Bulletin of the Technical Committee on

Data Engineering

September 2016 Vol. 39 No. 3 IEEE Computer Society

Letters

Letter from the Editor-in-Chief . . . . . *David Lomet* 1
Letter from the 2016 IEEE TCDE Impact Award Winner . . . . . *Michael Carey* 2
Call for Nominations for TCDE Chair . . . . . *Kyu-Young Whang* 3
Letter from the Special Issue Editor . . . . . *Haixun Wang* 4

Special Issue on Text, Knowledge, and Database

Managing Google's data lake: an overview of the Goods system . . . . . *Alon Halevy, Flip Korn, Natalya F. Noy, Christopher Olston, Neoklis Polyzotis, Sudip Roy, Steven Euijong Whang* 5
Data services leveraging Bing's data assets . . . . . *Kaushik Chakrabarti, Surajit Chaudhuri, Zhimin Chen, Kris Ganjam, Yeye He* 15
Attributed Community Analysis: Global and Ego-centric Views . . . . . *Xin Huang, Hong Cheng, Jeffrey Xu Yu* 29
Ten Years of Knowledge Harvesting: Lessons and Challenges . . . . . *Gerhard Weikum, Johannes Hoffart, Fabian Suchanek* 41
Towards a game-theoretic framework for text data retrieval . . . . . *ChengXiang Zhai* 51
Neural enquirer: learning to query tables in natural language . . . . . *Zhengdong Lu, Hang Li, Ben Kao* 63
Multi-Dimensional, Phrase-Based Summarization in Text Cubes . . . . . *Fangbo Tao, Honglei Zhuang, Chi Wang Yu, Qi Wang, Taylor Cassidy, Lance Kaplan, Clare Voss, Jiawei Han* 74
Answering End-User Questions, Queries and Searches on Wikipedia and its History . . . . . *Maurizio Atzori, Shi Gao, Giuseppe M. Mazzeo, Carlo Zaniolo* 85

Conference and Journal Notices

TCDE Membership Form . . . . . back cover

The TC on Data Engineering

Membership in the TC on Data Engineering is open to all current members of the IEEE Computer Society who are interested in database systems. The TCDE web page is http://tab.computer.org/tcde/index.html.

The Data Engineering Bulletin

The Bulletin of the Technical Committee on Data Engineering is published quarterly and is distributed to all TC members. Its scope includes the design, implementation, modelling, theory and application of database systems and their technology.

Letters, conference information, and news should be sent to the Editor-in-Chief. Papers for each issue are solicited by and should be sent to the Associate Editor responsible for the issue.

Opinions expressed in contributions are those of the authors and do not necessarily reflect the positions of the TC on Data Engineering, the IEEE Computer Society, or the authors' organizations.

The Data Engineering Bulletin web site is at http://tab.computer.org/tcde/bull_about.html.

Letter from the Editor-in-Chief

The Current Issue

Many years ago, there was an IBM ad with the catch phrase "Not just data, reality". ADspeak is not science, so we should not take this phrase as describing the historical situation when it was published, nor should we look back at it with too critical an eye. (Though it is funny.) I think of it as aspirational, and in that context, it is really right-on!

It is one thing to have petabytes of data, it is another to be able to exploit it to some purpose. Gathering data is easy, though it can be expensive. Extracting utility from data requires more than expensive infrastructure. It requires insights to build technology to "mine" the data. The data engineering community has been working on this for many years for multiple and increasing purposes, from knowledge research to health care to business opportunity, and many places in-between.

The current issue surveys the efforts by our community to put data to good use. And, perhaps unique to the Bulletin, it spans historical to work in progress, from universities, research centers, and industry. Thus, the issue provides a broad perspective on the purposes to which data is being put and the technology to enable this. Haixun Wang, the issue editor, has succeeded in bringing this topic to Bulletin readers. The topic has, is, and will continue to be timely. I want to thank Haixun for his hard work in bringing this important issue to us.

"No" on Proposed IEEE Constitutional Amendment

The Computer Society is part of the IEEE. As such it has a role to play in the governance of the IEEE, as do all societies within the IEEE. A proposed amendment to the IEEE constitution would change the way that the IEEE is governed. As of this moment, the Computer Society is joined with nine other societies who have had formal votes in opposition to the amendment. And no society has voted in favor of the amendment.

The thrust of the IEEE amendment is to move governance partially out of the constitution and make it subject to the more easily changed by-laws. Prior versions of the amendment have suggested that the purpose may be to reduce the role of societies in IEEE governance. It also removes IEEE members from some governance votes that are currently required. The proposal in detail can be seen at
`https://www.ieee.org/documents/constitution_approved_amendment_changes_election.pdf`
(login is required using your IEEE account).

As a member of the Computer Society Board of Governors (BOG), I have participated in BOG discussions on the proposed IEEE amendment, and agree with the position taken by the Computer Society in opposition. I believe the amendment is misguided. Professional meetings can be contentious, as there are sometimes conflicting interests that need to be resolved. But the change proposed seems to me to not improve the way we deal with these conflicts but rather seeks to bury them. That is not the way democracy is supposed to work. So I urge you to vote "no" on this amendment.

IEEE TCDE Award Winners

The Technical Committee on Data Engineering (TCDE) initiated awards for the data engineering community in 2014. This is the third year for the awards. The winners of the awards this year are listed on the TCDE web site
`http://tab.computer.org/tcde/tcdeawardsrecipients.html`

This year, award winners have the opportunity to publish a short communication about their thoughts as a result of the award in the Bulletin. The current issue contains a short communication from Impact Award winner Michael Carey. Award winners have made truly distinguished contributions. I want to congratulate them on their awards, and I would encourage you to see what Mike has to say in the current issue.

David Lomet
Microsoft Corporation

Letter from the 2016 IEEE TCDE Impact Award Winner

In March I received the following news by e-mail: "...on behalf of the IEEE TCDE I am delighted to inform you that the TCDE Award Committee has awarded you the IEEE TCDE CSEE (Computer Science, Engineering, and Education) and Impact Award for this year 'for leadership and research excellence in building impactful data management systems, engineering tools, products, and practices'." I was very surprised, and I was extremely honored to be chosen! Along with that news came an opportunity to write a one-pager about "anything" for the DE Bulletin, and I've decided to use my DE space to offer some in-my-opinion (IMO) "stylistic suggestions" for researchers in our community – thoughts aimed at all of us, but especially at the younger set.

1. **Results, not papers, are the objective!** All too often, conversations with our fellow academics (both faculty and students alike) or research lab staffers include phrases like "I'm working on a paper about ..." or "I want to write a paper for SIG*XXX*...". The goal of our work should be to "do cool stuff" – to build cool systems or subsystems, or to come up with cool and useful algorithms or data structures or insights – not to write papers. Once we have something to report, it's paper time – but the paper should never be the end goal (IMO).

2. **It's possible to over-publish – but please don't!** In the "good old days", a fresh Ph.D. student would have a small handful of papers on their CV – maybe one per year spent in graduate school – and that was sufficient as long as the results were cool (see point 1). It's really not possible for one person to come up with a new publication-worthy result in under 6-12 months per result. Look at some of the most impactful systems researchers in CS – e.g., let's look at two of my heroes, Barbara Liskov (MIT) and John Ousterhout (currently at Stanford). Both are "repeat offenders" at doing terrific and impactful work on topics like operating systems, file systems, programming languages, CAD tools, storage systems, ... Each has built systems that have had truly lasting impact on our field. But check out their publication records in DBLP...! During a number of their years, each published just a small handful of papers with their students. If we just counted papers, neither would have gotten hired or been tenured – it's their cool stuff and their impact that have mattered over the years. We should become suspicious and skeptical about quality and motivation when we see a CV with more than a few papers per year – it's pretty unlikely that those are all papers that really deserved to be written (IMO).

3. **You won't really know unless you try!** Somewhat sadly, our systems conferences often include some papers that – if you really tried to use the ideas – just wouldn't work. If you work on a problem in isolation, it's dangerously easy to develop something that sounds good on paper, simulates or works well in isolation, but would fall on its face in a real system due to incompleteness or aspects that were overlooked (possibly on purpose, more likely by accident). Great, so you have a really cool new search structure! But can you build a big instance of it fast enough? Can it be updated? Made multiuser and recoverable? Is the information needed to build and maintain it available in an actual system setting? Are its APIs compatible with how real systems are structured? Is the portion of the end-to-end path that it improves on really where the key bottlenecks are? (Being 10x faster on 2% of a query's execution time sounds impressive if you don't say the 2% part too loudly.) The ideal approach (IMO) to contributing a cool new result in systems is what I like to call the "BMW" approach: Build, Measure, Write – in that order. Come up with your idea(s), or examine ideas from others, and then start by building them completely – preferably in some actual system setting. Then measure their behavior and understand what you see – figure out why you're seeing what you're seeing. When doing experiments, first use workloads used by others – yes, their same workloads – before you show how your ideas do on your own favorite use cases. (That's how it's done in the real sciences – and CS should do this too.) Finally, once you're done, you'll have a well-understood and cool new result (see point 1) – so *now* you should write it up.

