Exploring Datasets via Cell-Centric Indexing

Jeff Heflin[1], Brian D. Davison[1] and Haiyan Jia[2]

[1]Computer Science & Engineering, Lehigh University, 113 Research Dr., Bethlehem, PA, 18015, USA
[2]Journalism and Communication, Lehigh University, 33 Coppee Dr., Bethlehem, PA, 18015, USA

Abstract

We present a novel approach to dataset search and exploration. Cell-centric indexing is a unique indexing strategy that enables a powerful, new interface. The strategy treats individual cells of a table as the indexed unit, and combining this with a number of structure-specific fields enables queries that cannot be answered by a traditional indexing approach. Our interface provides users with an overview of a dataset repository, and allows them to efficiently use various facets to explore the collection and identify datasets that match their interests.

Keywords

cell-centric indexing, dataset search, exploratory interface,

1. Introduction

The twenty-first century has experienced an information explosion; data is growing exponentially and users' information retrieval needs are becoming much more complicated [1]. Given people's increasing interests in datasets, there is a need for user-friendly search services for data journalists, scientists, decision makers, and the general public to locate datasets that can meet their data needs.

Even though users, under many circumstances, are not experts in the domain in which they search, they should be able to easily use such an application; the query process should be responsive and efficient. The result should provide a general picture of what the dataset is about, and offer enough information for the searcher to know how likely the dataset will contain data that they look for.

Traditional database management systems group data by tables and then organize this data into rows and columns. When users are aware of the database schema, they can construct queries, but what if users are simply trying to find which tables in a large data lake are relevant to their needs? One approach is to simply index information about the table in various fields: e.g., title, description, columns, etc. While this approach may be sufficient for some queries, in many cases the user will not be able to determine if the table is a perfect match until they have downloaded the (potentially large) table.

Studies have shown that understanding what is inside the content of a dataset, rather than simply the dataset descriptions and metadata, could be critical for their evaluation of whether any of the search results sufficiently matches the search need, especially for non-expert users. For instance, a recent user study [2] has revealed that query refinement, as a result of unsatisfying search results, is negatively associated with user experience with the dataset search tools. What reduces the need for query refinement is a preview of the dataset content, which helps users gauge the relevance of the datasets. Similarly, an experimental study that explores novel dataset search engine prototypes has found that interfaces with a content preview feature were perceived as more usable. In particular, non-expert users reported greater benefits from the content preview, as they rated the interfaces with higher levels of usefulness, ease of use, usability, and technology adoption intention, than expert users [3]. These indicate the strong need for understanding the actual content of datasets, even at the cell level.

To enable sufficient query refinement for schema-optional queries, we present the novel concept of cell-centric indexing. The key idea is that we use individuals cells of a table as the fundamental unit and build inverted indices on these cells. These indices provide different fields that index both the content of the cell and its context. For our purposes, the context includes other cell values in the same row, the name of the column (if available), and metadata about the containing dataset. This approach allows us to refine our search by row descriptors, column descriptors or both at the same time. In essence we free the data from how it is structured, and schema information, when available, is merely one of the many ways to locate the data of interest. Thus, we take the view that fundamentally, users are searching for specific data (i.e., particular cells or collections thereof), and

DESIRES 2021 – 2nd International Conference on Design of Experimental Search & Information REtrieval Systems, September 15–18, 2021, Padua, Italy

heflin@cse.lehigh.edu (J. Heflin); davison@cse.lehigh.edu (B. D. Davison); haiyan.jia@lehigh.edu (H. Jia)
http://www.cse.lehigh.edu/~heflin/ (J. Heflin); http://www.cse.lehigh.edu/~brian/ (B. D. Davison); https://journalism.cas.lehigh.edu/content/haiyan-jia (H. Jia)
0000-0002-7290-1495 (J. Heflin); 0000-0002-9326-3648 (B. D. Davison); 0000-0002-8388-7860 (H. Jia)

the tables are merely artifacts of how the data is stored.

We recognize that this approach also has downsides. In particular, an index of cells (and their contexts) will incur substantial storage overhead in comparison to an index of dataset metadata. Moreover, if the desired search result is one or more datasets, at run-time there will be additional processing to assemble the cell-specific results to enable the retrieval and ranking at that level of granularity. However, our cell-centric approach gives us some additional flexibility and we believe that good system design, appropriate data structures, and efficient algorithms can ameliorate the costs.

