its source, by incorporating three things:

1. the user's mental model

2. a diverse and accurate representation of their system, and

3. and the ability to navigate data via a clear user interface.

1.2 Example User Scenario

The following user scenario guided development of this thesis. Our intended user, whom we will call Sandra, is a business metrics system administrator, responsible for setting up the collection of real-time, business-related, structured data in the system. After completing setup, or adding any new business metrics to her account, she has two objectives: to validate that the setup of the data collection is correct and to generate graphs of business metrics so that the engineers responsible for monitoring and troubleshooting the system can easily access the information they need. The data she is working with comes from a variety of sources, some customized to her system, and others being automatically generated from third party systems her business relies upon.

1.3 Thesis Contributions

This thesis introduces three main contributions:

1. An new interface designed for rapidly building metrics queries. This interface will allow Sandra to validate her data collection setup quickly and easily, using two methods. The first is through a "deep" look at the data: seeing an exhaustive list of the keys in her system or all the values of a given key in her system. The second is through a "wide" view of her data: seeing a small set of complete metrics, which allows her to see how key-value pairs she saw in the deep view relate to one another and define pieces of structured data.

2. An evaluation of schemes for prioritizing structured data search results. Better prioritization of structured data search results allows Sandra to more efficiently search for what she wants, by incorporating some notion of her user history and

context (personalization), as well as providing her with a more diverse set of results to cover a larger space in the same number of results.

3. A novel structured data visualization supported by a novel hierarchy induction technique to induce an ordering on metadata keys. Visualization of the structured data allows Sandra to draw insights from looking at the bigger picture: the shape and organization of her system, rather than individual metrics or pieces of metadata.

1.4 Outline

The rest of this thesis consists of the following five chapters. Chapter 2 outlines related work in three categories: interface design, search result prioritization, and hierarchy induction. Chapter 3 discusses experiments with structured data discovery user interfaces, as well as their performance under various metrics. Chapter 4 covers experiments with structured data search result prioritization and the experimental results. Chapter 5 covers experiments with hierarchy induction on structured data and structured data visualization enabled by the induced hierarchy, and Chapter 6 concludes the thesis.

Chapter 2

Related Work

This chapter details three lines of related work: a general overview of structured data discovery interfaces, a view of related work in adding diversity and relevance to search results, and an overview of work in hierarchy generation for structured data.

2.1 Structured Data Discovery Interfaces

Most structured data discovery interfaces–interfaces that allow the user, typified by Sandra, to locate structured business metrics pertaining to their particular use case–fall into one of two categories: query building interfaces, whose main objective is to help Sandra build a query to locate the desired data; and curated structured data discovery interfaces, whose purpose is to surface expert-curated data and views in order to help Sandra gain an overview of what is in the system. Some businesses use a combination of these two approaches to solve different use cases. [21, 6, 15, 19]

2.1.1 Query Building Interfaces

Query building interfaces contain two main components. The first is an interface which allows the user to create a valid query string, often using a precise query language, through some combination of clicking and typing (e.g. autocomplete on a free text type bar, or building a query piece by piece by adding components such as

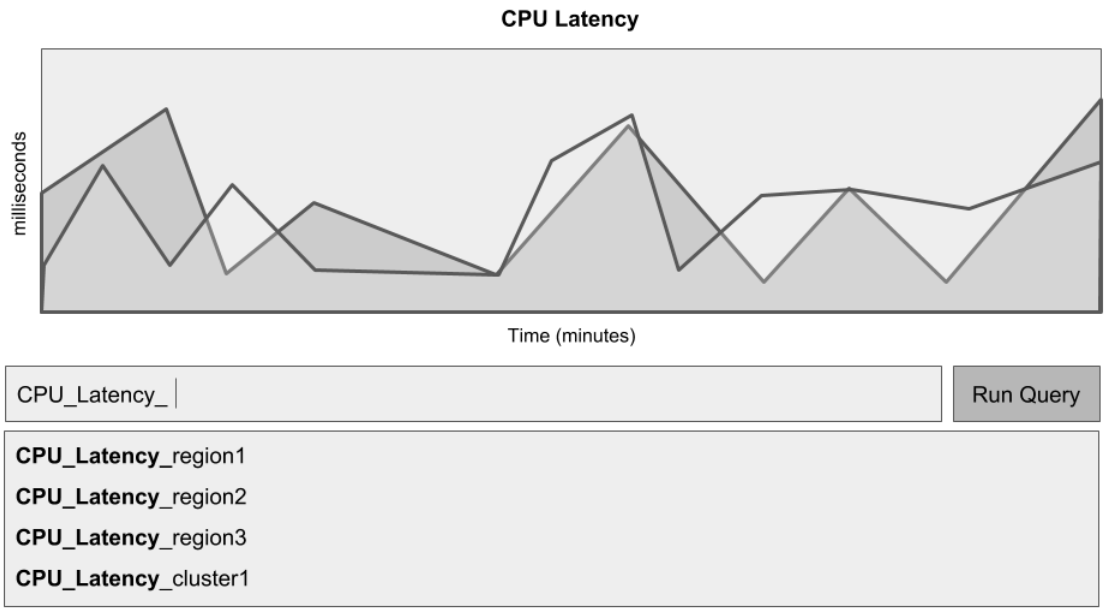


Figure 2-1: Sample query building interface. The chart is a sample data preview, and the text bar with autocomplete is a sample query string creator.

selectors and mathematical operators in a predetermined fashion). [21, 6] The second is a preview of the data which has been selected by the query, which is an essential component for refining the query and validating its correctness. See 2-1 for a skeleton layout of this interface.

2.1.2 Curated Structured Data Discovery Interfaces

Curated structured data discovery interfaces also contain two main components. The first is an actionable side panel, which performs actions such as filtering, searching, and navigating, similar to a side panel in e-commerce. The second is a main panel, which contains interactive, expert-curated data views relevant to data selected via the side panel.[19, 15] See 2-2 for a skeleton layout of this interface.

2.2 Diversity and Relevance in Search Results

Search result diversity and relevance relate to the issue of prioritizing the results of a given query in a way to maximize user utility. For example, a search engine query

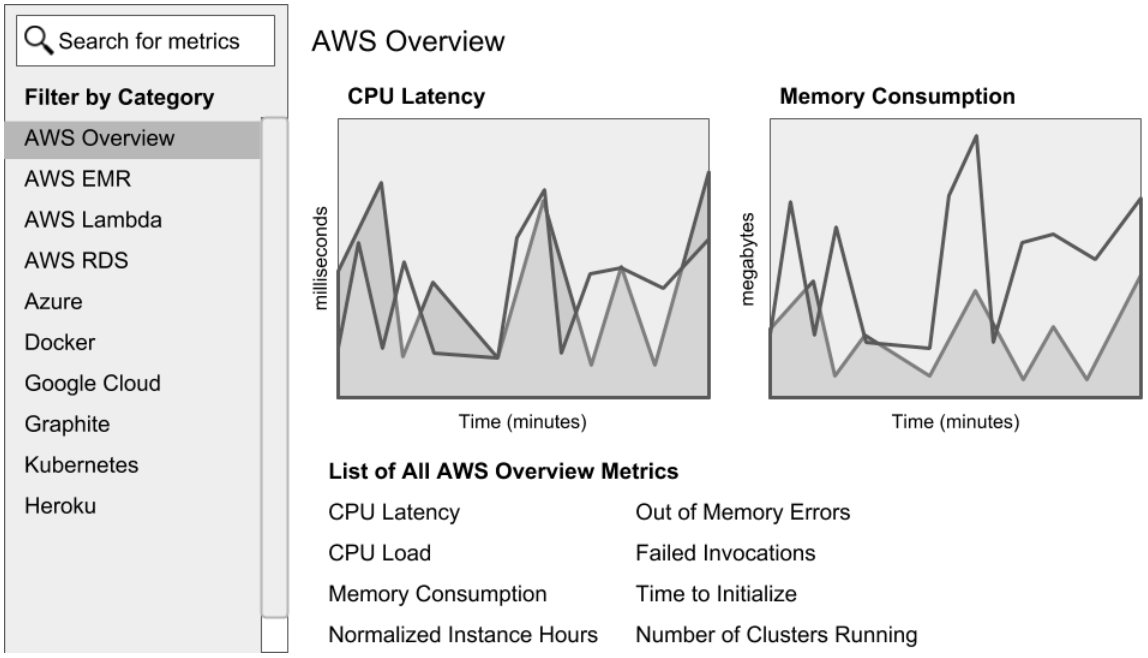


Figure 2-2: Sample curated data view interface. The side panel on the left provides actions, and the view on the right provides relevant information selected by an expert.

for "java" would yield diverse results if the top few included the Java programming language, coffee, and the island Java. It would yield relevant results if it included more prioritized Java programming language results, given that the user had performed many previous searches related to programming. Diverse results cover a wider set of topics, while relevance dives deeper into the user's context.

Diversity maximization is often framed as a submodular problem–that is, a problem with diminishing marginal returns, such that the sum of the values of a diversity function over two disjoint sets is greater than (or equal to) its value over the union of those two sets[11]. This problem has been studied in the context of recommendation systems, such as e-commerce results, movies, or news articles. Because problems in these contexts are often motivated by a query, state-of-the-art solutions usually incorporate an additional treatment of "relevance" or "utility," i.e. they aim to maximize an objective that combines both diversity and relevance of the recommended items. When posed in this dual format, diversity maximization is an NP-hard problem [14].

Maximal Marginal Relevance, or MMR, is a classic algorithm for treating diversity maximization and relevance together[5]. MMR builds a subset of recommended items

by iteratively adding the item with the greatest marginal relevance. The marginal relevance is computed as a function of an item's novelty with respect to the other items in the subset, as well as its relevance score. Indeed, this algorithm is used as the baseline by which to compare new algorithms[1], and also as a foundation for further work.