Please give some consideration to the points above. Our field doesn't suffer from a lack of papers, so let's see if we can reset our research culture a bit and move it back in the direction of the "good old days". Thanks for reading – and thanks to the TCDE community for somehow picking me as its 2016 CSEE Awardee! While it wasn't actually deserved (IMO), I'm certainly honored and appreciative.

Michael Carey
U.C. Irvine

Call for Nominations for TCDE Chair

The Chair of the IEEE Computer Society Technical Committee on Data Engineering (TCDE) is elected for a two-year period. The mandate of the current Chair, Xiaofang Zhou, expires at the end of this year and the process of electing a Chair for the period 2017-2018 has begun. A Nominating Committee chaired by Kyu-Young Whang has been struck. The Nominating Committee invites nominations for the position of Chair from all members of the TCDE. To submit a nomination, please contact Kyu-Young Whang (kywhang@mozart.kaist.ac.kr) before Oct. 30th, 2016.

More information about TCDE can be found at `http://tab.computer.org/tcde/`.

Kyu-Young Whang
KAIST

Letter from the Special Issue Editor

The field of data management has gone a long way since the age of relational databases where data is well defined, managed by a centralized system, and accessed through methods based on well-founded semantics. In today's data management world, we are facing two major challenges: the diversity and the heterogeneity of the data, and the ad-hoc methods for accessing and manipulating such data. Even within a single organization, data could be so varied and applications so diverse that no centralized system is able to cover all needs of data management. In the public domain, challenges are often more broad and complex, with data ranging from Wikipedia to human genome and applications ranging from question answering to disease prediction.

In this issue, we survey several typical scenarios of data engineering in the post relational databases age. The data we look into include enterprise data, social network data, Wikipeida data, etc., and the applications we cover range from simply cataloging the data to building natural language interfaces for the data. While there is not going to be a one-size-fits-all solution to the many challenges that come with the heterogeneity of the data, we argue there are still several priorities to follow in the new age of data engineering. First, cataloging heterogeneous datasets to make them readily available to the user is the first requirement of data management. Second, data is characterised by richer and denser relationships they exhibit, and it is such relationships that drive many applications. Third, effective retrieval is essential for data to provide value. While we may not have a set of operators defined on well-founded semantics that can serve any need, investigating novel retrieval methods is important. Finally, data management is no longer just a system issue. Modeling and understanding the semantics of the data (particularly text data) holds the key to a new generation of intelligent applications.

We start with creating awareness for enterprise data. Enterprise data has seen such an explosive growth in recent years that most enterprise users are not necessarily aware of the availability or the semantics of their own data. Halevy et. al. discuss the situation at Google. They handle the scale and heterogeneity challenge of Google's internal data and introduce techniques to extract metadata from them, so as to facilitate the understanding and use of such data internally at Google. Microsoft's Kaushik et. al. focus on the massive amount of data amassed by web search engines. They describe services (e.g., synonymy service and web table service) created out of Bing's search data that benefit a variety of Microsoft's products including Office and Cortana.

Rich relationships inside graph-like data are the driving force of many applications, and such data posts great challenges to data management. Huang et. al. cast their attention on social networks. They survey several state-of-the-art community models based on dense subgraphs, and investigate social circles, which are a special kind of communities formed by friends in 1-hop neighborhood network for a particular user. Another important graph data is the knowledge graph. Weikum et. al. focus on advances the knowledge harvesting community has made in turning internet content, with its wealth of latent-value but noisy text and data sources, into crisp "machine knowledge" that can power intelligent applications.

New data calls for novel retrieval methods. Zhai introduces a game-theoretic formulation of the text retrieval problem to optimize user experience in search The key idea is to model text retrieval as a process of a search engine and a user playing a cooperative game, with a shared goal of satisfying the users information need (or more generally helping the user complete a task) while minimizing the users effort and the operation cost of the retrieval system. Lu et. al. propose a neural network architecture for answering natural language questions against databases. It achieves this by finding distributed representations of queries and knowledge base tables.

As we see in both Weikum et. al.'s work on knowledge harvesting and Lu et. al.'s work on neural natural language QA, the biggest challenge in data engineering is no longer just a system challenge, instead, how to model and understand the data is often the key to the success of applications. Tao et. al. study the problem of *phrase-based summarization* of a set of documents of interest. The authors introduce a phrase ranking measure to leverage the relation between subsets of documents. Atzori et. al. describe a smart system that allows people enter natural language questions and then translates them into SPARQL queries executed on DBpedia.

Haixun Wang
Facebook Inc.

Managing Google's data lake: an overview of the GOODS system

Alon Halevy[2*], Flip Korn[1], Natalya F. Noy[1], Christopher Olston[1], Neoklis Polyzotis[1], Sudip Roy[1], Steven Euijong Whang[1]

[1]Google Research

[2]Recruit Institute of Technology

alon@recruit.ai,{flip,noy,olston,npolyzotis,sudipr,swhang}@google.com

Abstract

For most large enterprises today, data constitutes their core asset, along with code and infrastructure. For most enterprises, the amount of data that they produce internally has exploded in recent years. At the same time, in many cases, engineers and data scientists do not use centralized data-management systems and end up creating what became known as a data lake*—a collection of datasets that often are not well organized or not organized at all and where one needs to "fish" for useful datasets. In this paper, we describe our experience building and deploying* GOODS*, a system to manage Google's internal data lake.* GOODS *crawls Google's infrastructure and builds a catalog of discovered datasets, including structured files, databases, spreadsheets, and even services that provide access to the data.* GOODS *extracts metadata about datasets in a post-hoc way: engineers continue to generate and organize datasets in the same way that they have before, and* GOODS *provides value without disrupting teams' practices. The technical challenges that we had to address resulted both from the scale and heterogeneity of Google's data lake and from our decision to extract metadata in a post-hoc manner. We believe that many of the lessons that we learned are applicable to building large-scale enterprise-level data-management systems in general.*

1 Introduction

Most large enterprises today witness an explosion in the amount of data that they generate internally for use in ongoing research and development. The reason behind this explosion is simple: by allowing engineers and data scientists to consume and generate data in an unfettered manner, enterprises promote fast development cycles, experimentation, and, ultimately, innovation that drives their competitive edge. As a result, this internally generated data often becomes a prime asset of the company, on par with source code and internal infrastructure.

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

*Work done while at Google Research.

The flip side of this explosion is the creation of a so called *data lake* [1, 2, 6]: a growing volume of internal datasets, with little codified information about their purpose, value, or origin. This scarcity of information is problematic: data becomes siloed within the teams who carry the "tribal knowledge" of the data's origin, which, in turn, results in significant losses in productivity and opportunities, duplication of work, and mishandling of data.

In this paper we describe Google Dataset Search (GOODS), a system that we built and deployed in order to help Google's engineers organize and manage datasets in its data lake. GOODS operates in a *post-hoc* manner: it collects and aggregates metadata about datasets after the datasets were created, accessed, or updated by various pipelines. Put differently, teams and engineers continue to generate and access datasets using the tools of their choice, and GOODS works in the background, in a non-intrusive manner, to gather the metadata about datasets and their usage. GOODS then uses this metadata to power services that enable Google engineers to organize and find their datasets in a more principled manner. Hence, GOODS is very different from Enterprise Data Management (EDM) systems, which act as gateways and require dataset owners and consumers to use specific protocols and APIs for data access.