This paper incorporates material previously presented in a poster [4] and a workshop [5]. Our contributions of this paper are:

- We propose cell-centric indexing as an innovative approach to an information retrieval system. A cell-centric index enables a user to find data without having to know the pre-existing structure of each table;
- We describe the mechanisms of one implementation of a cell-centric dataset search engine. We describe the structure and method of data storage and querying of our server; and,
- We describe a novel prototype interface that leverages cell-centric indexing in order to give users summaries of a dataset repository in terms of titles, content, and column names. The user can filter on any of these facets to generate more specific summaries.

The rest of the paper is organized as follows: we first discuss related work, briefly describe the idea of cell-centric indexing and its advantages and disadvantages, introduce the structure of our server and the methodology involved in querying, and finally describe a prototype interface.

2. Related Work

Scholars have investigated exploratory search to help searchers succeed in an unfamiliar area by proposing novel information retrieval algorithms and systems; some of them propose innovative user interfaces, while others try to predict the user's information need and to use the prediction to better facilitate the subsequent interaction. Chapman et al. [6] have reviewed different approaches to dataset search. Google's dataset search [7] is an example of a traditional approach to indexing web datasets: the system crawls the Web and indexes dataset that have metadata expressed in the schema.org (or a related) format. The only required properties are name and description.

Derthick et al. [8] describe a visual query language that dynamically links queries and visualizations. The system helps a user to locate information in a multi-object database, illustrates the complicated relationship between attributes of multiple objects, and assists the user to clearly express their information retrieval needs in their queries. Similarly, Yogev et al. [9] demonstrate an exploratory search approach for entity-relationship data, combining an expressive query language, exploratory search, and entity-relationship graph navigation. Their work enables people with little to no query language expertise to query rich entity-relationship data.

In the domain of web search, Koh et al. [10] devise a user interface that supports creativity in education and research. The model allows users to send their query to their desired commercial search engine or social platform in iterations. As the system goes through each iteration, it will combine the text and image results into a composition space. Addressing a similar problem, Bozzon et al. [11] design an interactive user interface that employs exploratory search and Yahoo! Query Language (YQL) to empower users to iteratively investigate results across multiple sources.

A tag cloud is a common and useful visualization of data that represents relative importance or frequency via size. Some researchers have adapted this idea to visualize query results. Fan et al. [12] focus on designing an interactive user interface with image clouds. The interface enables users to comprehend their latent query intentions and direct the system to form their personalized image recommendations. Dunaiski et al. [13] design and evaluate a search engine that incorporates exploratory search to ease researchers' scouting for academic publications and citation data. Its user interface unites concept lattices and tag clouds to present the query result in a readable composition to promote further exploratory search. On the other hand, Zhang et al. focus their work on knowledge graph data [14]. They combine faceted browsing with contextual tag clouds to create a system that allows users to rapidly explore graphs with billions of edges by visualizing conditional dependencies between selected classes and other data. Although they don't use tag clouds, Singh et al. [15] also display conditional dependencies in their data outline tool. For a given pivot attribute and set of automatically determined compare attributes, they show conditional values, grouped into clusters of interaction units.

Other scholars have investigated query languages and models. Ianina et al. [1] concentrate on developing an exploratory search system that facilitates the user having a way to conduct long text queries, while minimizing the risk of returning empty results, since the iterative "query–browse–refine" process [16] may be time-consuming and require expertise. Meanwhile, Ferré and Hermann [17] focus more on the query language, LISQL,

and they offer a search system that integrates LISQL and faceted search. The system helps users to build complex queries and enlightens users about their position in the data navigation process.

Yet another approach is to predict the user's search intent so that better search results can be presented. Peltonen et al. [18] utilize negative relevance feedback in an interactive intent model to direct the search. Negative relevance feedback predicts the most relevant keywords, which are later arranged in a radar graph where the center denotes the user, to represent the user's intent. Likewise, Ruotsalo et al. [19] propose a similar intent radar model that predicts a user's next query in an interactive loop. The model uses reinforcement learning to control the exploration and exploitation of the results.