Another, newer algorithm called DUM, or Diversity-weighted Utility Maximization[3] formulates the problem so that it is not intractable, specifically by maximizing a "modular function subject to a submodular constraint" rather than a submodular function like the marginal relevance function. The result is an algorithm with runtime $O(n\log(n))$, where n is the number of results that match the original query. The requirement for this formulation is that the objective function is submodular and monotone, where f is monotone if $f(A) \leq f(B)$ when $A \subseteq B$.

In contrast to the two approaches above, many algorithms separate the treatment of diversity and relevance. Most important to our case are the approaches to diversity, since we use a hard cutoff for relevance via query matching. We outline some of these approaches to diversity as follows.

The work of Gollapudi and Sharma[8] is an early milestone in this problem. Their paper, "An Axiomatic Approach for Result Diversification," builds the notion of pairwise distance functions being used to holistically evaluate the diversity of an entire set. A more recent extension of this idea is Local Search for Diversity Maximization[2], an algorithm which specifies the exact threshold at which to swap an element into the recommended subset in order to achieve a 2-approximate solution[22] to the optimal diverse subset. A greedy algorithm which achieves the same guarantee is GMM or Global MaxMin[16], which greedily builds a diverse subset by adding elements (to an initial set composed of one random element) whose minimum distance to the current subset is maximized. The most common underlying pairwise distance functions in these algorithms are variations of Jaccard Indices. [1, 2]

Recent work in the area of diversity maximization focuses on ways to make these basic approaches more performant, either via stronger approximation guarantees or via parallelization. For example, the work of Indyk et al.[10] addresses the latter:

they use representative core sets (i.e. smaller sets of data that, when treated by an algorithm, give similar results) to decrease the runtime of the diversity maximization algorithm. Essentially, they run two rounds of mapreduce, first to find the most diverse subset for each core set, then the most diverse subset for the union of those subsets. Because each core set can be operated on independently, this approach may be parallelized. The tradeoff in performance is an increase in the approximation factor from 2 to 3. On the other hand, Abbassi et al. proposes a decrease in the approximation factor to $\frac{1}{2}$ by using matroid constraints and partitioning the data. [2]

2.3 Hierarchy Induction

Hierarchy induction is the process of automatically generating a hierarchy of condensed information from an underlying structured data set. In our application, this means imposing a hierarchy on the set of keys of some structured data points, though generally speaking hierarchy induction is thought of in the relational database context. Imposing a hierarchy on the keys of structured data would allow us to do several things, including:

1. infer the user's mental model of their data
2. generate meaningful data visualizations for a set of structured data
3. optimize data storage for faster access based on hierarchical knowledge

Related work in hierarchy induction is most often applied to databases, where the resulting "concept hierarchies" are used to summarize and analyze the data for the end user. For example, a concept hierarchy may reduce information stored (data roll-ups, binning), provide information about how data attributes are related (partial ordering of nominal keys, e.g. $country \prec state \prec city$), or group data points together by similarity (clustering). For our purposes, the most useful database analog to the work presented here is partial ordering of nominal keys. While there is a large corpus of work on how to use such concept hierarchies, most techniques rely on manual

creation of these hierarchies rather than automatic generation. There are several notable exceptions, several of which are described below. [9]

One approach piloted by Lee et al. uses natural language understanding to generate a concept hierarchy on the nominal values of a single attribute. [12] Their approach incorporates WordNet to automatically discover higher-level connections between nominal values (e.g. Python, Java, and C+ would automatically be grouped into the coding languages concept, which in turn could be grouped into a larger computer science category), thus generating a human-like hierarchy of values.

Another approach by Godin et al.[7] uses incremental clustering and rule generation to create a concept lattice, which is used to group data so that questions about the relationship of various values across data attributes may be answered (e.g. does type=mammal necessitate blood=warm).

Finally, Yichun Lu[13] suggests creating a partial order of keys using the number of unique values associated with each key, noting positive results on the nominal location-based data in the CITYDATA database. This approach is most similar to the ones outlined in this thesis.

Chapter 3

User Interfaces

The interface design described below is optimized for four things in Sandra's use case:

1. Allowing her to provide information about her mental model of the system, and leveraging that data intelligently

2. Showing her an exhaustive (but controlled) view of her data so she can easily validate her data collection

3. Using a data agnostic main view which will help her validate her setup and generate graphs easily

4. Allowing her to do the above quickly, with minimal confusion.

3.1 Interface design

The overall interface design incorporates aspects from both the curated data view interface (Figure 2-2) and the query building interface (Figure 2-1). From the query building interface, we retain the main query input bar and a form of autocomplete; from the curated data view we retain the side panel. A diagram of the interface is shown in Figure 3-1. It is similar in appearance to a Google search, in order to give the affordance that the query input text *does not have to be in query language*–instead, it can be a free-form keyword search, making locating data much easier, as it places less

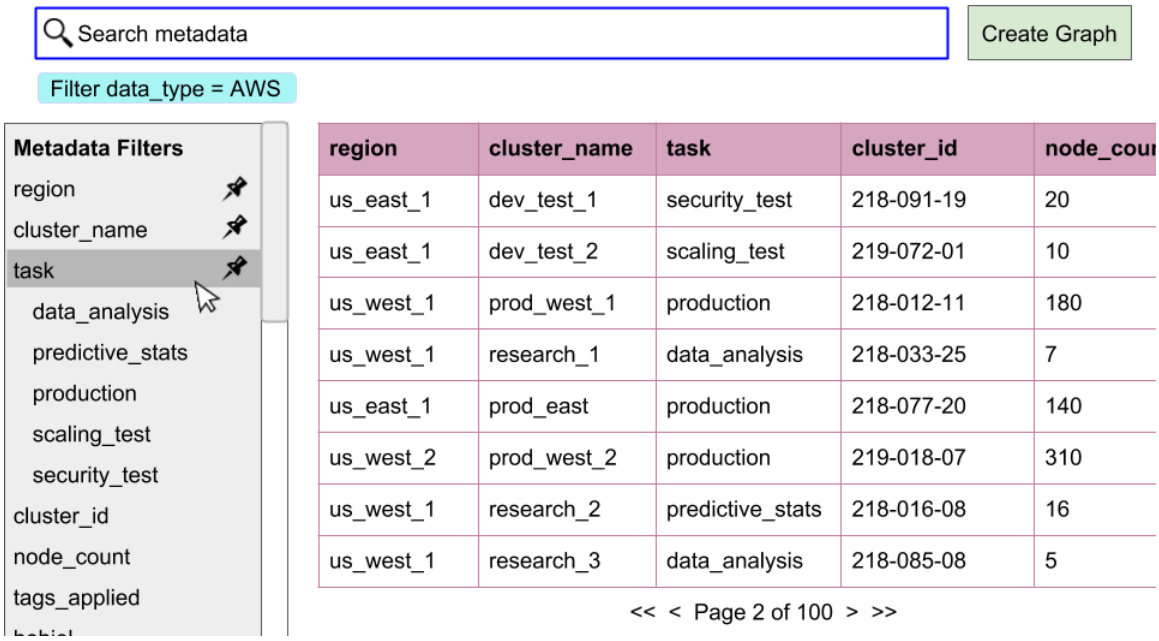


Figure 3-1: Sample interface view. The top search bar (bright blue) allows free-text search for keys and values. The combination of the search terms and the filters applied (light blue, below the search bar) form the query. The left panel (grey) shows an exhaustive list of keys and their values once clicked. The table (magenta) shows the structured data that matches the query, one data point per row.

burden of knowledge on the user. For example, if Sandra was searching for business structured data related to http errors, she may type "http error" instead of being forced to type "HTTP_error" to exactly match the desired item (meaning that she must know that the delimiter is an underscore as opposed to a dash, dot, space, etc.) or "*http*error*" to specify imprecision using query language syntax (which she may not know, as it is platform specific).

Another key set of design decisions centers around the interaction between the query bar, the side panel filters, and the main view. The query building process centers around successive refinement–a process of iteratively adding onto a query by applying more filters via a combination of user knowledge and interface aid (for example, using autocomplete or looking at results for a broader query to discover the best filters to apply to find the desired items). With this knowledge, we decide to have the central results table (further discussed in Chapter 4) be the autocomplete–that is, it is a preview of the results that is updated *each keystroke*. Deeper interaction is

possible with this table than for standard autocomplete (for example, applying filters via clicking on its entries, paging through results, seeing the number of matched results to understand how much more search refinement is necessary). At the same time, the table still gives the same benefit of quick refinement as autocomplete. Additionally, having a results table for the autocomplete as opposed to a list of matched items allows the user to see how other key-value pairs are related to their currently selected key value pairs. In addition to the results table being updated at each keystroke, the side panel is also adjusted at each keystroke, allowing users to see which key-value pairs are invalid once they specify a search term.

The duality of the side panel and the table allow the interface to provide both width and depth to the structured data view. The side panel provides exhaustiveness (depth): each key is shown, and for each key each value may be shown if the user chooses. This is especially helpful when Sandra wishes to validate her data collection, as it is all laid out for her to view. Here, she is able to form conclusions such as, "The number of data collection regions for AWS is off. We shouldn't be collecting data in Australia. I need to fix this." The table shows displays system "width", i.e. how each key-value pair relates to the others in order to form pieces of structured data. Here, she is able to form connections such as "AWS data generally comes with a few key value pairs that I care about: e.g. cluster, metric, region, where region is a broader category than cluster."

Finally, the design incorporates the user's mental model through the concept of pinned keys in the side panel. With pinned keys, Sumo Logic may suggest certain keys they believe are relevant to the user, and the user can refine the set of relevant keys to suit their own needs. For example, as a business metrics administrator, Sandra might pin the keys for "metric type" (CPU Latency, Memory Utilization) and "region" (us-east-1, us-west-1). These relevant keys rise to the top of the side panel, and they affect which key columns are shown most prominently in the results table (pinned keys are pulled to the left so they are visible without scrolling).