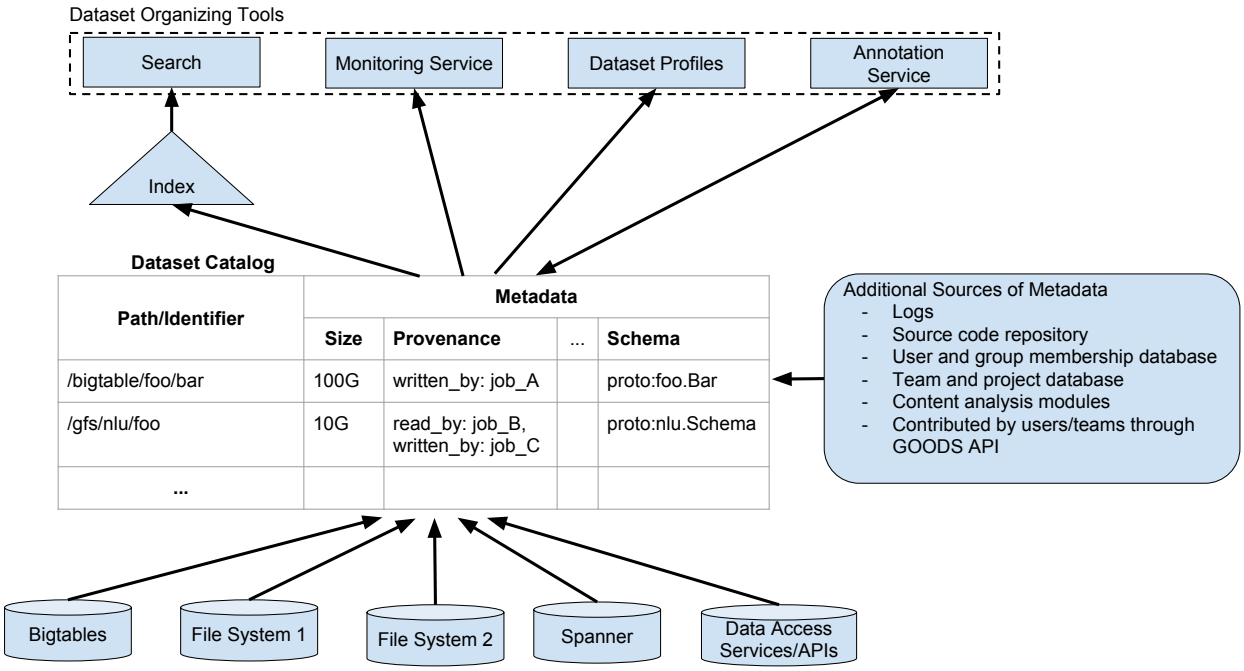


Figure 1: Overview of Google Dataset Search (GOODS). GOODS collects metadata about datasets from various storage systems. It infers metadata and relationships among datasets by processing additional sources such as logs and information about dataset owners and their projects, by analyzing content of the datasets, and by collecting input from the GOODS users. The collected metadata powers user-facing tools, such as search, dataset profiles, monitoring, and a dataset-annotation service.

Figure 1 shows a schematic overview of our system. GOODS continuously crawls different storage systems and the production infrastructure (e.g., logs from running pipelines) to discover which datasets exist and to gather metadata about each one (e.g., owners, time of access, content features, accesses by production pipelines). GOODS aggregates this metadata in a central catalog and correlates the metadata about a specific dataset with

information about other datasets. GOODS then uses this catalog to provide Google engineers with services for dataset management. These services include the following:

- A *search engine* over all the datasets in the company, with facets for narrowing search results, to help engineers and analysts find the most relevant datasets.
- A per-dataset *profile page* that renders the metadata that GOODS has recorded about a specific dataset and can thus help users understand the dataset and its relationships to other datasets in the company. The profile page also integrates with other tools that can further process the dataset, thus helping users act on the data.
- A *monitoring service* that allows teams to monitor features of the datasets that they own, such as size, distribution of values in the contents, or availability. Users can configure this monitoring service to issue alerts if the features change unexpectedly.
- A *annotation service* that allows dataset owners or trusted principals (e.g., data librarians, or a data-stewardship team) to extend a dataset's metadata with domain-specific annotations that can appear in the profile page. As an example, a dataset owner can provide a textual description for the dataset's contents or attach a visualization that can inform other users.

Search is the most frequently used service in GOODS, which demonstrates the importance of dataset discovery within a data lake. However, we have seen good adoption for the remaining services. We were also pleasantly surprised to see that teams used GOODS for scenarios that we did not originally anticipate:

- *Schema auditing* A team used search to identify datasets owned by other teams that conformed to a specific schema that this team owned. The team then audited the resulting datasets to ensure that the schema was used according to its specifications.
- *Data discovery through provenance* The GOODS catalog includes provenance metadata, in the form "dataset Y was read/written by production job X". This information, and in particular the transitive closure of the provenance links, can be useful in understanding the pedigree of a dataset and its downstream dependencies, and thus features prominently in the profile page of each dataset. A team found a different usage for these links and relied on them for dataset discovery. Specifically, a ML team wished to use datasets that were publicized as canonical for certain domains, but they found that these datasets were too "groomed" for ML. To overcome this problem, the team relied on the provenance links to browse upstream and identify the datasets that were used to derive the canonical datasets. These input datasets contained less groomed data and were more suitable for the specific ML task.
- *Content visualization* A technical-infrastructure team used the annotation service to attach a visualization to datasets that represent training data for ML pipelines. This visualization, which illustrates statistics on the features of the training examples, is surfaced on the dataset profile page in order to help users understand the distribution of features and also to spot anomalies that can affect the quality of machine-learned models.

The remainder of the article describes our experience with the development of GOODS. We begin by identifying the key challenges that we had to address in building the data-lake management system at Google—a data lake that contains more than 20 billion datasets. Many of the challenges that we addressed in GOODS were precipitated by the scale and characteristics of the data lake at Google, but we believe that our experience and the lessons that we learned will apply to similar systems in other enterprises. We then introduce the logical structure that we used to organize the data lake's metadata, using relationships among entities (datasets, teams, projects, and so on). We argue that this organization, which is inspired by structured knowledge bases and their

application to Web search, can help answer important questions about the contents of the data lake. We then review briefly some of the system-related and technical issues that we solved in developing GOODS, and finally we present a few directions for future work based on our experience with the current system deployment.

2 Challenges in Managing and Organizing an Enterprise Data Lake

We encountered many challenges as we designed and build GOODS. In this section, we highlight and elaborate on some of the important challenges. A more detailed list of challenges can be found in our previous work [5].

2.1 Organizing the Data Lake

As we discussed, a major goal of a data-lake management system is to gather metadata for the datasets in it. We can view the metadata for each dataset independently. However, we found that is far more powerful to consider how to integrate this metadata in order to uncover relationships among datasets. Identifying and inferring these relationships helped us address some of the scalability challenges that we just described. Furthermore, it helped us organize the datasets in the data lake and thus to provide our users with a better understanding of the data lake's contents.

As an example, consider the following query over the data lake: "Find all datasets that are derived from a dataset owned by project X." This query can help assess the impact of project X or identify teams that need to be notified if there is a plan to wipe the datasets owned by X. Answering this query requires knowledge of the composition of various teams, dataset ownership, and provenance relationships. Another interesting query is "Find all datasets written by a production job whose code uses a specific version of method X", which can help identify datasets that might have been affected by faulty code. Again, answering this query requires knowledge of several types of relationships, including provenance, the versions of binaries in production jobs, and the linkage of source code to binaries.

To support this type of functionality we first have to identify what types of relationships exist and which of them we can determine efficiently. All the challenges that we have identified already still remain. The large scale means that we cannot perform an exhaustive search to identify these relationships. For instance, while it would be useful to know which datasets, or parts of datasets, have identical content, we cannot compare every pair of datasets to each other. Or, because we are building GOODS in a post- hoc manner, we need to understand which infrastructure signals can help uncover these relationships. For instance, to identify provenance relationships between datasets we may join the dataset catalog with logs of different jobs that generate datasets. However, because these resources come from different teams, they often use different naming schemes, different levels of granularity for what they define as a dataset or what they include, and so on.

2.2 Scalability of Metadata Extraction

The first challenge that we had to address is the scalability of of gathering metadata for the datasets in the data lake. Three factors contribute to this challenge: the sheer number of datasets that GOODS manages, the daily churn of the datasets, and the cost of extracting metadata for each individual dataset. Our recent snapshot of Google's data lake contained 20 billion datasets, and this number has likely increased since then. Furthermore, we observed that one billion datasets were added to or removed from the data lake each day, with a significant fraction corresponding to transient short-lived datasets. At the same time, the cost of metadata extraction for individual datasets is high, because this extraction typically requires an expensive operation to the source storage system (e.g., accessing the contents of a file in a distributed filesystem). These factors lead to a prohibitive cost to analyze each and every dataset in the data lake.

One way to address this challenge is to prioritize metadata extraction so that the catalog of the data-lake management system covers the "important" datasets. Coming up with a good metric of dataset importance is

difficult. For example, this metric may depend on the type of the dataset, the context in which the dataset is used (e.g., datasets that power user-facing services may be more important), or relationships to other datasets (e.g., datasets may be more important if they are used to derive other datasets). Similarly, while one may argue that transient datasets are not important given their limited lifespan, we discovered that this is not always the case: In some cases it was necessary to analyze short-lived datasets in order to derive metadata for non-transient datasets. For example, provenance links between non-transient datasets often go through temporary transient datasets.

2.3 Post-hoc Management of Metadata

Early on in the design of our system we decided to adopt a *post-hoc* approach to the process of metadata extraction: the system would gather metadata about datasets by analyzing signals from Google's infrastructure (e.g., by processing logs or crawling storage-system catalogs) after these datasets have been accessed or updated. We can contrast this approach with traditional Enterprise Data Management (EDM) systems, which prescribe specific APIs to access datasets and thus act as gateways between teams and their data. EDM systems can gather very precise metadata because they are in the critical path of data access, but they also require an enterprise-wide opt in. Instead, we designed GOODS to operate from the sidelines assuming that teams were free to choose how they access their data. Our goal was to organize the data lake and to bring value to Google's teams without disrupting their practices. We also believe that this post-hoc approach is in line with the nature of a data lake, which allows engineers and analysts to experiment with data in an unfettered fashion.