3. Cell-Centric Indexing

We define a table as $T = \langle l, H, V \rangle$ where l is the label of the table, $H = \langle h_1, h_2, ..., h_n \rangle$ is a list of the column headers, and V is an $m \times n$ matrix of the values contained in the table. $V_{i,j}$ refers to the value in the i-th row and the j-th column, which has the heading h_j. We note that this model can be easily extended to include other metadata, as appropriate.

A naïve approach to indexing a collection of datasets would be to simply treat each table as a document, and have separate fields for the label, column headings, and (possibly) values. When terms are used consistently and the user is familiar with the terminology, this may work well. However, this approach has several weaknesses:

- Any query on values has lost context of what column the value appears in and what identifying information might be present elsewhere in the same row. For example, a table that contains capitals like (Paris, France) and (Austin, Texas) is unlikely to be relevant to a query about "Paris Texas" but would otherwise match.
- It is difficult to determine which new terms can be used to refine the query. Users would need to download some of the datasets and choose distinctive terms from the most relevant ones.
- A user's constraint could be represented in different tables in very different ways. If the user is looking for "California Housing Prices", there may be a table with some variant of that name, there may be a "Real Estate Prices" table with rows specific to California, or there may be a "Housing Prices" table that has a column for each state, including California. A user should be able to explore the collection to see how the data is organized and what terminology is used.

We have proposed cell-centric indexing as a novel way to address the problems above. Rather than treating the table as the indexed object, each datum (cell in the table) is an indexed object. In its simplest form, we propose four fields: *content*: the value of the cell, *title*: the label of the dataset the cell appears in, *columnName*: the header of the column the cell appears in, and *rowContext*: the values in all cells in the same row as the indexed cell. Formally, a cell value $V_{i,j}$ from table $T = \langle l, H, V \rangle$ can be indexed with: *content* $= V_{i,j}$, *title* $= l$, *columnName* $= h_j$, and *rowcontext* $= \bigcup_{k=1}^{n} V_{i,k}$. This index would allow users to find all cells that have a column header token in common regardless of dataset, or all cells that appear in the same row as some identifying token, or look for the occurrence of specific values in specific columns.

However, in this form, users still need to know which keywords to use and which fields to use them in. A cell-centric index alone is not helpful to a user who is not already familiar with the collection of datasets. In order to support the user in exploring the data, we propose the abstraction *conditional frequency vectors* (CFVs). Let I be a set of items, D be a set of descriptors (e.g., tags that describe the items), and $F \subseteq I \times D$ be a set of item and descriptor pairs $\langle x_i, d_i \rangle$. Let Q be a query, where $Q(F) \subseteq F$ represents the pairs for only those items that match Q. Then a CFV for Q and F is a set of descriptor-frequency pairs where the frequency is the number of times that the corresponding descriptor occurs within $Q(F)$: $\{\langle d, f \rangle \mid f = \#\{\langle x, d \rangle | \langle x, d \rangle \in Q(F)\}\}$. For cell-centric indexing, the items I are the set of all cells regardless of source dataset, and F_i pairs cells with terms from the i-th field. For example, if a cell c_5 was in a column titled "Real Estate Price," then $F_{columnName}$ includes the pairs $\langle c_5, real \rangle$, $\langle c_5, estate \rangle$, and $\langle c_5, price \rangle$. Typically, we sort the CFV in terms of descending frequency.

4. System Architecture

The architecture of the system is depicted in Figure 1. At the core of our system is an Elasticsearch server. Elasticsearch [20] is a scalable, distributed search engine that also supports complex analytics. Our system has two main functions: 1) parse collections of datasets, map them into the fields of a cell-centric index, and send indexing requests to Elasticsearch; and, 2) given a user query, issue a series of queries to Elasticsearch and construct histograms (CFVs) for each field. The Query Processor translates our high-level query API into specific ElasticSearch queries, and assembles the results into CFVs.