3.2 Results

The evaluation of this interface consists of two parts: subjective analysis from user tests, and objective comparison of the same task done in the original interface and the updated interface described above.

3.2.1 Benchmarking against original interface

The benchmarking task, which represents a common use case, is as follows: successively refine a query with four search parameters (i.e. four key value pairs which need to be specified, with varying degrees of knowledge about the exact wording of each) in order to create a graph of the structured data.

The original interface follows the pattern shown in Figure 2-1. In order to perform this task using the original interface, Sandra would perform the following actions:

1. Type the best known key value pair in query format, using autocomplete.
2. Type the second best known query parameter, filling it using autocomplete.
3. Attempt to locate the third query parameter using autocomplete, but as this term is less-well known, the wording is incorrect (e.g. "500-http-error" instead of "http_500_error"). Autocomplete does not help in this case, so Sandra specifies the unknown using query language ("*http* *500* *error*), and runs the query in order to see the results.
4. After the query results load, scroll through them to find the exact term desired. Refine the query with the exact term ("http_500_error"), and run the search again in order to find the final search term, which Sandra will recognize when she sees it.
5. Search the new query results and find the final term.
6. Modify and run the query one final time to obtain the desired graph.

In order to perform this task in the new interface, Sandra would do the following:

1. Type the best known value from the key value pair, checking the results preview to see if she typed it correctly.

2. Repeat (1) to obtain the value in the second-best-known key value pair.

3. Type "500," see the matching key-value pair in the results table, and click on it to apply that filter to the query.

4. Use the side panel to find the key for the final search parameter, then load the values for the key and click on the value of interest to apply it as a filter to the search.

5. Run the query to obtain the graph. (At this point, if Sandra chooses to, she may pin those four keys in order to have an easier querying experience with them in the future.)

The bottleneck of the old interface lies in running the query: this step will occur on a time scale of minutes for a system comprising several tens of million pieces of structured data, as all the data matching the query must be loaded: both the key-value pairs, as well as the underlying business metrics they represent. However, this is an essential part of the query refinement process in the old interface, and must be done multiple times in most cases. This issue is addressed in the new interface, which loads only a preview of the results without loading the underlying business metrics data, cutting the load time to milliseconds for a system of the same size (thus it can be run at every keystroke).

The bottleneck of the new interface lies in querying for all the values of a particular key, an operation which occurs on a time scale of several seconds for several tens of millions of metrics, However, that load time is amortized by caching.

Thus, if we compare the overall time taken to complete the benchmark, the old interface takes several minutes to perform the task (discounting the time Sandra takes to scan the table of all results matching the query), while the new interface takes under one minute (discounting time taken to scan the list of keys and values).

Additionally, if we compare the user experience when completing the benchmark, the new user interface provides a better long-term experience, as it allows Sandra to gain an understanding of the structured data as she interacts with it via its use of dual "deep" and "wide" data views. The interface also improves Sandra's experience by devoting all of its space to the query-building task, rather than placing that information below a graph which is empty until the very end of the task. Finally, it allows Sandra to easily leverage different levels of knowledge about her data, something that is not provided by the old interface.

3.2.2 User testing

We performed 20 in-depth user tests in order to evaluate the new interface, and the use cases we gathered from asking users about their workflows were used to develop the benchmark task.

The overall reaction to the new interface design was very positive, with 95% of users preferring the new interface. Users frequently referred to the reduction in latency for their most common use case as the reason for the preference. Additionally, the ability to see an exhaustive view of their data was also perceived as useful, both for the data validation use case outlined in the original user scenario with Sandra and for gaining a deeper understanding of how their system is organized. Many users reported that having pinned keys was useful, as they use a handful of keys much more frequently than other keys. Additionally, users enjoyed the smaller interaction optimizations, such as being able to click on the table to apply a filter from it, though some users found several of these interaction mechanisms unintuitive, as they break common existing design patterns, such as clicking and sorting interactions for tables established by the spreadsheet model.

For more information on this user test, see Appendix A.

Chapter 4

Search Result Prioritization

Search result prioritization is an important concept in structured data discovery, particularly when the number of results matching a particular query is greater than the number that can be displayed to the user. This is often the case, as queries begin broadly and are successively refined according to the process described in Chapter 3.

The value of diversification lies primarily in the successive refinement step. With diversification, there is a higher chance that the user will be able to locate the desired data or query refinement more quickly, with fewer keystrokes, and more accurately, by simply copying the required refinement from the data shown as opposed to trying to remember or guess what it is if it does not appear in the results. For example, assume Sandra is searching for data about HTTP "Not acceptable" errors (i.e. 406 errors, but she does not remember the number). Her first search for "HTTP error" is more likely to show the relevant key-value pair if the search results are diverse than if not, especially considering that 406 errors are quite rare.

Relevance provides similar speed and accuracy benefits by attempting to recall a user's context and thus prioritize results according to some personalization. For example, assume Sandra's teammate has to replicate Sandra's work of graphing HTTP "Not acceptable" errors for a new computing cluster. Since this teammate has a use pattern similar to Sandra's, and Sandra recently showed interest in 406 errors, 406 error results would be more heavily prioritized when her teammate searches for "HTTP error".

4.1 Diversity

This section addresses the diversification of structured data results when the relevance model is flat. In other words, we assume that all of the results have equal relevance, as each result to be diversified has 100% "relevance" to the query since it matches the query exactly (i.e., we do not correct for spelling mistakes).

4.1.1 Experiments

We perform an experiment to test the value of diversity in structured data result prioritization. For this test, we compare four different sets of data: data prioritized according to a diversification scheme which does not incorporate the user's mental model, data prioritized according to a diversification scheme which incorporates the user's mental model, data without any prioritization scheme (i.e. the control), and data prioritized randomly.

Techniques of Diversification

The following descriptions assume we have obtained the subset of data matching the query from all the data in the system. This matching data will be referred to simply as "the data," of size m, with n result displayed per page.

·**Diversification without mental model** This model forms the baseline diversification algorithm, with no modifications to accommodate understanding of the user's mental model. In this algorithm, the values for a structured data point are treated like a bag of words, i.e. each value for every key-value-pair defining the structured data point becomes one "word" stripped of its key. This "bag of words", or set of values, can then be used in a distance function based on Jaccard Indices, calculated simply as:

$$distance = \frac{\mid X \cap Y \mid}{\mid X \cup Y \mid}$$

where X and Y are two sets of values representing two structured data points.

For example, say we have two metrics:

X = {region: us-east-1, metric: cpu-latency, cluster: prod-2, id: iax14df3},

Y = {region: us-east, metric: memory-use, cluster: prod-1, id: vf43dx22}. Our bags of words would be X_b = {us-east, cpu-latency, prod-2, iax14df3},

Y_b = {us-east, memory-use, prod-2, vf43dx22}. The Jaccard Index of X and Y would be:

$$\frac{|\ \{\texttt{us-east}\}\ |}{|\ \{\texttt{us-east, prod-2, prod-1, cpu-latency, iax14df3, memory-use, vf43dx22}\}\ |} = \frac{1}{7}$$

The algorithm runs as follows: one seed data point is selected from the data and added to the final results subset, S. Next, a data point is added to S by calculating the data point which has the greatest summed distance from all the points in S. This step is repeated until S contains n data points, then S is returned. Pseudocode for the algorithm appears in Algorithm 1 below.

Algorithm 1 Calculate a diverse subset of results

Require: D = list of all results, n = size of desired subset

```
s ← []
while size(s) < m do
    s_{i+1} ← max(Σ_{n=1}^{i} Jaccard(s_n, x)) for all x | x ∉ s AND x ∈ D)
end while
return s
```

·**Diversification with mental model** This model modifies the baseline diversification algorithm by incorporating the user's mental model. This is done in the following way.

The user labels keys important to them in the system through "pinning". For example, if Sandra's workflow requires her to frequently locate metrics by searching for metric type, cluster, and region, she may "pin" those three keys so that they appear more prominently in the results. Additionally, in lieu of user-provided keys, we provide default pinned keys which incorporate our knowledge of what is probably important to the user. These "important keys" are a good

indicator of what items the user cares about. Thus, in this modified diversification model, we weigh these keys more heavily in the diversity distance function by eliminating consideration of non-pinned keys altogether.

Second, we modify the "bag of words", or set of values, by tokenizing the values or words, e.g. breaking each value into smaller words via space delimiters, dash delimiters, etc. This is to accommodate the fact that often, the user's values are formatted as

<descriptor>[<descriptor><delimiter>]+<number>

where the descriptor is a word describing some feature of the data, a delimiter is a space, underscore, dash, or dot, and the number indicates a particular node, cluster, server, etc. Without tokenization, the values "ohio-3" and "ohio-2" would be considered as diverse as "alaska-1" and "ohio-3".

Thus, the difference between diversification with mental model and diversification without mental model lies in the Jaccard Index calculation. If we calculate the Jaccard Index using the same example metrics in the previous section, i.e. $X = \{$region: us-east-1, metric: cpu-latency, cluster: prod-2, id: iax14df3$\}$ and $Y = \{$region: us-east, metric: memory-use, cluster: prod-1, id: vf43dx22$\}$, then our bags of words would be $X_b = \{$us, east, cpu, latency, prod, 2$\}$, $Y_b = \{$us, east, memory, use, prod, 1$\}$, assuming our pinned keys are "metric", "region", and "cluster". The Jaccard Index of X and Y would be:

$$\frac{|\,\{\texttt{us, east, prod}\}\,|}{|\,\{\texttt{us, east, prod, memory, use, cpu, latency, 1, 2}\}\,|} = \frac{3}{9}$$

·**Control: no diversification** This model forms the baseline control that naively prioritizes results with no modifications, i.e. by returning the first n results which match the query. Because the data in the system is stored in a way that naturally groups similar data together, these results are virtually guaranteed to be very homogeneous. Data is stored this way because related metrics (e.g.

metrics for the clusters in the same region or performing the same task), are usually added into the system database together.