The downside of this approach is that it becomes more difficult to extract and reason about dataset metadata. First, we now have to deal with uncertainty in the metadata. As an example, consider the problem of inferring a schema for the contents of a dataset whose storage format does not record this information (e.g., a file containing serialized protocol buffers [7]). The analysis of the contents may yield several candidates for the schema, corresponding to different ways to parse the contents, and in the absence of other information we record all of these candidates in the dataset's metadata. Second, the metadata that we collect is heterogeneous because different types of datasets that appear in the data lake (e.g., files, spreadsheets, relational databases, or instances of key-value stores) have different metadata and may require different tools to extract it. A data-lake management system must be able both to extract and handle metadata across a variety of source storage systems and to record this heterogeneous metadata in a single catalog. However, having a single catalog for diverse metadata is also an opportunity to partially lift this heterogeneity in the data lake: the catalog can define a subset of the gathered metadata that is often common across datasets of different types. This common metadata can provide users and services with a unified view of the datasets in the data lake.

3 The Data Lake Relationship Graph

We view the GOODS catalog not only as a collection of metadata describing the datasets in the Google's data lake but also as a representation of relationships among datasets and other related entities (teams, projects, code, and so on).

This approach is inspired by the concept of *knowledge graphs* [3], which are used by modern enterprises to describe entities in the real world and to allow users to search with complex queries. Nodes in such a knowledge graph represent entities in the world (e.g., Tom Hanks, "Forrest Gump") and edges link these entities to each other (e.g., Tom Hanks played a role in "Forrest Gump"). This graph enables an extensible and flexible representation and supports queries such as "female actors who played lead roles in comedies," which require traversing diverse relationships. We can view the structure of a data lake in a similar way, in particular as we link it to other components on enterprise infrastructure. First, we can have relationships between datasets (e.g., dataset A is a new version of dataset B). Second, we can link datasets to other entities, such as jobs that generate the datasets, projects and users that own the datasets and these jobs, or source code that defines these jobs. As

we list the relationships that we identified between datasets in a data lake, it is important to note that we infer all these relationships automatically, in a post-hoc fashion, relying on a variety of information that we can gather inside the enterprise.

The following is a list of relationships among datasets that we identify as important and that we infer as we collect the metadata in the GOODS catalog.

Dataset containment: Some datasets may contain other datasets. For instance, bigtable column families[4] are first-class entriies in the GOODS catalog, and so are the bigtables themselves. We link the latter to the entries for the column families that they contain. This containment information is usually part of the metadata that we can extract directly from specific storage systems.

Provenance: Datasets are produced and consumed by code. This code may include analysis tools such as dashboarding solutions or SQL-query engines, serving infrastructures that provide access to datasets through APIs, or ETL pipelines that encode dataset transformations. For each dataset, we maintain the provenance of how the dataset is produced, how it is consumed, what datasets this dataset depends on, and what other datasets depend on this dataset. We identify and populate the provenance metadata through an analysis of production logs, which provide information on which jobs read and write each dataset. We then create a transitive closure of this graph connecting datasets and jobs, in order to determine how the datasets themselves are linked to one another. However, the number of data-access events in the logs can be extremely high and so can be the size of the transitive closure. Therefore, we trade off the completeness of the provenance associations for efficiency by processing only a sample of data-access events from the logs and also by materializing only the downstream and upstream relations within a few hops as opposed to computing the true transitive closure.

Logical clusters: We identify datasets that belong to the same *logical cluster*. While our definition of clusters is domain-dependent, in general, we usually group the following collections of datasets into a single logical cluster: datasets that are versions of the same logical dataset and that are being generated on a regular basis; datasets that are replicated across different data centers; or datasets that are sharded into smaller datasets for faster loading. Because engineers tend to use specific conventions in naming their datasets, we can identify these logical clusters efficiently by examining the dataset paths. For example, consider a dataset that is produced daily and let `/dataset/2015-10-10/daily_batch` be the path for one of its instances. We can *abstract* out the day portion of the date to get a generic representation of all datasets produced in a month: `/dataset /2015-10-<day>/daily_batch`, representing all instances from October 2015. By abstracting out the month as well, we can go up the hierarchy to create abstract paths that represent all datasets produced in the same year: `/dataset/2015-<month>-<day>/daily_batch`. By composing hierarchies along different dimensions, we construct a *granularity semi-lattice* structure where each node corresponds to a different *granularity* of viewing the datasets.

Content similarity: Content similarity—both at the level of dataset as a whole and at the level of individual columns—is another graph relationship that we extract. Given the size of Google's data lake, it is prohibitively expensive to perform pairwise comparison of all datasets. Instead, we rely on approximate techniques to determine which datasets are replicas of each other and which have different content. We collect fingerprints that have checksums for the individual fields and locality-sensitive hash (LSH) values for the content. We use these fingerprints to find datasets with content that is similar or identical to the given dataset, or columns from other datasets that are similar or identical to columns in the current dataset.

In addition to the relationships that link datasets in the GOODS catalog to each other, we also rely on the rest of Google infrastructure to enrich this relationship graph. While this part of the system continues to grow, we list here some of the relationships that we currently extract:

- information on owners of the jobs that produce or read datasets;
- information on dataset owners and user groups that determine visibility permissions for datasets;
- links between schema that we infer for the datasets and its definition in the source code repository.

Using this (conceptual) graph-based organization enables us to add new relationships between different types of entities easily. An example is the relationship between a dataset representing training data and the visualization of the corresponding features, which we mentioned in Section 1. More important, the graph allows users to navigate the data-lake catalog in a flexible way and to answer rich queries that require linking datasets to other assets in the enterprise.

4 System Design

GOODS maintains a metadata catalog for the data lake (Figure 1). Our previous work [5] details the physical organization of the catalog, the continuous processes that update it, and its usage to power the user-facing services mentioned in Section 1. Here we highlight some of the important technical approaches we adopted in our system to address the challenges in Section 2.

4.1 Leveraging the Data Lake Relationship Graph

The relationships that we have identified in Section 3 are critical in supporting most of the functionality in GOODS. They enable us both to address some of the scalability challenges that we described in Section 2 and to provide new services to the engineers who use GOODS.

First, clustering provides enormous savings in metadata extraction, albeit potentially at the cost of precision. That is, instead of collecting expensive metadata for each individual dataset, we can collect metadata only for a few datasets in a cluster. We can then propagate the metadata across the other datasets in the cluster. For instance, if the same job generates versions of a dataset daily, these datasets are likely to have the same schema. Thus, we do not need to infer the schema for each version. Similarly, if a user provides a description for a dataset, it usually applies to all members of the cluster and not just the one version. When the clusters are large, the computational savings that we obtain by avoiding analysis of each member of the cluster can be significant.

Second, we can use different types of graph edges to propagate metadata when it is too expensive to extract it directly or we simply do not have this metadata. For instance, we can propagate the description of one version of a dataset to all of its versions. Similarly, versions of the same dataset are likely to have the same schema. Naturally, some of this propagation will introduce uncertainty into our metadata—a fact that we are already dealing with at different levels.

Finally, the links to knowledge graphs representing other parts of Google infrastructure, make GOODS a data-centric starting point for users exploring other resources. For instance, a user can find a definition of a protocol buffer in source code and immediately jump to the list of all the datasets that use that protocol buffer as their schema. A new member of a team can find datasets generated by her team. A profile page for a dataset has links to dataset owners, profile pages for jobs that read and write the dataset, and so on.

4.2 Coordinating Modules for Post-Hoc Processing

The GOODS backend uses a large number of diverse batch-processing jobs to collect information from a variety of systems and to add and update new information into the GOODS catalog. This diversity of jobs is a consequence of our post-hoc approach: we collect information from many diverse corners of the Google's internal infrastructure. Each job includes one or more modules, such as crawlers or analyzers. Different GOODS modules have different characteristics that influence how modules are grouped together in jobs, and how jobs are

scheduled. Next we discuss some of these characteristics and a few rules of thumb that we followed to design and optimize the GOODS backend.

First, not all GOODS modules are critical for the smooth functioning of the system. For example, modules that identify the existence of datasets and extract metadata on who owns and who can access the datasets are critical to ensure that the state of the system is fresh and that any changes in access control for the datasets are reflected accurately. On the other hand, modules like schema analyzer that identifies the schema of a dataset—while useful—are not as time sensitive. Therefore, we explicitly allocate more resources to jobs that include critical modules, and schedule non-critical jobs using spare resources.