4.1. Index Definition

In Elasticsearch, a mapping defines how a document will be indexed: what fields will be used and how they will be processed. In cell-centric indexing **the cell is the**

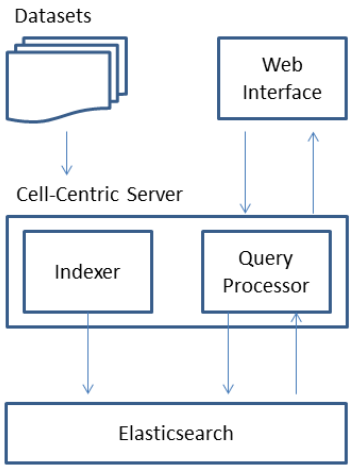

Figure 1: High-level system architecture

document, and our index must have fields that describe cells. Our mapping is summarized in Table 1. In addition to the four fields mentioned in Section 3, we have fields for the *fullTitle* (used to identify which specific datasets match the query) and metadata such as tags, notes, organization, and setId. The setId allows us to distinguish between different datasets with the same title, and to get an accurate count of how many datasets match a query. Note, that content is divided into two fields: *content* and *contentNumeric*, for reasons that will be described below. For each field, we give its type and, if applicable, the analyzer used to process text from the field.

We use three types of fields: *text*, *keyword*, and *double*. *Text* type fields are tokenized and processed by word analyzers, whereas *keyword* type fields are indexed as is (without tokenization or further processing). *Double* type fields are used to store 64-bit floating point numbers. Most of our fields are *text* fields, but contentNumeric is a *double* field, which allows it to store both integer and real numeric values, and both fullTitle and setid are *keyword* fields, since we want users to be able to view the complete name of the dataset in the result, and there is no need to parse setIds.

All *text* fields require an analyzer which determines how to tokenize the field and if any additional processing is required. We use two built-in Elasticsearch analyzers: the *stop* analyzer divides text at all non-letter characters and removes 33 stop words (such as “a”, “the”, “to”, etc.). For most fields, we use the *stop* analyzer, but we use the *wordDelimiter* analyzer for the *colunnName* field. In addition to dividing text at all non-letter characters, it also divides text at letter case transitions (e.g., “birthDate” is tokenized to “birth” and “date”). This analyzer does not remove stop words.

4.2. Indexing a Dataset

The system loads each dataset using the following process:

1. Read the metadata, which can include title, tags, notes and organization. If the original table is formatted as CSV, then this data might be contained in a separate file in the same directory, or as a row in a repository index file. If the table is formatted using JSON, the metadata may be specified along with the content, and there may be many datasets described in a single file.
2. Read the column headings $\langle h_1, h_2, ..., h_n \rangle$
3. For each row in the dataset:
   a) Read the row values: $\langle v_1, v_2, ..., v_n \rangle$
   b) Create *rowContext* by concatenating the values in the row. Note, to avoid creating different large context strings for each value in the row, we create a single *rowContext*. This means that each value is also part of its own row context. This decision helps make the system more efficient. An additional efficiency consideration is that each value included in *rowContext* is truncated to the first 100 characters.
   c) Build an index request for each cell value v_i. If the content is numeric (integer or real), it will be indexed in the *contentNumeric* field; otherwise it is indexed in the *content* field. The *columnName* field will be indexed with the corresponding header h_i. The title is indexed twice, once as a tokenized field that can be used in queries, and again as a keyword field that preserves the order of the title and can be used to precisely identify the dataset the cell originated from. All other metadata fields are indexed in a straight-forward way.

Field	Type	Analyzer
columnName	text	wordDelimiter
content	text	stop
contentNumeric	double	N/A
rowContext	text	stop
title	text	stop
fullTitle	keyword	N/A
tags	text	stop
notes	text	stop
organization	text	stop
setid	keyword	N/A