·**Diversification via randomization** This model serves as a second form of control: could purposeful diversification perform better than randomization? Randomization is done simply by selecting n random numbers from $[0, m)$, and returning the subset composed by the data at those indices.

Complexity analysis

The analysis given below assumes that the data subset matching the query is already loaded, a step with time complexity $O(z \cdot v \cdot q)$, where z is the number of data points, v is the number of key-value pairs per data point, and q is the number of parameters in the query.

· **Diversification without mental model** Since the complexity of computing the Jaccard Index between two data points is $O(v)$ (assuming use of the MinHash approximation algorithm)[4], and this operation is performed $O(v \cdot m + 2v \cdot m + 3v \cdot m + ... + nv \cdot m) = O(v \cdot n^2 \cdot m)$ times, its time complexity is $O(n^2 \cdot m)$.

· **Diversification with mental model** Since the alterations to accommodate mental model only affect the Jaccard Index calculation, whose complexity remains approximately the same (taking into account the reduction in number of values from only taking pinned values, and the increase from tokenizing them), its time complexity is the same as the above, that is $O(v \cdot n^2 \cdot m)$.

· **Randomization** Since this must fetch n random results, the runtime complexity is $O(n)$.

· **Control** Since this simply returns the first n results, the runtime complexity is $O(n)$.

Type A	Type B	% preference for Type A
Random	Control	53
Diversification 2	Control	53
Diversification 1	Diversification 2	66
Diversification 1	Random	74
Diversification 1	Control	70
Diversification 2	Random	58

Table 4.1: Pairwise preference percentages for the four methods of diversification, where percent preference indicates the percent of time that people selected the results from Type A as more useful than those of Type B.

4.1.2 Results

For this experiment, we ran a survey among users who fit Sandra's use case profile, as described in Chapter 1.2. This survey asked participants to answer seven questions, where each question provided a different query and four randomly ordered subsets with 10 data points each that matched the query. These subsets were populated according to the four algorithms described in Chapter 4.1.1. Given the query and the four datasets, the user was asked to create a total ordering on the sets based on their subjective view of which sets were more "useful" if they had to continue to refine the query.

The survey had a 50% response rate among qualified respondents, and yielded approximately 400 data points for analysis.

Statistics

The most preferred ranking based on the preferences described by the users was Diversification 1 (without mental model), Diversification 2 (with mental model), Random, and Control last. Among these, the preference distance between Diversification 1 and Diversification 2 was largest, and the preference distance between Control and Random was the smallest, though the the preference distance between Diversification 2 and Control was relatively small too. We list the pairwise percent preferences in Table 4.1.

We also analyze the data a different way–by analyzing the preferred subsets to

Diversity Metric	Correlation Coefficient	P-Value
diversity of all values	0.385	0.043
diversity of pinned values	0.354	0.040
diversity of all values tokenized	-0.285	0.041

Table 4.2: Correlation between diversity and perceived usefulness

discover what people perceived as useful regardless of the method of creation.

We calculated the correlation coefficients as follows. For each subset generated by each method, we generated a "usefulness score" by summing its rankings from respondents (if a user ranked this set first among its peers, it is scored 3 points, second it is 2 points, third is 1 point, last is 0 points). We then computed the "diversity score" for that subset using Jaccard indices and summing all the pairwise distances. The correlation coefficient between the two scores was calculated using the Pearson method.

This yielded the results shown in Table 4.2.

Analysis

Though there was a clear user preference for diversified results, the correlation coefficient between actual diversity and preference was low. This suggest that there are other significant factors which account for usefulness that are not addressed by these metrics. Additionally, tokenizing values had a negative effect on user preference, as seen with the negative correlation coefficient. Qualitative analysis of the diversified subsets suggests that diversity is difficult for the human eye to discern in many cases, so the marginal utility of diversity may be low most of the time.

4.2 Relevance

We perform a second experiment to test the value of relevance in structured data result prioritization. For this test, we compare four different sets of data: data prioritized according to relevance via the user's most recent query parameters, relevance via the user's most common query parameters, relevance via the user's organization's most

common query parameters, and a control with no relevance treatment.

4.2.1 Background

Prior to performing the experiment, a brief analysis of historical user queries was performed in order to predict whether relevance would play a significant role in usefulness of prioritization. Specifically, 10,000 user queries over the course of 30 days were analyzed to see how repetitive a given user's queries were. The results were as follows.

For a given user,

- 87% of queries use the most common key-value-pair query parameter
- 75% of queries use the second most common key-value-pair query parameter
- 50% of queries use the third most common key-value-pair query parameter
- 43% of queries use the fourth most common key-value-pair query parameter

For a given organization,

- 82% of queries use the most common key-value-pair query parameter
- 75% of queries use the second most common key-value-pair query parameter
- 52% of queries use the third most common key-value-pair query parameter
- 44% of queries use the fourth most common key-value-pair query parameter

This indicates a high degree of repetition in queries, so based on this information relevance should have a high value in prioritization.

4.2.2 Experiments

For this experiment, we performed a combination of a survey and user tests that were personalized for each user's personal relevance statistics. The four methods of

personalization were: data prioritized according to the user's most recent query parameters, data prioritized according to the user's most common query parameters, data prioritized according to the user's organization's most common query parameters, and a control with no relevance prioritization. This data was gathered from the user's queries over the course of the 30 days prior to the user test.

4.2.3 Results

The results of both tests were inconclusive. No statistically significant statistics could be drawn from the survey responses, as the expressed preferences were split approximately evenly among the four options, and many respondents choose to opt out of ranking because they did not recognize any difference in utility among the four options. When various respondents were personally interviewed, they stated that while relevance does matter, their preferences vary widely by use case and level of expertise. For example, being able to view their most common query parameters is a moot point if they are a new user or a user looking for something new; when they have this data populated in a stable way, it is also moot since they know exactly what they want and are not "discovering" the data.

4.3 Summary

Based on the above results for diversity and relevance, it is clear that additional work must be done to discover the best metrics for usefulness in structured data search result prioritization, so that we can justify modifications that require additional runtime complexity.

Chapter 5

Hierarchy Induction and Structured Data Visualization

Structured data visualization is an important topic which is inadequately addressed in all current solutions to the structured data discovery problem. A good structured data visualization allows Sandra to understand her system's structured data at a glance. It does this by leveraging the data's structure to provide insight into the system's organization and any potential problems with the data's structure without making Sandra sift through each piece of data individually.

The structured data visualization proposed here is based on a sunburst diagram [Figure 5-1]. This visualization provides a two-fold benefit. First, it is able to represent a single piece of structured data as one traces a straight-line path from the center of the diagram to the outside, where each segment represents a single key-value pair. Second, it is able to group the data along the lines of its structure through the concentric rings. Thus, one can visually understand what proportion of the data shares a particular key-value pair.

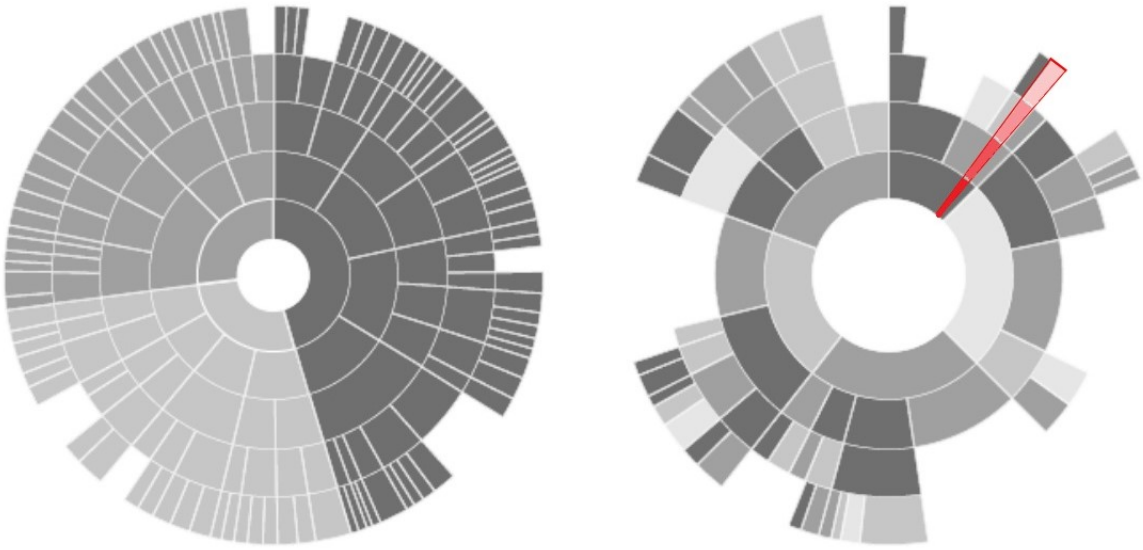


Figure 5-1: Two example sunburst diagrams. A straight-line path from the center of a diagram to the outside defines a structured data point (an example is highlighted in red). Thus, if 100 data points are represented, each data point is represented by a 3.6 degree slice of the circle. The diagram on the left shows a well-structured data set, which is only missing values at the outermost levels, while the one on the right has data missing from two of the outermost levels. The left diagram has a color scheme based on the division of the innermost level, while the right diagram's color scheme is not clear from visual inspection.