Second, certain modules may depend on successful run of other modules. For example, a fingerprint analyzer uses the schema identified by the schema analyzer to compute column level fingerprints. Grouping together dependent modules into the same job enables dependent modules to check the status of the all modules they depend on and take an informed decision. While all GOODS modules store the status of execution for each dataset in the catalog itself to avoid repetitive work on failure and between different instances of the same job, grouping dependent modules reduces the overall number of bytes read from the persistent storage backend.

Third, the failure characteristics of modules vary widely. It is important to isolate some modules which are prone to failure to ensure steady progress of other modules. For instance, several of our modules that examine content of datasets use a variety of libraries specific to different file formats. At times, these libraries crash or go into infinite loops. Because we cannot have long-running analysis jobs crashing or hanging, we sandbox such potentially dangerous jobs in a separate process. We then use a watchdog thread to convert long stalls into crashes while allowing the rest of the pipeline to proceed.

Finally, different modules have different computational complexity and therefore different resource footprints. While we schedule modules that have low complexity to run over the entire catalog every day, we avoid re-running computationally expensive modules unless there is a strong signal that a re-run will yield different results. For example, unless a dataset is modified, the fingerprint for the contents is unlikely to change and therefore the fingerprint analyzer can bypass such dataset.

While some of these design choices were part of our initial design of the GOODS system, we made many of them after experiencing an issue and redesigning parts of our system to address it.

4.3 Search Ranking

As we mentioned in the introduction, dataset discovery through search is the most frequent use case for GOODS. An important design choice for us was to build the search functionality at the level of logical clusters: We index the metadata that describes the logical cluster corresponding to a collection of related datasets (e.g., daily versions of the same dataset) and the user sees results corresponding to these logical datasets. This decision allowed us to compress search results in a meaningful way, instead of overwhelming the user with many similar datasets that match the same query (e.g., showing datasets that differ only on one component of their path denoting their version). We also experimented with the alternative of indexing metadata at the level of each physical dataset, but the end-user experience suffered from the sheer number of similar results.

We faced a few technical difficulties in building the search index. For example, we needed to propagate the metadata from individual members of the cluster itself so that users can search for it. However, by far the biggest challenge was to design a good ranking function for search results. In general, it is hard to overstate the importance of good ranking for search. The problem has unique characteristics in the case of dataset search (and is different than, say, web or bibliographic search) because of the domain-specific signals that can determine the relevance of a dataset to a search query. After much experimentation (and a few false starts), we ended up using a mix of standard IR-type signals (e.g., how well the search terms match the metadata) with domain-specific signals derived from each dataset's metadata. For instance, we found that provenance relationships among datasets provide a strong relevance signal. Specifically, it is common for teams to generate denormalized versions of some "master" dataset in order to facilitate different types of analysis of the master data. These

denormalized datasets can match the same search keywords as the master dataset, yet it is clear that the master dataset should be ranked higher for general-purpose queries or for metadata-extraction. Another example comes from provenance relationships that cross team boundaries, when the dataset from one team is processed to create a dataset in the scope of another team or project. In this case, we can boost the importance of the input dataset as evidenced by its usage by an external team. The output dataset is also important, because we can view it as an origin of other datasets within the external project. Dataset type is another example of a domain-specific signal: for instance our ranking function primes datasets that correpsond to relational databases, because the latter tend to have richer metadata and be more tailored for wide usage compared to, say, files on a distributed filesystem.

Ranking methods for dataset search remains an interesting research problem. It is particularly interesting to consider how techniques from other domains could apply in a data-lake setting, e.g., what would be the equivalent of personalized pagerank when searching for a dataset. However, as we discuss in the next section, our experience with users shows that they have different needs for ranking depending on the purpose of their search. This evidence indicates that a data-lake management system should support several ranking methods and allow users to choose at search time which one to use.

5 Future Directions

After deploying GOODS for the initial use cases that we mentioned in Section 1, we had a chance to observe how the system got adopted, to interact with users, and to receive feedback on feature requests. Based on this information, we have identified a few directions that are interesting for further development on GOODS and, more generally, for the type of functionality that a system for data-lake management can offer.

- **Building a community around datasets** GOODS allows users to share and exchange information about datasets, and to augment the metadata in the catalog with domain-specific knowledge. We view this functionality as the means to develop a community whose shared goal is the curation of the data lake. In this spirit, we can add more community-like features including comments on datasets, reviews, or Q&A functionality, to name a few. The goal is to foster a culture of joint data stewardship and of best practices on adding new datasets to the data lake.

- **Rich analytics over the data lake** The first use case that we targeted with GOODS was dataset discovery within the data lake, through a simple keyword-search interface with relevance ranking. Over time, we found that users "outgrew" this modality and started asking for more flexible ways to query the catalog. The requests ranged from different options for result ranking (e.g., rank datasets by size or by modification timestamp) to full SQL access over the catalog. (In fact, GOODS itself has an internal monitoring component that tracks the state of the data lake through SQL queries over the metadata.) In some cases, users also found it convenient to explore the catalog through a graph visualization based on provenance relationships, which points to the idea of exposing slices of the catalog through specialized user interfaces. Thinking forward, we can also view the catalog as a temporal store that enables comparisons between snapshots of the data lake and thus the discovery of trends. Another option is to view GOODS as a generator of a stream of metadata events, where each event encodes the creation of a dataset, the discovery of a provenance edge, or any other piece of metadata that can be stored in the catalog. Under this model, users can issue continuous queries to monitor the contents of the data lake.. For example, a team can set up a continuous query to monitor accesses to its datasets by other teams, or a user can monitor the generation of datasets by daily runs of some pipeline. In general, our experience with GOODS is that there is value in enabling rich data analytics over the catalog, both to allow users to explore the data lake more effectively but also to monitor the overall state of the data lake.

- **Beyond post-hoc** Our initial approach of crawling and analyzing datasets in a post-hoc manner enabled the collection of basic metadata for datasets stored in many different systems in a minimally invasive

manner. However, there are two main downsides to the post-hoc approach. First, the data in the catalog has a time lag, which conflicts with user expectation of the catalog immediately reflecting the creation or modification of any metadata. Second, the analyzers in GOODS collect only generic, and often uncertain, metadata. As GOODS got adopted within the company, many teams expressed the desire to use GOODS infrastructure to store, retrieve, share, and serve custom metadata for their datasets, ideally within a short time interval of such changes taking place. In order to tackle such use cases, we envision a hybrid approach that supports one-off deeper integration with storage infrastructures to reduce the time lag for discovery, and APIs for teams to register datasets and to contribute custom metadata to the catalog.

- **Acting on data** The information in the catalog not only helps users understand the datasets that they know about but also enables them to discover what they can do with their datasets, how they can use datasets in other tools, or to discover new related datasets. More concretely, one of the very popular features of profile pages in GOODS are pre-populated queries and code snippets that use the path and schema information, which users can simply copy and paste into other tools. We can envision extensions of this feature that are based on a deeper analysis of the dataset's metadata and that can help the user act on the dataset with more elaborate tools; for example, we can automatically build a dashboard for the dataset based on its characteristics. These extensions can become more powerful if we can leverage the relationships encoded in the catalog. For instance, if GOODS can tell us that a key column in our dataset has very similar context to a key column in another dataset, then the two datasets might be candidates for joining and the user may even be presented with possible actions on the join results. This specific example is intriguing, as the data-lake management system helps the user understand what datasets *could* exist in the data lake instead of merely summarizing what has already been generated.

Overall, a data-lake management system should promote the treatment of datasets as first-class objects within the computing infrastructure of the enteprise. A big part of this goal involves services that make it easier for engineers and analysts to integrate datasets within their workflow in an organic fashion, and this direction has been the main focus of our work with GOODS. Furthermore, through these services the system can instill best practices for dataset management and break team-delinated silos, thus fostering a culture of responsible data stewardship across the enterprise.

References

[1] Azure data lake. `https://azure.microsoft.com/en-us/solutions/data-lake/`.

[2] Data lakes and the promise of unsiloed data. `http://www.pwc.com/us/en/technology-forecast/2014/cloud-computing/features/data-lakes.html`.

[3] A. Brown. Get smarter answers from the knowledge graph. `http://insidesearch.blogspot.com/2012/12/get-smarter-answers-from-knowledge_4.html`, 2012.

[4] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. E. Gruber. Bigtable: A distributed storage system for structured data. *ACM Trans. Comput. Syst.*, 26(2):4:1–4:26, June 2008.

[5] A. Halevy, F. Korn, N. F. Noy, C. Olston, N. Polyzotis, S. Roy, and S. E. Whang. Goods: Organizing google's datasets. In *Proceedings of the 2016 International Conference on Management of Data*, SIGMOD '16, pages 795–806, New York, NY, USA, 2016. ACM.