Table 1
Elasticsearch mappings used to implement cell-centric indexing

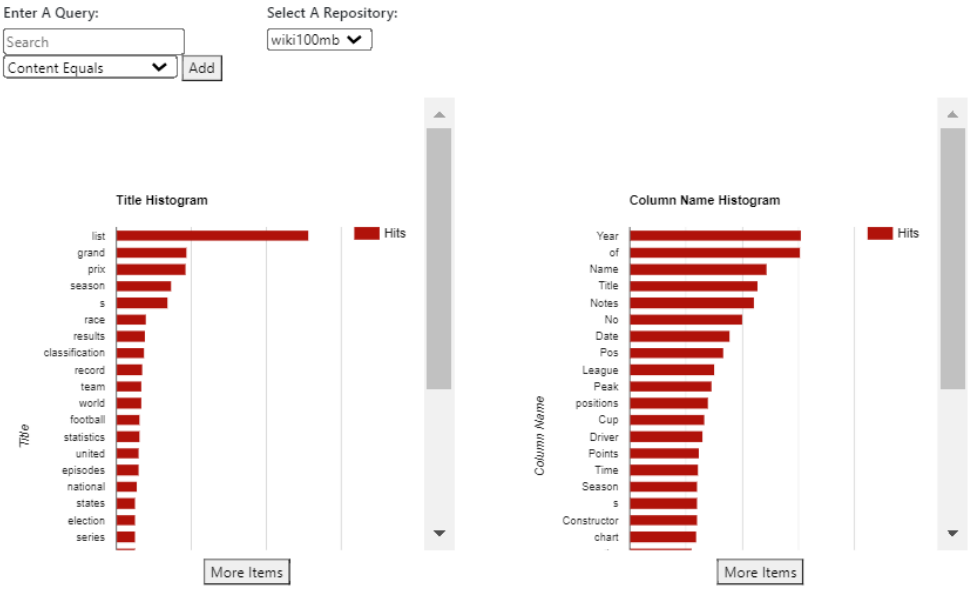

Figure 2: Initial pre-query histograms

d) For efficiency, index requests are batched and sent to the server in bulk. Synchronization is disabled in ElasticSearch during bulk loading to avoid excessive delays.

4.3. Querying the index

Our Query Processor takes a conjunctive, fielded query and returns a histogram for each response field. The response fields are fields that contain information that helps the user understand the characteristics of cells that match the query. Currently, response fields are *title*, *columnName*, *content*, *rowContext*, and *fullTitle*. Given a query q, the query process is:

1. Issue query q requesting *term aggregations* for title, columnName, content, rowContext and fullTitle, Term aggregations are a feature of Elasticsearch that return a list of terms that appear in the selected documents, along with their frequency, i.e., CFV's for q.
2. Calculate the min a and max b for matching contentNumeric data.
3. Select a representative set N of matching numeric content by issuing a *percentile* query against the *contentNumeric* field that excludes the top and bottom 5 percent of the data.
4. Calculate the mean μ and standard deviation σ of set N.
5. If $a < \mu - 1.5\sigma$ and $b > \mu + 1.5\sigma$ (i.e., numeric data is not particularly skewed), build an aggregation query for contentNumeric data using ranges calculated from μ and σ: the lowest range is $[a, \mu - 1.5\sigma]$ and the highest is $[\mu + 1.5\sigma, b]$, where there are 3 intermediate ranges of uniform size with the middle range $\mu - 0.5\sigma, \mu + 0.5\sigma]$. If data are skewed, the ranges are shifted appropriately.
6. Issue an Elasticsearch *histogram aggregation* query with the calculated ranges. Treat each range as a content term, and insert these terms and their frequencies into the the *content* CFV.
7. Return CFVs for each response field.

Much of the processing above allows the system to dynamically determine buckets for numeric content that provide a useful picture of its distribution. Unlike textual terms, numeric terms exhibit greater variability. Histograms built using distinct numeric strings are unlikely to have significant value. For example, "135", "135.0" and "1.35E+2" are all equivalent, while many users might consider "135.0001" to be close enough. To address this, we create ranges over numeric values. Our approach computes the mean and standard deviation over the middle 90% of data, thus removing the influence of outliers, and then specifies the buckets to have a width of one standard deviation with one bucket centered over the mean. Once the histogram of numeric ranges is created, its data is merged with the content histogram to produce a single histogram that shows frequencies of textual terms and