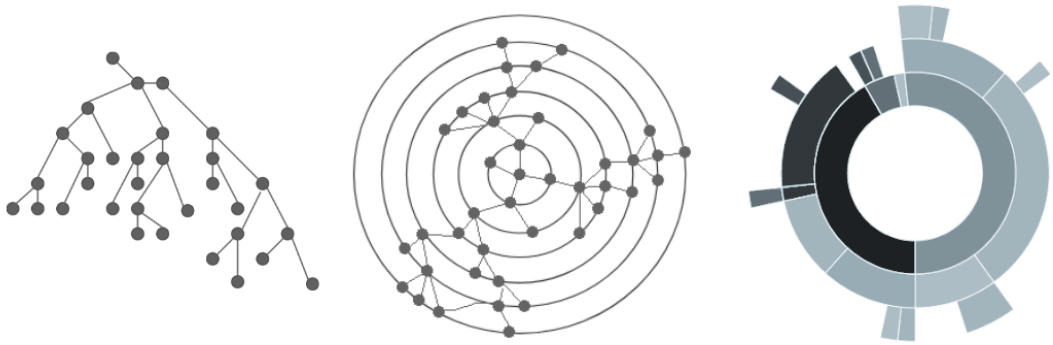
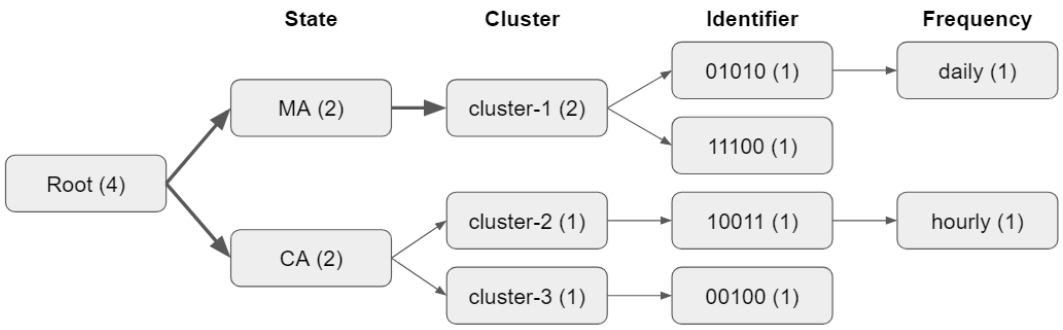


Figure 5-2: The evolution of a tree (left) into a sunburst (right). The first transition transforms the tree into polar coordinates, and the second transition adds the appropriate volume to each node.

5.1 Understanding the Sunburst Diagram in Relation to Structured Data

A simple way to understand the sunburst diagram is as a tree, with the root node in the center and the leaves on the outer radii, as shown in Figure 5-2. Each key is a level of the tree, and each node in a level is defined by the level key and a value of that level key. A node represents a set of structured data points whose key value pairs match every single key value pair in that node's path. A node cannot exist if it does not represent any data points. Figure 5-3 is an example of a set of data and a valid tree representing it.

By transforming the data from a simple tree to a sunburst, Sandra gets the additional value of being able to see the volume of data in each bucket, its siblings, and its children. For example, a sunburst is able to answer a question like: how many data points have ($region = us{-}east$ and $cluster = prod{-}2$) relative to ($region = us{-}east$ and $cluster = prod - 1$)? via a visual representation, as shown in Figure 5-4.



	State	Cluster	Identifier	Frequency
Data Point 1	MA	cluster-1	01010	daily
Data Point 2	MA	cluster-1	11100	
Data Point 3	CA	cluster-2	10011	hourly
Data Point 4	CA	cluster-3	00100	

Figure 5-3: A tree (above) and the structured data it represents (below). The numbers within each tree node are the number of data points which it represents. In a sunburst diagram, these numbers would be represented visually.

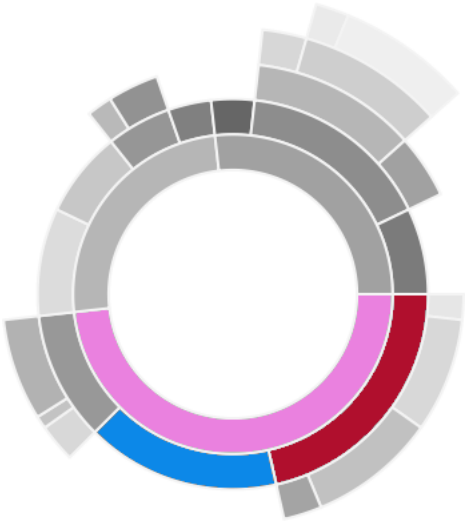


Figure 5-4: A sample insight: within the *region=us-east* data (pink), which comprises a little less than half the data in the system, we can see that roughly 45% have *cluster=prod-2* (red) and roughly 35% have *cluster=prod-1* (blue).

5.2 Hierarchy Induction

A sunburst diagram of structured data has the potential to show Sandra at a glance how her data is organized. However, in order to realize this potential, the diagram must have a good organization principle, i.e. one that makes intuitive sense to Sandra and visually groups similar data together. The "organization principle" of a sunburst is its hierarchy, which determines the various branches of the data tree. For example, if we choose our hierarchy to be $region \prec cluster \prec metric$, then the first level of the tree (corresponding to the innermost ring of the sunburst) would be *region*, the second would be *cluster*, and the third would be *metric*.

Since structured data does not have any intrinsic hierarchy (i.e. each key-value pair has equal weight), and the sunburst diagram is hierarchical in nature, we must create a hierarchy that imposes a total order on the set of all keys in order to create the sunburst diagram.

5.2.1 Experiments

We experiment with two novel techniques of hierarchy induction: one based on cardinality, i.e. the number of unique values for each key, and another based on information entropy, a measure that incorporates cardinality as well as quality of data distribution (where a more even distribution of data across the keys is scored more highly).

Methods of Induction

We begin this section by addressing the concerns that arise when there is a "null" value, i.e. when a structured data point does not contain a value for a given key. This occurs often, as structured data may come from various sources which represent fundamentally different systems. (Related work on hierarchy induction does not face the "null value" problem, as their databases have strict schemata.)

In both methods of hierarchy induction described here, we treat these null values as **unique** values. Thus, if we were counting the number of unique values for a key, each time we encounter a null value for that key we would add one to the count of

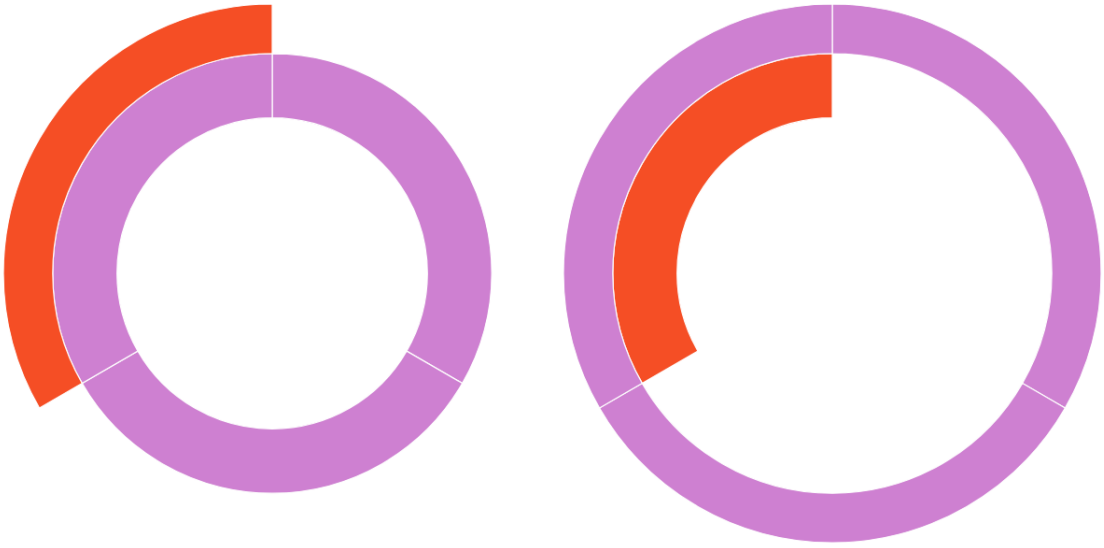


Figure 5-5: An illustration of ordering accounting for nulls (left) and not accounting for nulls (right). Purple indicates *city*, (reading from the 12:00 position clockwise: Cambridge, Sommerville, and Boston), and red indicates *city_income_tax* (1%). Missing blocks indicate null values, i.e. *city_income_tax*=null.

unique values.

This is to avoid biasing the system in an illogical way. For example, if there was one key, *city*, which had three unique values– Boston, Cambridge, and Somerville–and was defined for each structured data point, and a second key, *city_income_tax*, which had a value–1%–for only *city*=Boston and was null for *city*!=Boston, if we created a hierarchy according to the number of unique values without accommodating nulls then *city_income_tax* would be ranked before *city*, creating a very odd diagram, as shown in Figure 5-5.

· **Cardinality** The idea behind the method for induction via cardinality is simple. Count the number of unique values for all keys, addressing nulls as described above. Then rank the keys according to the the counts, with the lowest count at the top of the hierarchy and the highest count at the bottom. A pseudocode procedure for this is outlined in Algorithm 2 below.

· **Entropy** To calculate induction by entropy, we need to store two types of information: the unique values per key, as well as the number of data points which

Algorithm 2 Calculate a cardinality hierarchy

Require: D = list of all results, K = list of all keys
  for $x \in D$ **do**
    for $k_i \in K$ **do**
      $cardinality_{ki} \leftarrow cardinality_{ki} + 1$ if $x[k_i]$ has not been seen or is NULL
    end for
  end for
  return $sort(cardinality)$

fall under each unique value. Given that, the formula to calculate information entropy of a given key is as follows:

$$H(k) = -\sum_{i=1}^{n} P(v_i) log(P(v_i))$$

where $H(k)$ is the entropy of a key, k, n is the number of unique values of k, v_i is the ith value of k, and $P(x)$ is fraction of data points which have the value x.[17]

The entropy hierarchy is calculated iteratively–that is, in the first calculation, the key with the lowest entropy score is chosen. Then, the data is split according to that key, and *in each branch of the data*, the entropy score for the remaining keys is calculated and then *summed across the branches*. The key with the lowest summed entropy score is chosen next, and the previous step repeats until each key is chosen.