[6] I. Terrizzano, P. M. Schwarz, M. Roth, and J. E. Colino. Data wrangling: The challenging journey from the wild to the lake. In *CIDR 2015, Seventh Biennial Conference on Innovative Data Systems Research, Asilomar, CA, USA*, 2015.

[7] K. Varda. Protocol buffers: Google's data interchange format. *Google Open Source Blog, Accessed July*, 2008.

Data Services Leveraging Bing's Data Assets

Kaushik Chakrabarti, Surajit Chaudhuri, Zhimin Chen, Kris Ganjam, Yeye He

Microsoft Research
Redmond, WA
{kaushik, surajitc, zmchen, krisgan, yeyehe}@microsoft.com

Abstract

Web search engines like Bing and Google have amassed a tremendous amount of data assets. These include query-click logs, web crawl corpus, an entity knowledge graph and geographic/maps data. In the Data Management, Exploration and Mining (DMX) group at Microsoft Research, we investigate ways to mine the above data assets to derive new data that can provide new value to a wide variety of applications. We expose the new data as cloud data services that can be consumed by Microsoft products and services as well as third party applications. We describe two such data services we have built over the past few years: synonym service and web table service. These two data services have shipped in several Microsoft products and services including Bing, Office 365, Cortana, Bing synonyms API and Bing Knowledge API.

1 Introduction

Web search engines like Bing and Google have amassed a "treasure trove" of data assets. One of the most important assets is the query-click log which contains every search query submitted by a user, the urls and other information (e.g., answers) returned by the search engine, and the items clicked on by the user. Other important assets include the web crawl corpus, an entity knowledge graph (that contains information about named entities like people, places and products) and geographic/maps data.

The above data assets are leveraged by the web search engine to deliver a high quality search experience. For example, query-click log is used to improve the quality of web result ranking. The entity knowledge graph is used not only to improve web result ranking but also to compose the "entity information card" for entity queries. In the Data Management, Exploration and Mining (DMX) group at Microsoft Research, we explore ways to mine the above data assets to *derive new data* that can provide new value to a wide variety of applications. We expose the new data as cloud data services that can be consumed by Microsoft products and services as well as third party products and services. The main idea is depicted in Figure 1.

Synonym service: Let us start with an example of such a data service called synonym service. People often refer to a named entity like a product or a person or a place in many different ways. For example, the camera 'Canon 600d' is also referred to as 'canon rebel t3i', the film 'Indiana Jones and the Kingdom of the Crystal Skull' also as 'indiana jones 4' the person 'Jennifer Lopez' also as 'jlo' and the place 'Seattle Tacoma International Airport' also as 'sea tac'. We refer to them as entity synonyms or simply synonyms (in contrast to other types

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

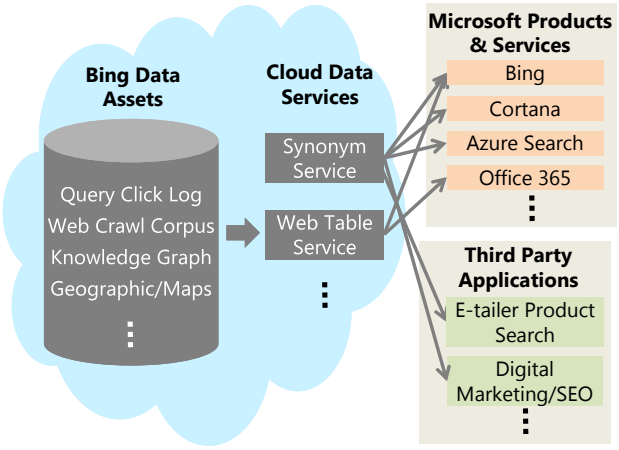


Figure 1: Data services leveraging Bing's data assets

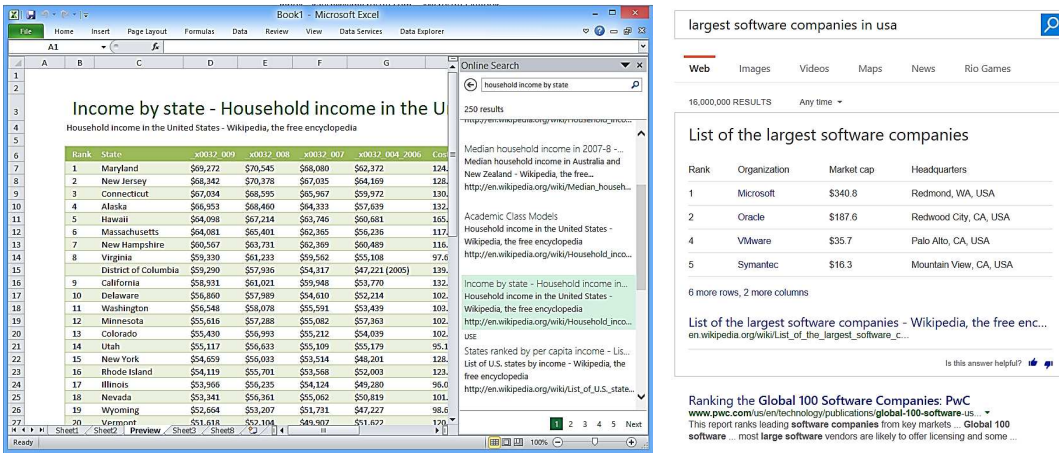
of synonyms such as attribute synonyms [15]). Consider the product search functionality on an e-tailer site like bestbuy.com. Without the knowledge of synonyms, it often fails to return relevant results. For example, when the user searches for 'indiana jones and kingdom of crystal skull' on bestbuy.com, it returns the DVD for that movie (the desired result). However, if she chooses to search using the query 'indiana jones 4', it fails to return the desired result. This is because bestbuy.com does not have the knowledge that the above film is also referred to as 'indiana jones 4'. If we can create a data service that computes the synonyms for any entity by leveraging the rich data assets of a search engine, it will bring tremendous benefit to such e-tailers [9]. It will also be valuable to specialized entity portals/apps like movie portals (Fandango, Moviefone), music portals (Pandora, Spotify), sports portals (ESPN, NFL) and local portals (Yelp, Tripadvisor). Hence, we built such a data service called the synonym service [7, 18].

Challenges in synonym service: The main challenge is to mine synonyms in a domain independent way with high precision and good recall. While there is significant work on finding similar/related queries [3, 4, 12], there is little work on mining synonyms with the above precise definition of synonymity (i.e., alternate ways of referring to the *exact same entity*). Furthermore, existing synonym mining works rely mostly on query co-clicks [10], which we find in practice to be insufficient to ensure very high precision (e.g., over 95%) that is required for scenarios described above. In our work, we develop a variety of novel features to complement query log features that achieve high precision and recall. We describe those techniques in detail in Section 2.

Impact of synonym service: The synonym service is being used extensively by Microsoft products and services as well as by external customers. Many Bing verticals like sports and movies use the synonym data to improve their respective vertical search qualities. In Bing Sports for example, when a user asks the query 'tampabaybucs', entity synonyms will help to trigger the entity card for "Tampa Bay Buccaneers". Our synonyms are also used inside Bing's entity-linking technology which in turn is used in several applications like Bing Snapp [1], Ask Cortana [2], and Bing Knowledge API [3]. For external customers (e.g., e-tailers), synonym technologies can be accessed from the Bing developer center as a web service called Bing synonym API [1]. An entity name can be submitted as input, and all synonyms of that entity will be returned by the service. This is used in customer scenarios such as query enrichment and catalog enrichment. Thousands of customers subscribe to this API.

Web table services: A second data service [4] we have built is the web table service. Although the web crawl corpus mostly consists of unstructured data like textual documents, images and videos, there is also a vast amount of structured data embedded inside the html documents. For example, there are more than a billion html tables,

[1] https://www.microsoft.com/en-us/store/p/snapp/9nblggh09m4d
[2] https://www.microsoft.com/en-us/mobile/experiences/cortana/
[3] https://www.bing.com/widget/knowledge
[4] Actually two closely related services



(a) (b)

Figure 2: (a) Web table search service in action in Excel PowerQuery (b) Web table answer service in action on Bing.

html lists and spreadsheets on the web. Each such table/list/spreadsheet contains a set/list of named entities and various attributes about those entities. For example, the html table embedded inside `en.wikipedia.org/wiki/List_of_U.S._states_by_income` contains the median household income for all the U.S. states. These tables are often very valuable to information workers. For example, a business data analyst often needs to "join" business data with public data. Consider a data analyst analyzing sales numbers from various U.S. states (say, in Excel) and wants to find how strongly they are correlated with the median household income. She needs to join the former with the table in `en.wikipedia.org/wiki/List_of_U.S._states_by_income`. It is hard for the analyst to discover the above table and also to import it into Excel in order to do the join. It would be very valuable if we can index such tables and allow Excel users to easily search for them (e.g., using keyword search) as shown in Figure 2(a). We built a data service called *web table search service* for the above purpose.