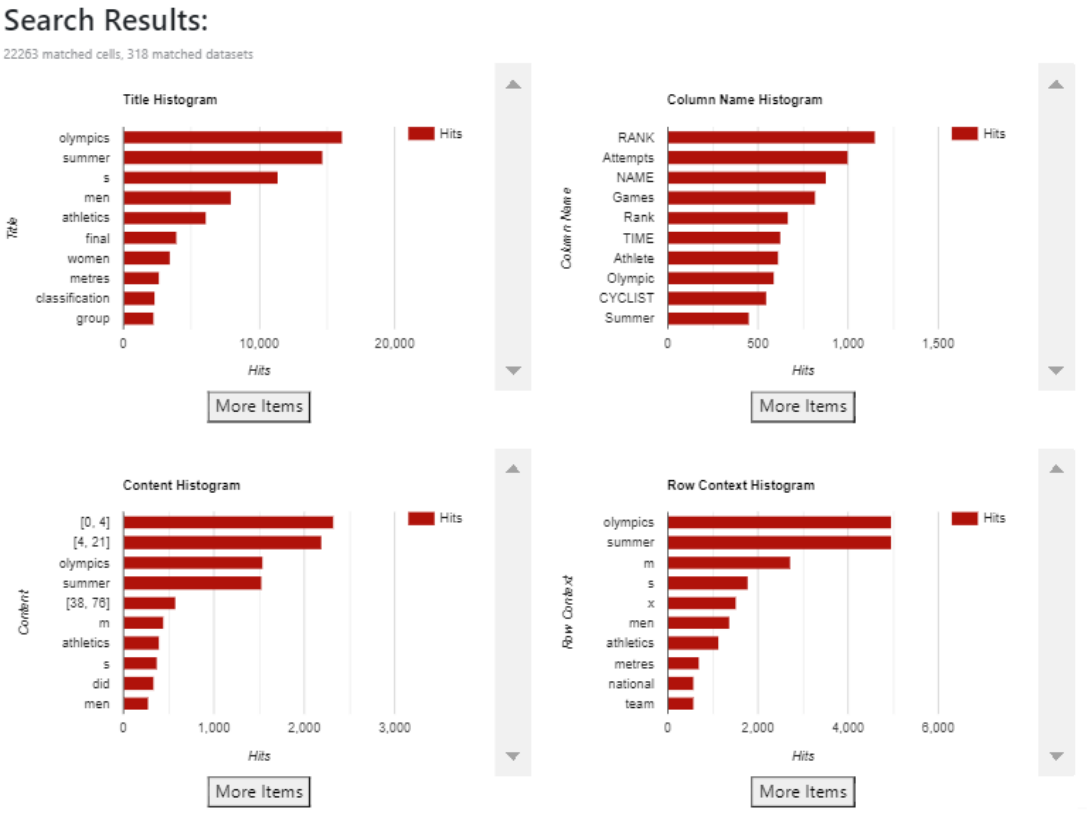

Figure 3: Search results with query: title=olympics

numbers within ranges that depend both on the dataset and the query.

5. Prototype User Interface

An example of a typical use case is demonstrated using Figures 2-4. In this specific case, the user wants to find a dataset containing data on Kenya's performance in the 2004 Athens Olympics. Initially, the user is presented with the graphs in Fig. 2. These histograms show the most frequent title and column terms in the collection of indexed datasets. However, the example histograms do not initially show anything regarding the Olympics. By using the "More Items" button at the bottom of the title histogram the user can find the term Olympics and add it directly to their query. After this term is added the screen changes to that shown in Fig. 3. The user can now look through all 4 histograms and decide which term best helps them get to their desired data. Once again using the "More Items" button, the term Athens can be found in the content histogram. Once this is added the user might direct themselves to the full title histogram shown in Fig. 4. There the user can find a dataset titled "Kenya at the Olympics Medalists". To gain access to the dataset the user must add the full title to their query. Once a full title is in the query a button appears that performs a Google query of the full title. Since this specific dataset is from WikiTables, the Google query will provide a link to the Wikipedia page containing the table. We now discuss specific interface components in more detail.

5.1. Pre-query Histograms

Before any search parameters are set, the user is shown two pre-query histograms that return up to the 50 most frequent title and column name tokens within the current repository (see Fig. 2). Column name and title histograms provide a good overview and are vital in allowing the user to explore the datasets without prior knowledge of the contents. The pre-query histograms are presented to the user when there are no active queries, such as when

the page is initially loaded or when all queries have been deleted. Clicking on a histogram bar will automatically add the corresponding term to the query and generate the standard set of histograms.

5.2. Results Histograms

The standard screen displays the user's current query and five histograms. Each histogram is associated with a field, and tokens are sorted in descending frequency of co-occurrence with the query. The length of a bar indicates how many cells match the query. As with the pre-query histograms, clicking on a bar adds the associated term to the query, and generates a new result histogram. Below each histogram is an option to provide more results on the histogram. Initially, each histogram presents the top 10 results, however, the top 25 results are pre-fetched which allows the newly requested results to be automatically added to the histogram.

Due to the connection between the count of matched cells and bar length, there is the possibility that the first bar will be significantly larger than all remaining bars, making them difficult to see or select. To combat this, we compare the counts of the two most frequent results. If the first result contains 10 times more hits than the second most frequent we change the scale of the histograms to logarithmic, thus making it easier to visualize distinctions in skewed distributions.