Algorithm 3 Calculate an entropy hierarchy

Require: d = list of all results, k = list of all keys
  $buckets = [d]$
  while $size(hierarchy) < size(k)$ **do**
    for b in $buckets$ **do**
      for $k_i \mid k_i \in k$ AND $k_i \notin hierarchy$ **do**
        $entropy_{ki} \leftarrow entropy_{ki} + Entropy(k_i)$
      end for
      $hierarchy \leftarrow hierarchy + min(entropy)$
      $buckets \leftarrow flatten(b.split(value)$ for $value$ in $min(entropy)$, for b in $buckets)$
    end for
  end while

Complexity analysis

The runtime complexity of these algorithms is as follows.

· **Cardinality** Since cardinality can be calculated with a single pass through the data, its complexity is $O(n \cdot m)$ where n is the number of data points, and m is the number of key-value pairs per data point.

· **Entropy** Entropy also requires making a pass through each data point and its key-value pairs, but since the entropy hierarchy is built iteratively, its complexity is $O(n \cdot m \cdot k)$ where n is the number of data points, m is the number of key-value pairs per data point, and k is the number of unique keys in the data set.

5.2.2 Results

Tests against ground truth data

For the structured data Sumo Logic uses, there is a data source which contains its own hierarchy–and thus, any data which comes from this source will contain a "ground truth" hierarchy. Though the data is treated the same (i.e. it is flat in our system), if data is known to come from this data source then the hierarchy can be easily recovered.

For this test, we evaluate 15 distinct data sets from this "ground truth" data source. Each set contains 500 data points. We run the algorithms over each set and obtain a hierarchy. From that hierarchy, we extract only the hierarchical keys, and judge the hierarchy's correctness according to the Spearman Score. To calculate the Spearman Score, we simply sum the displacements of all the list items from their proper places.[20] For example:

$$\text{Induced hierarchy order} : A, F, B, D, C, E$$

$$\text{Correct hierarchy order} : A, B, C, D, E, F$$

$$\text{Displacement} : A = 0, B = 1, C = 2, D = 0, E = 1, F = 4$$

$$\text{Total Displacement} : 0 + 1 + 2 + 0 + 1 + 4 = 8$$

From this test, the results were as follows.

· **Cardinality** Cardinality recieved an average Spearman score of 2.53, with a standard deviation of 3.46. For reference, a swap of two adjacent values recieves a Spearman score of 2.

· **Entropy** The average Spearman score of the entropy hierarchies was 3.87 with standard deviation of 4.16. For reference, a swap of two pairs of adjacent values, or a swap of two values which are separated by a single value both result in a Spearman score of 4.

· **Random** We created a control by randomly ordering the hierarchical keys. This method received an average Spearman score of 33.87, with a standard deviation of 6.91.

Qualitative analysis of results on all data

Qualitative double-checking of the induced hierarchies over all keys (not just ones with ground truth) showed that the total ordering for both algorithms corresponded well with the expected ordering, with the most commonly shared, distinctive keys at the top of the hierarchy, and the most obscure, individualized keys at the bottom. Because cardinality performed better in our ground truth test and it has a better runtime, we choose it as the preferred method of hierarchy induction.

5.3 Structured Data Visualization

Once the results of hierarchy induction validated cardinality as a better algorithm, the following experiments of user understanding, usability, and preference were conducted upon the structured data visualization sunburst created with a cardinality hierarchy.

5.3.1 Experiments

Applications of hierarchy induction

Three different ways of using the cardinality algorithm to generate and use hierarchy were tested:

· **Universal ordering** This is the cardinality algorithm output described above. There is one universal ranking of keys that dictates the branching structure of the sunburst diagram.

· **Local ordering** This is a modification of the cardinality algorithm such that each branch of the data may have its own local hierarchy independent of its siblings. This local hierarchy is calculated using the cardinality algorithm, and the key with the lowest cardinality is picked at each level for each branch. This minimizes the number of splits of the data, i.e. attempting to create the most compact data summary possible.

· **Collapsed ordering** This is a modification of the data visualization model that uses the universal ordering of keys produced by the original cardinality algorithm. In this version, any time the data does not split on a key, the parent and the child are collapsed into a single node. This makes the diagram cleaner and easier to read by only highlighting the points where the data splits.

Figure 5-6 and Figure 5-7 illustrates these three concepts.

Use of color

Three different uses of color for the visualization were tested:

· **One color per key** This use of color guaranteed that a single key (mostly corresponding to one level of the diagram) was one color, though two keys could be assigned the same color.

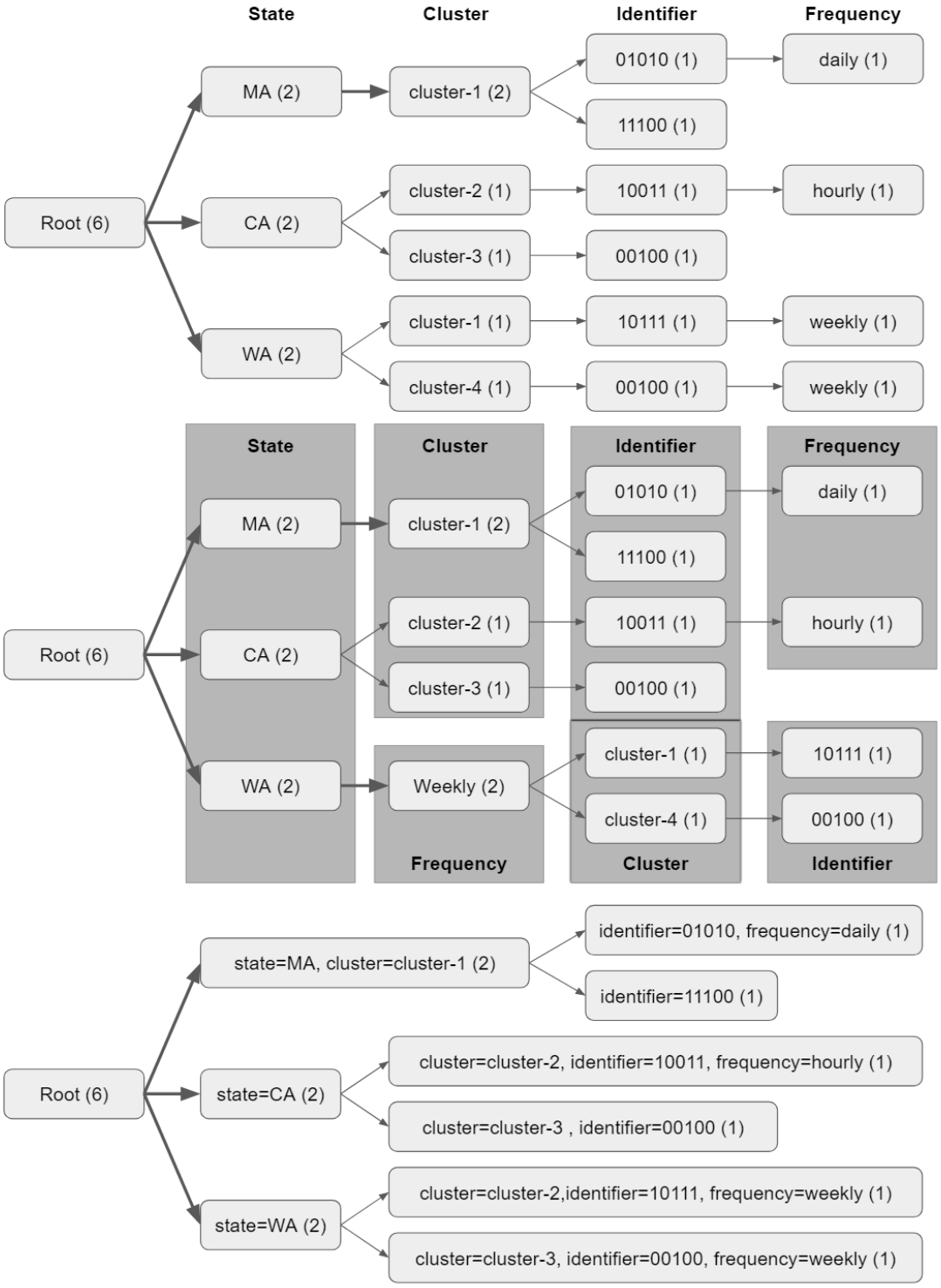


Figure 5-6: From top to bottom: the universal ordering tree, the local ordering tree, and the collapsed tree for the same dataset.

Figure 5-7: [Continuation from Figure 5-6] From top to bottom: the universal ordering tree visualization, the local ordering tree visualization, and the collapsed tree visualization for the dataset in 5-6. The diagram can be read starting at the 12:00 position moving clockwise. The blue highlighted sections correspond to differences between the top diagram and the highlighted diagram.

· **One color per key-value pair** This use of color guaranteed that a single key-value pair (corresponding to one or more segments of a level or radii) was one color, though two key-value pairs could be assigned the same color.

· **Color corresponds to collapsible sections** This use of color roughly corresponded to a set of data which could be collapsed (in other words, parent nodes and an unbroken line of their descendents with no siblings) was one color, though two collapsible sets could be assigned the same color.

Figure 5-8 illustrates these concepts.

Interaction method

Finally, three methods of hover interaction were tested. Specifically, three different types of information highlighting were used when a user hovered their cursor over a key-value segment of the visualization.

· **Path highlighting** With path highlighting, a node and its entire ancestry are highlighted upon hover. This illuminates the relationship between a node and its ancestors.