A substantial fraction of queries on Bing and Google can be best answered using a table. For example, for the query 'largest software companies in usa', it is much better to show the table from Wikipedia containing the largest software companies (`en.wikipedia.org/wiki/List_of_the_largest_software_companies`) than just the blue links as shown in Figure 2(b). We refer to the above class of queries as *list-intent queries* as the user is looking for a *list* of entities. We built a data service called *web table answer service* for the above purpose. We explore other classes of queries as well for table answers. Although both web table search and web table answer services take a keyword query as input and returns web tables, they are quite different. The former always returns a *ranked list of tables* relevant to the query. On the other hand, since the latter is invoked by a web search engine where the top result spot is reserved for the best possible answer (among all possible types of answers as well as top algorithmic search result), the desired output for the latter is a *single table* if is the best possible answer, otherwise it should return nothing.

Challenges in web table services: Most of raw HTML tables (i.e., elements enclosed by the <table></table> tags) do not contain valuable data but are used for layout purposes. We need to identify and discard those tables. Furthermore, among the valuable tables, there are multiple different types. We need to distinguish among them in order to understand their semantics which in turn is necessary to provide high quality table search and table answer services. These are challenging problems as they cannot be accomplished via simple rules [5]. Furthermore, providing a high quality table ranking as well as providing table answers with high precision and good coverage are hard problems as well. While there is extensive prior work on table extraction [6, 5, 21, 20, 14, 11], there is limited work on the latter two challenges: table ranking and providing table answers with high precision. In Section 3, we present our table extraction techniques and highlight their differences

with prior table extraction work. We also present the novel approaches we have developed for the latter two challenges.

Impact of web table services: The web table search service was released in Excel PowerQuery in 2013 [2]. It allows Excel users to search and consume public tables directly from Excel. A screenshot is shown in Figure 2(a). The web table answer service has been shipping in Bing since early 2015. It shows table answers for list-intent and other types of queries with 98% precision and with a current coverage of 2%. A screenshot is shown in Figure 2(b).

2 Synonym Service

Given an entity name, the synonym service returns all the synonyms of the entity. We mine all possible synonym pairs in an offline process (200 million pairs in our latest version); the service simply performs a lookup into that data. Currently the service is hosted as a public Bing service in Bing Developer Center [1]. We focus on the key technologies used in offline mining process in the rest of this section.

2.1 Synonym Mining Requirements

Based on the intended use cases, we summarize key requirements of synonym mining as follows.

Domain independence. Entity synonyms are ubiquitous in almost all entity domains. A natural approach is to leverage authoritative data sources specific to each entity domain to generate synonyms. For example, one may use extraction patterns specific to IMDB for movie synonyms. However, techniques so developed are specific to one domain that cannot easily generalize to different domains. Given the scale and the variety of the synonyms we are interested in, developing and maintaining specific techniques for each domain is unlikely to scale. Our goal is to develop domain-independent methods to systematically harvest synonyms for all domains.

High precision. Since the output of our service is used by an array of Microsoft products and third party retailers, who would for example use synonyms to enrich their product catalogs, the precision of our synonyms needs to be very high (e.g., above 95%). Entities that are only related but not equivalent (e.g., "Microsoft office 2015" and "Microsoft office 2013") should not be considered as synonyms, for otherwise they will adversely affect downstream applications like product search.

Good recall. In addition to high precision, we want to discover as many synonyms as possible. The types of synonyms we are interested in ranges from simple syntactic name variations and misspellings (e.g., "Cannon 600d" for the entity "Canon 600d"), to subset/superset variations (e.g., "Canon EOS 600d" for the entity "Canon 600d"), to more semantic synonyms (e.g., "Canon rebel t3i" or "t3i slr" for the entity "Canon 600d"). The synonyms we produce should ideally cover all these types.

Freshness. Since new entities (movies, products, etc.) are constantly being created, and new names coined for existing entities, we want the synonym data to be up-to-date. The mining process thus needs to be refreshed regularly to reflect recent updates, and hence needs to be easily maintainable with minimal human intervention.

2.2 Prior Work on Synonym Mining

Prior work on discovering entity synonyms relies on query co-clicks [10]. Our experience suggests that this alone often leads to many false positives. For example, name pairs like "iphone 6" and "iphone 6s", or "Microsoft office 2015" and "Microsoft office 2013" share significant co-clicks and are almost always predicted as synonyms. We find synonyms so generated to have considerably lower precision than the 95% requirement, and incorrect synonyms like the ones above are particularly damaging to application scenarios such as product catalog enrichment. In this work we develop novel features utilizing a variety of orthogonal data assets to overcome the limitations of query logs.

The problem of finding semantically similar/related queries is related to synonym-finding, and is extensively studied in the literature [3, 4, 12]. The fuzzy notion of semantic relatedness used in this body of work, however,

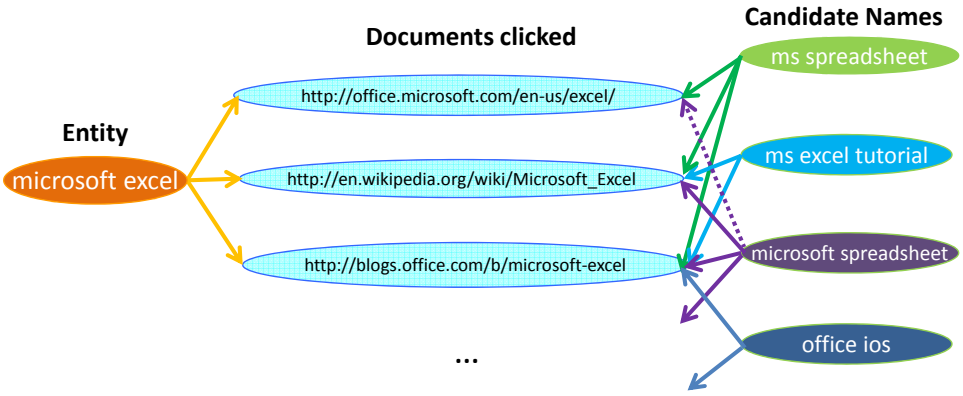


Figure 3: Example query log click graphs

does not match our precise requirement of entity-equivalence for synonyms, and is thus insufficient for high precision entity synonyms that we intend to produce.

2.3 Exploiting Bing Data Assets for Synonym Mining

At a high level, our synonym mining has three main steps: (1) generate billions of candidate name pairs that may be synonyms; (2) for each candidate pair, compute rich features derived from data assets such as Bing query logs, web tables, and web documents; (3) utilize manually labeled training data and machine learning classifiers to make synonym predictions for each candidate pair.

We start by generating pairs of candidate names for synonyms. In order to be inclusive and not to miss potential synonyms in this step, we include all pairs of queries from the query logs that clicked on the same document for at least 2 times. This produces around 3 billion candidate synonym pairs.

For each candidate pair, we then compute a rich set of features derived from various data sources. Given the feature vectors, we use training data and boosted regression tree [13] to train a binary classifier to predict synonyms. Since boosted regression tree is a standard machine learning model, we will focus our discussions on the design of features using various data sets.

Query log based features. Query logs are one of the most important data assets for synonym mining. The key idea here is the so-called "query co-clicks" [7, 10]. Namely, if search engine users frequently click on the same set of documents for both query-A and query-B, then these two query strings are likely to be synonyms. The rationale here is that search engines clicks form implicit feedback of query-document relevance, which when aggregated over millions of users and a long period of time, provide robust statistical evidence of synonymity between two query strings.

In the example of Figure 3, suppose "microsoft excel" is the entity of interest. For this query users click on three documents in the middle as represented by their urls. If we look at other queries whose clicks share at least one document, we can see that "ms spreadsheet" clicks on the exact same set of documents (a true synonym of "microsoft excel"). Both "microsoft spreadsheet" and "ms excel tutorial" share two co-clicks with "microsoft excel". While the first query is a true synonym, the second is only a related entity (tutorial) and thus not a synonym. Lastly, query "office ios" shares only one clicked document with "microsoft excel", indicating a lower degree of semantic relationship.

Intuitively, the higher the overlap between the clicks shared by two query strings, the more likely they are actual synonyms. We use a number of metrics to quantify relationships between two queries – Jaccard similarity and Jaccard containment when representing their clicked documents as sets, and Jenson-Shannon divergence when representing click frequencies on documents as distributions.