Figure 3 shows the response of our prototype interface to the query with *title*="olympics". It displays a CFV for each field as a histogram; the longer the red bar, the more frequently that term co-occurs with the query. As we can see, 318 datasets contain matches, and after "olympics," the most common title word is "summer." The most frequently-occurring terms in the column names of matching cells are "RANK" and "attempts". The content histogram combines terms with numeric ranges. In particular, the first, second, and fifth rows were all inserted by the numeric range processing (as described previously in Sect. 4.3). For this query, there are many cells with values between 0 and 4, and slightly fewer with values between 4 and 21. The next most common content values are the terms "olympics" and "summer." Note, the figure does not show the histogram for full titles that corresponds to this query (but is still part of the prototype interface). As discussed in the next paragraph, this histogram indicates how many matching cells are in each dataset.

The user can refine their query and create new histograms by clicking on any terms in the result. For example, if the user clicks on "athens" in the content histogram (after scrolling down), the system will display a new set of histograms summarizing the datasets that have "athens" as a content field and "olympics" in the title; in other words the query will be *title*="olympics" and

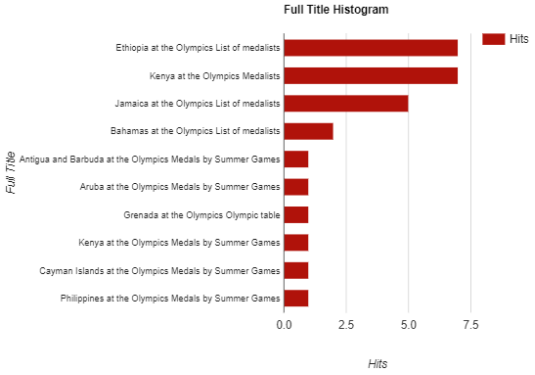

Figure 4: Results of Full Title Histogram with query: title=olympics, content=athens

content="athens". For this refinement, we show the Full Title histogram (see Fig. 4). In this histogram, the bars represent the number of cells in a data set that match the user's query. The user can add this bar to the query to get specific information about the distribution of terms in the chosen dataset. Additionally, this enables the option to search for the dataset, which is accomplished using a Google query of the dataset's full title.[1] The user can continue to explore the dataset collection by adding terms to and removing terms from the query.

6. Conclusion

We have proposed cell-centric indexing as an innovative approach to information retrieval of tabular datasets. Such indices support richer queries about tables that do not require the user to know the pre-existing structure of each table. They also provide the potential for new exploratory interfaces, and we describe one that gives users summaries of a dataset repository in terms of titles, content, and column names. The user can filter on any of these facets to generate more specific summaries. Future work will test the effectiveness of this novel approach in facilitating dataset searches especially amongst non-expert users.

Acknowledgments

This material is based upon work supported by the National Science Foundation under Grant No. III-1816325. Lixuan Qiu and Drake Johnson contributed to early drafts of this paper. We thank Alex Johnson, dePaul Miller, Keith Register, and Xuewei Wang for contributions to the system implementation.

[1] Many of our dataset collections do not have a URL recorded, which is why we do not simply link to the dataset as a result.

References

[1] A. Ianina, L. Golitsyn, K. Vorontsov, Multi-objective topic modeling for exploratory search in tech news, in: A. Filchenkov, L. Pivovarova, J. Žižka (Eds.), Artificial Intelligence and Natural Language, Springer, 2017, pp. 181–193. Communications in Computer and Information Science, vol 789.

[2] H. Borchart, Effects of content preview on query refinement in dataset search, Senior Project Report, Cognitive Science Program, Lehigh University, Bethlethem, PA, 2021.

[3] L. Miller, Facilitating dataset search of non-expert users through heuristic and systematic information processing, Honors Thesis, Cognitive Science Program, Lehigh University, Bethlethem, PA, 2020.

[4] D. Johnson, K. Register, B. D. Davison, J. Heflin, An exploratory interface for dataset repositories using cell-centric indexing, in: Proceedings of the 2020 IEEE International Conference on Big Data (IEEE BigData 2020), 2020, pp. 5716–5718. Poster paper.