· **Path and all matching parts highlighting** This highlighting scheme includes path highlighting, plus highlighting of any node of the graph which matches any part of the path. This illuminates the equality relationship of a node and its ancestors to any other equal other node in the graph.

· **Path and hovered matching parts highlighting** This highlighting scheme includes path highlighting, plus highlighting of any node is equal (i.e. has the same key-value pair) to the node being hovered.

Figure 5-9 illustrates these concepts.

5.3.2 Results

In-depth user tests were conducted with seven users whose normal use case fits the one described in Chapter 1.2. Each user was shown a data visualization representing

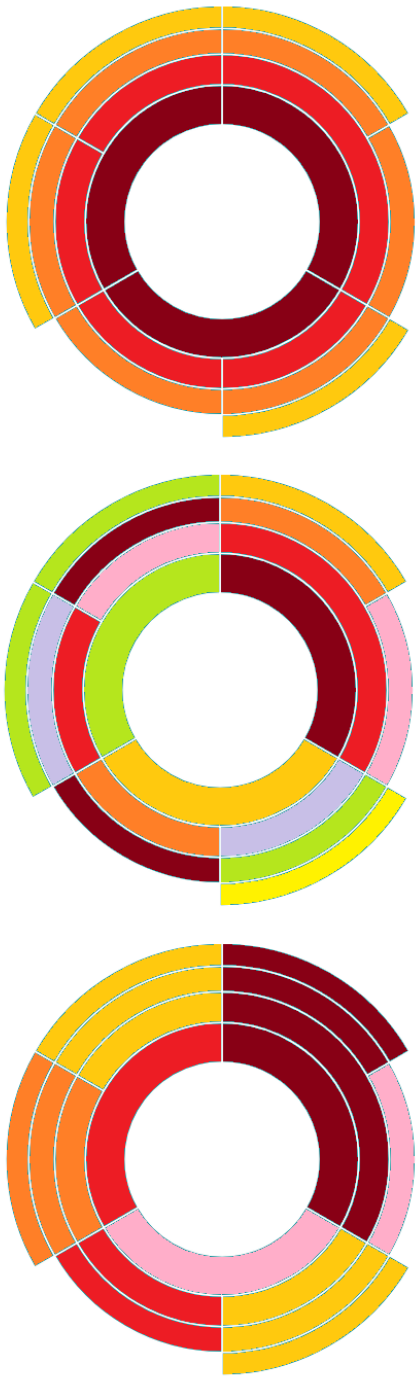


Figure 5-8: From top to bottom: one color per key, one color per key-value pair, and color corresponding to collapsible sections for the same universal ordering diagram in 5-6.

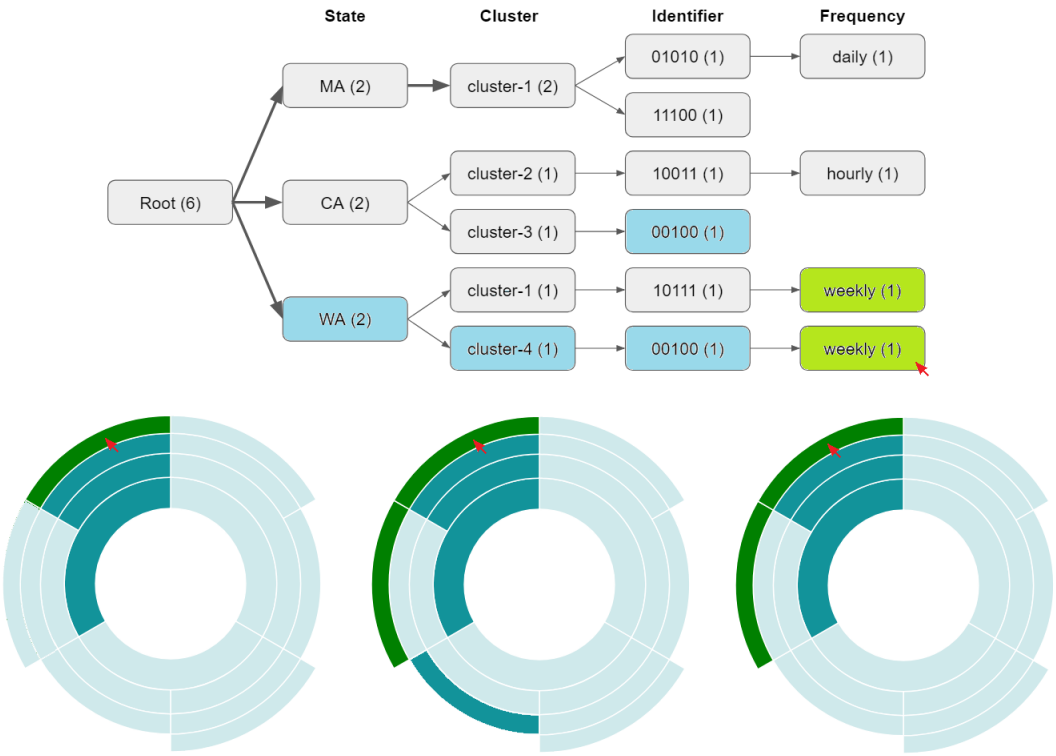


Figure 5-9: From bottom left to bottom right: path highlighting, path/all matching parts highlighting and path/hovered matching parts highlighting for the same universal ordering diagram (top) in 5-6. The hovered node is indicated with the red arrow, and the tree diagram is reproduced on the right with the sections of interest highlighted.

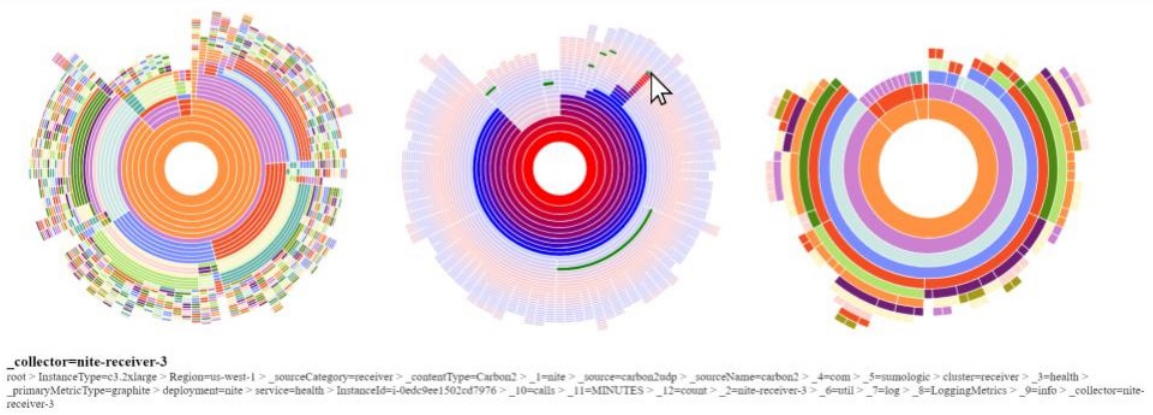


Figure 5-10: A screenshot of the user test interface. Pictured is a test of the different hierarchy induction methods and color schemes. Notice the tracing of the hover path on the bottom of the screen, as well as the bolded title of the hovered node just above the path.

1000 structured data points that the user uses in a normal working context. An example of the test interface shown to users is shown in Figure 5-10. The results of these tests are summarized below.

Intuitive understanding of visualization

The users' intuitive understanding of the diagram was universally high: each one almost immediately understood, after a short interaction, that each level of the diagram corresponds to a single key, and that the size of the arc taken by a key-value pair corresponds to the proportion of metrics which has that key-value pair. After further interaction, users usually also understand the relationship of a key-value pair to the key-value pairs below it (where below means closer to the center of the circle, and above means farther from the center of the circle): that these form a group of metrics whose definition can be traced, at least partially, from the inside of the diagram to the outside. Additionally, users eager to interact with the diagram and discern its underlying structure.

Usefulness

The users found almost immediate use of and interest in the diagram, as it helped them troubleshoot the structure of their data (multiple users brought up problems with their data definitions as they explored the system). They also found the sunburst "cool," as it slices the data in two ways so they could understand how unique a data point is at each step of its definition, as well as where the variation in their structured data arose.

Preferences

For the experiments outlined below, the most common preferences and rationales were as follows.

· **Applications of hierarchy induction** Most users preferred the single, universal ordering of keys over the collapsed keys or the local hierarchy, though they stated that the other versions were theoretically interesting. The universally ordered version was preferred because it was easiest to understand, and it gave the most accurate pictorial view of the overall system.

· **Use of color** Most users preferred the color corresponding to collapsible sections, as it was the helpful in guiding them toward a less obvious class of insights, i.e. seeing where the data bifurcates. The other color schemes did not illuminate any interesting insights.

· **Interaction method** Users generally preferred the option that highlighted the hover path, plus any key-value pairs that matched the currently hovered key-value pair. This was the least confusing highlighting scheme that simultaneously illuminated the relationship across levels of the sunburst diagram (i.e. between a node and its parents) and along the levels of the tree (i.e. between a node and other nodes in the same level).

Chapter 6

Conclusion

6.1 Summary

The main focus of this study is to find useful ways of managing structured user data. Our primary contributions:

1. We discussed a novel interface for quickly building queries. This interface was unique in its combined use of whole-metric autocomplete, as opposed to single key or single value autocomplete; dual use of exhaustive (deep) listings of values and keys and sampled (wide) listings of whole metrics to increase ease of use; and incorporation of a user's mental model into the search process via pinned keys.

2. We enumerate experiments in ranking structured data search results. The first set of these experiments, related to diversity, showed clear preference for diversity over naive search matching or random matching, though the correlation between our diversity metric (which we use to construct diverse sets of data) and user defined "usefulness" was not strong. The second set of these experiments, related to relevance, was inconclusive, in that a user did not show consistent preference for how results should be prioritized based on their historic context, but seemed to have needs that were highly dependent on their current use case.

3. We investigate different methods of hierarchy generation in order to create a novel structured data visualization. Of the two methods proposed, based on cardinality and entropy, respectively, cardinality performed better both in run-time and accuracy when compared to data with a known hierarchy. We also discuss user preferences for the visualization, which were discovered through in-depth user testing.

6.2 Review of User Scenario

In Chapter 1, we introduced a typical user, Sandra, and her use case. Sandra, as a business metrics system administrator, is responsible for setting up the collection of real-time, business-related, structured data in the system. After completing setup, or adding any new business metrics to her account, her two objectives are validating the setup of the data collection and to generating graphs of business metrics for other engineers responsible for monitoring and troubleshooting. We will now imagine an updated workflow using the tools outlined in this thesis.

Sandra begins by validating the setup of her data. First, she looks at the overview of her entire system, which is presented to her in a hierarchical data visualization. She notices that one section of the data looks particularly jagged, corresponding to missing key-value pairs. By interacting with that part of the visualization, she determines that the problem is with a set of custom metrics, and she is able to see which keys in that set are are missing values, causing the jaggedness. She amends the problem, then sees the overview system visualization is how she expects.

Sandra now moves to generating graphs of business metrics. She enters the query-building interface, which looks like a Google search. For the first few graphs she creates, she has a very clear idea of what she wants, so she types all her plain-text search terms into search box, visually checks that the matching metrics are what she expects using the table, then graphs the query results. For the next set of graphs, she has a good idea of what she wants, but she is less familiar with this aspect of the system. She uses a combination of search terms and filters from the side panel

in order to successively refine her query until it seems to be what she wants. The well-prioritized list of metric results reduces the amount of scrolling Sandra must do in order to understand what effects her search terms and filters have. After finishing each query, she creates the graph as the final step.

Thus, Sandra is more efficient, due to her ability to diagnose macro issues with her system via the visualizations and dive deeply into her system in a way suited to her knowledge level for any given task.

6.3 Future Work

Several questions arise from this study which merit additional research. These are as follows.

The first question relates to the integration of the interface of Chapter 3 with the visualization of Chapter 5. Obviously, they address similar use cases, but they do so in very different ways. Alternately, this question can be formulated as how to present the user with the type of interface they need–a precise, query-building interface, or a general, click-navigable visualization for generating insights–with minimal overhead for the user.

The second question relates to better ways of prioritizing search results. How can we build a model of what type of prioritization a user wants based on the data we have collected about them and other users? This primarily relates to how we can better leverage relevance, but also asks how we can better model diversity so it has a higher correlation with user-defined "usefulness".

The third question relates to scaling and performance. How do we best scale algorithms for visualization generation and search prioritization so that they maintain the low latencies required for acceptable user experience, even when dealing with millions of structured data points? Additionally, what are the implications of distributing the work in various ways on the correctness of the end result?

Appendix A

User Test of Query Interfaces

The following script is an excerpt from the research report generated as a result of the user tests, modified slightly for clarity.

A.1 Research Goals

This research focuses on two objectives:

1. Sandra (for an engineering team) needs to find a metric she in order to create a chart

2. Sandra (for an engineering team and/or entire organization) needs to validate whether all her metrics have been ingested correctly

A.2 Research Questions

1. Is the new way of displaying data (wide and deep) useful?

2. Is the "Google search with filters" model valid?

3. Do users type key-value pairs into the query box?

4. Do users press enter to create filters, or to run the query?

5. What should the order of the results table columns be (in terms of filter-specified keys, search-term-matched keys, pinned keys, etc)
6. Can Sandra successfully find a metric she needs?
7. How usable is the new side panel component? Do users understand the functionality?
8. Is a visualization needed while finding the metric (e.g. the metric chart)? Why or why not?
9. Do users understand the new path to visualize the metric (a second step)?
10. Do users find the new design helpful?
11. Do they understand they can validate whether the metrics are ingested properly using the new design?
12. Which interface do users prefer?

A.3 Summarized Findings

A.3.1 General Sentiment

Once given a full tour of the feature after the test ended, all users who gave a preference stated that the new interface would be immensely helpful and game-changing in their workflows. One user declined to comment on his preference, stating that he believed a different solution–one not focused on solving the query-building issue–was necessary to solve the problem of metric discoverability.

A.3.2 Information Hierarchy

Overall, the information hierarchy in the new interface was difficult to perceive, making it difficult to navigate through all the new UI components on the screen. This difficulty consequently made it difficult for users to know "where to start". Thus,

the interface should provide both implicit (UI navigation cues) and explicit (walk-throughs) guidelines to demonstrate a clear step-by-step hierarchy.

The side panel draws a lot of user attention, but it does not offer enough information to communicate the functionality of the query builder as a whole. Thus, we should consider giving users some information in the results table when they enter the interface, to avoid "blinking cursor" syndrome. Additionally, because the side panel draws all the attention, a lot of users ignored the top search bar in the beginning. We should consider reorganizing the structure of components to make the search bar function more visible.

Users prefer an organized hierarchy to achieve a "funneling down" experience, i.e. users want to feel like they have fewer and fewer tasks things to look at as they click and search. In contrast, this interface seems to offer more paths for the user to take as time goes on, as well as a lot of interface updates at each keystroke. An example of a corrective action for this issue: as users type and search, the results table should update in a more obvious manner that highlights their selection and more clearly shows the fact that unrelated info has been filtered. A design that walks the user through the best way to use the interface based on their knowledge level would help users understand how to best use the UI elements.

A.3.3 Element-specific Feedback

1. Side panel: A long list of keys for the user to browse is a pain point for users with lots of data types. This is especially true when the panel intermixes Sumo keys with customers' custom keys. Consider separating these two categories, or creating a mechanism which would allow users to partition their keys.

2. Search bar: The search bar currently serves two very different function: applying filters (by pressing enter) and searching. This confuses some users, as they don't understand which actions trigger which behaviors. To give users clarity on what to do and what can be achieved, the search bar should only have search functionality. Users could then apply filters from the results of the search.

Additionally, there needs to be a tighter correspondence between the search bar and the results table, for example highlighting the specific items matching the keyword in the table.

3. Table: The table does not communicate its ability to apply filters on click. There needs to be an affordance for this capability, as it is seen as very useful once it is revealed by the researcher. Users stated they preferred the columns matching their filters to appear first in left-to-right order on the table, as it gives visual confirmation of what affect their actions have. Additionally, the table view was strongly preferred to the chart view of the data, though some users expressed a desire to see both.

Bibliography

[1] Sofiane Abbar, Sihem Amer-Yahia, Piotr Indyk, and Sepideh Mahabadi. Real-time recommendation of diverse related articles. In *Proceedings of the 22nd international conference on World Wide Web*, pages 1–12. ACM, 2013.

[2] Zeinab Abbassi, Vahab S Mirrokni, and Mayur Thakur. Diversity maximization under matroid constraints. In *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 32–40. ACM, 2013.

[3] Azin Ashkan, Branislav Kveton, Shlomo Berkovsky, and Zheng Wen. Optimal greedy diversity for recommendation. In *IJCAI*, pages 1742–1748, 2015.

[4] Andrei Z Broder. On the resemblance and containment of documents. In *Compression and complexity of sequences 1997. proceedings*, pages 21–29. IEEE, 1997.

[5] Jaime Carbonell and Jade Goldstein. The use of mmr, diversity-based reranking for reordering documents and producing summaries. In *Proceedings of the 21st annual international ACM SIGIR conference on Research and development in information retrieval*, pages 335–336. ACM, 1998.

[6] Datadog. Graphing, 2018.

[7] Robert Godin and Rokia Missaoui. An incremental concept formation approach for learning from databases. *Theoretical computer science*, 133(2):387–419, 1994.

[8] Sreenivas Gollapudi and Aneesh Sharma. An axiomatic approach for result diversification. In *Proceedings of the 18th international conference on World wide web*, pages 381–390. ACM, 2009.

[9] Jiawei Han and Yongjian Fu. Dynamic generation and refinement of concept hierarchies for knowledge discovery in databases. In *KDD Workshop*, pages 157–168, 1994.

[10] Piotr Indyk, Sepideh Mahabadi, Mohammad Mahdian, and Vahab S Mirrokni. Composable core-sets for diversity and coverage maximization. In *Proceedings of the 33rd ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 100–108. ACM, 2014.

[11] Andreas Krause and Daniel Golovin. Submodular function maximization., 2014.

[12] Sangno Lee, Soon-Young Huh, and Ronald D McNiel. Automatic generation of concept hierarchies using wordnet. *Expert Systems with Applications*, 35(3):1132–1144, 2008.

[13] Yichun Lu. *Concept hierarchy in data mining: Specification, generation and implementation*. PhD thesis, Theses (School of Computing Science)/Simon Fraser University, 1997.

[14] George L Nemhauser, Laurence A Wolsey, and Marshall L Fisher. An analysis of approximations for maximizing submodular set functionsâĂŤi. *Mathematical programming*, 14(1):265–294, 1978.

[15] Inc. New Relic. Infrastructure hosts page: Measure resource usage, 2018.

[16] SS Ravi, Daniel J Rosenkrantz, and Giri Kumar Tayi. Facility dispersion problems: Heuristics and special cases. In *Workshop on Algorithms and Data Structures*, pages 355–366. Springer, 1991.

[17] Thomas D Schneider. Information theory primer with an appendix on logarithms. In *National Cancer Institute*. Citeseer, 2007.

[18] SignalFx. Cloud migration, 2018.

[19] SignalFx. Using the catalog to find metrics, 2018.

[20] Charles Spearman. The proof and measurement of association between two things. *The American journal of psychology*, 15(1):72–101, 1904.

[21] Inc. Sumo Logic. Create a metrics query and visualization, 2018.

[22] Vijay V Vazirani. *Approximation algorithms*. Springer Science & Business Media, 2013.