[5] L. Qiu, H. Jia, B. D. Davison, J. Heflin, An architecture for cell-centric indexing of datasets, in: Proceedings of PROFILES'20: 7th International Workshop on Dataset PROFILing and Search, 2020, pp. 82–96. Held with ISWC 2020.

[6] A. Chapman, E. Simperl, L. Koesten, G. Konstantinidis, L.-D. Ibáñez, E. Kacprzak, P. Groth, Dataset search: a survey, The VLDB Journal 29 (2020) 251–272.

[7] N. Noy, M. Burgess, D. Brickley, Google dataset search: Building a search engine for datasets in an open Web ecosystem, in: Proceedings of The Web Conference, 2019, pp. 1365–1375.

[8] M. Derthick, J. Kolojejchick, S. F. Roth, An interactive visual query environment for exploring data, in: Proceedings of the 10th Annual ACM Symposium on User Interface Software and Technology, UIST '97, Association for Computing Machinery, New York, NY, USA, 1997, p. 189–198. doi:10.1145/263407.263545.

[9] S. Yogev, H. Roitman, D. Carmel, N. Zwerdling, Towards expressive exploratory search over entity-relationship data, in: Proceedings of the 21st International Conference on World Wide Web, WWW '12 Companion, Association for Computing Machinery, New York, NY, USA, 2012, p. 83–92. doi:10.1145/2187980.2187990.

[10] E. Koh, A. Kerne, R. Hill, Creativity support: Information discovery and exploratory search, in: Proceedings of the 30th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR '07, Association for Computing Machinery, New York, NY, USA, 2007, p. 895–896. doi:10.1145/1277741.1277963.

[11] A. Bozzon, M. Brambilla, S. Ceri, P. Fraternali, Liquid query: Multi-domain exploratory search on the web, in: Proceedings of the 19th International Conference on World Wide Web, WWW '10, Association for Computing Machinery, New York, NY, USA, 2010, p. 161–170. doi:10.1145/1772690.1772708.

[12] J. Fan, D. A. Keim, Y. Gao, H. Luo, Z. Li, Justclick: Personalized image recommendation via exploratory search from large-scale flickr images, IEEE Transactions on Circuits and Systems for Video Technology 19 (2008) 273–288.

[13] M. Dunaiski, G. J. Greene, B. Fischer, Exploratory search of academic publication and citation data using interactive tag cloud visualizations, Scientometrics 110 (2017) 1539–1571.

[14] X. Zhang, D. Song, S. Priya, J. Heflin, Infrastructure for efficient exploration of large scale linked data via contextual tag clouds, in: International Semantic Web Conference, Springer, 2013, pp. 687–702.

[15] M. Singh, M. J. Cafarella, H. V. Jagadish, Dbexplorer: Exploratory search in databases, in: E. Pitoura, S. Maabout, G. Koutrika, A. Marian, L. Tanca, I. Manolescu, K. Stefanidis (Eds.), Proceedings of the 19th International Conference on Extending Database Technology, EDBT 2016, Bordeaux, France, March 15-16, 2016, OpenProceedings.org, 2016, pp. 89–100. doi:10.5441/002/edbt.2016.11.

[16] R. W. White, R. A. Roth, Exploratory Search: Beyond the Query-Response Paradigm, Synthesis Lectures on Information Concepts, Retrieval, and Services, Morgan & Claypool Publishers, 2009. doi:10.2200/S00174ED1V01Y200901ICR003.

[17] S. Ferré, A. Hermann, Semantic search: Reconciling expressive querying and exploratory search, in: International Semantic Web Conference, Springer, 2011, pp. 177–192.

[18] J. Peltonen, J. Strahl, P. Floréen, Negative relevance feedback for exploratory search with visual interactive intent modeling, in: Proceedings of the 22nd International Conference on Intelligent User Interfaces, IUI '17, Association for Computing Machinery, New York, NY, USA, 2017, p. 149–159. doi:10.1145/3025171.3025222.

[19] T. Ruotsalo, J. Peltonen, M. J. A. Eugster, D. Głowacka, P. Floréen, P. Myllymäki, G. Jacucci, S. Kaski, Interactive intent modeling for exploratory search, ACM Trans. Inf. Syst. 36 (2018). doi:10.1145/3231593.

[20] C. Gormley, Z. Tong, Elasticsearch: the definitive guide: a distributed real-time search and analytics engine, O'Reilly Media, Inc., 2015.