We further encode, for each query, frequency-based features such as the number of clicks, the number of

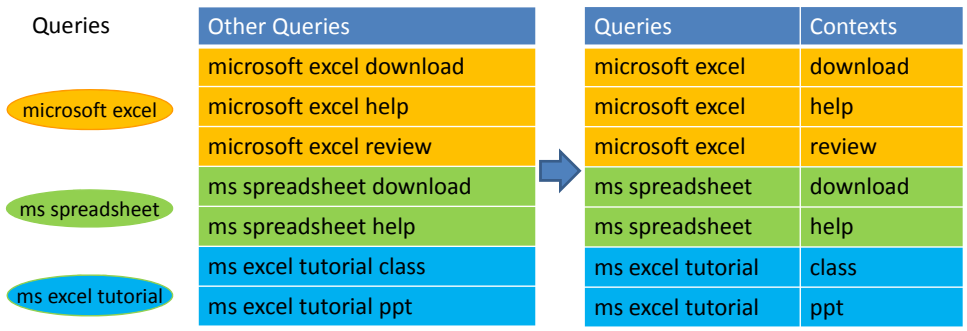


Figure 4: Example query contexts

distinct documents clicked and domains clicked, etc. as additional features, which allow machine learning models to differentiate popular queries from less popular ones for example.

One problem we encounter in using query logs is that click-edges in the query-document bipartite graph are sometimes too sparse. To overcome sparsity, we enhance the co-click graph by introducing artificial edges using *pseudo-documents* [7]. Specifically, we construct a pseudo-document $p(d)$ for each clicked document d as the union of the tokens in queries that click on d. Based on this definition, in Figure 3 for example, let d_1 be the first document, then $p(d_1) = \{\text{microsoft, excel, ms, spreadsheet}\}$. We can then add an artificial edge between query q and document d if the tokens of q are contained by the pseudo-document $p(d)$. In Figure 3, because the tokens of query "microsoft spreadsheet" is contained by $p(d_1)$, we add an edge between the two (shown by dotted edge) as if there exists such a click in the original click graph. Using this enhanced bipartite graph, we can define set-based similarity and distribution-based similarity like before. These form a group of pseudo-document-based features.

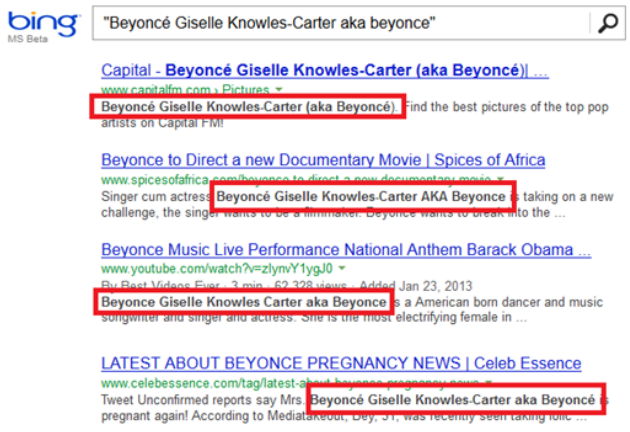
Notice that in the example of Figure 3, "ms excel tutorial" and "microsoft excel" are related entities with relatively high co-clicks, which however should not be recognized as synonyms because they are of different entity-types (e.g., one is software whereas the other is a tutorial). Similarity based features alone may mistakenly predict them as synonyms. In order to penalize pairs of candidates with different types, we introduce query-context based features. Figure 4 gives an example of query-contexts. For each target query (e.g., "microsoft excel", "ms spreadsheet", etc.), we find from the query logs additional queries that have the target query as a prefix or suffix. For "microsoft excel", we find queries like "microsoft excel download" and "microsoft excel help". From these, we compute the suffix-context of query "microsoft excel" as "download", "help" and "review" (and their respective frequencies). These are indicative of query types, because queries of the same entity type tend to share similar contexts. In this example the context of "microsoft excel" is very similar to that of "microsoft spreadsheet", showing that they are likely of the same entity type. However the context of "microsoft excel" is quite different from "ms excel tutorial", which we use as evidence that the pair may be of different types. We compute distributional similarity between two candidates' query contexts such as Jensen-Shannon divergence [16] as features for type similarity.

Web table based features. While query logs based features are powerful positive signals for semantic similarity, in many cases they are insufficient to differentiate between true synonyms, and pairs of highly related names that are non-synonyms. For example, the pair "Harry Potter and the Deathly Hallows: Part 1" and "Harry Potter and the Deathly Hallows: Part 2" share substantial co-clicks in the query logs. It is thus difficult to know from the query logs alone that they should not be synonyms.

For this we leverage another set of signals from HTML tables extracted from web pages indexed by Bing. Specifically, we observe that if two entities occur more frequently (than pure coincidence) in the same table columns, they are likely to be different entities and thus not synonyms. The reasoning is that humans are unlikely to put two synonymous mentions of the same entity in the same table column. In a real web table example shown in Figure 5(a), the fact that "Harry Potter and the Deathly Hallows: Part 1" and "Harry Potter

Motion Picture
Harry Potter and the Philosopher's Stone
Harry Potter and the Chamber of Secrets
Harry Potter and the Prisoner of Azkaban
Harry Potter and the Goblet of Fire
Harry Potter and the Order of the Phoenix
Harry Potter and the Half-Blood Prince
Harry Potter and the Deathly Hallows – Part 1
Harry Potter and the Deathly Hallows – Part 2

(a) Value co-occurrence in tables as negative features



(b) AKA text patterns

Figure 5: Web tables and document features

and the Deathly Hallows: Part 2" are co-occurring in the same table column indicates that they are likely to be distinct entities instead of synonyms.

We aggregate such statistical information from over 100 million tables extracted from the web, and use the point-wise mutual information of two candidate names occurring in the same table columns as an additional feature.

Another way in which we can leverage web tables is to directly utilize synonymous column pairs often seen in tables. Figure 6 gives a few example tables in different domains where two values from the same row are synonyms. We use an initial version of synonym scores to discover such pairs of synonymous columns from the web tables corpus, and then compute the number of times name-A and name-B occur in same rows of synonymous columns as additional features. We find this feature to be helpful in improving coverage of tail entities.

Web document based features. Web documents indexed by Bing provide an orthogonal source of signals. In particular, we consider two main groups of signals: text patterns, and anchor texts.

For text patterns, we scan all documents in Bing's crawl for patterns such as "name-A, also known as name-B", "name-A, aka name-B", "name-A, otherwise known as name-B", etc. Given the huge variety and extensive coverage of the Web documents, this helps us to capture a wide range of synonymous names. Figure 5(b) shows an example to illustrate this idea. The names "beyonce giselle knowles carter" and "beyonce" frequently co-occur in these AKA text patterns, and they are indeed synonyms. We simply count the number of times name-A and name-B occur in such patterns in all Bing documents as a feature for these two names. We note that the idea of using text patterns is well-studied especially in the literature of information extraction (e.g., [8, 17]). We leverage such patterns in the context of synonym mining for complementary signals derived from an orthogonal data source to achieve improved result quality.

For anchor texts, we utilize the observation that anchor texts describing the same hyper-links are often interchangeable in meanings. So for each hyperlink in Wikipedia, we extract all anchor texts associated with that link, and count the number of times name-A and name-B are used as anchor texts pointing to the same link. For instance, both "Jennifer Lopez" and "JLO" are used as anchor texts describing the link pointing to her Wikipedia page, and we use their respective frequencies pointing to the same page as features. This is another beneficial feature derived from distant supervision.

Wikipedia and Wiktionary. Wikipedia has rich redirect information. For example, "J.Lo" and "JLO" are redirected to the "Jennifer Lopez" page on Wikipedia. We extract such information from a dump of the Wikipedia pages as features. Synonyms as defined by traditional thesaurus are also useful. We parse such information from Wiktionary as additional features.

CH_3CHO	acetaldehyd
CH_3CO_2H	acetic acid
$(CH_3)_2CO$	acetone
CH_3CN	acetonitrile
$C_4H_{10}O_2$	1,2-Butanediol
$C_4H_{10}O_2$	1,3-Butanediol
$HOCH_2CH_2CH_2CH_2OH$	1,4-Butanediol
$C_6H_{14}O_2$	2-Butoxyethanol
$CH_3CH_2CH_2COOH$	butyric acid
$HN(CH_2CH_2OH)_2$	diethanolamine
$HN(CH_2CH_2NH_2)_2$	diethylenetriamine
$(CH_3)_2NC(O)H$	dimethylformamide
$C_4H_{10}O_2$	dimethoxyethane
$(CH_3)_2SO$	dimethyl sulfoxide

(a)

Homo sapiens	Human
Pongo abelii	Sumatran orangutan
Sus scrofa	Pig
Equus caballus	Horse
Mus musculus	Mouse
Rattus norvegicus	Rat
Monodelphis domestica	Gray short-tailed opossum
Danio rerio	Zebra fish
Canis lupus familiaris	Dog
Ailuropoda melanoleuca	Panda
Oryzias latipes	Medaka
Xenopus (Silurana) tropicalis	Western clawed frog
Aedes aegypti	Yellow-fever mosquito
Tribolium castaneum	Red flour beetle
Hydra magnipapillata	Hydra hydrozoan
Ciona intestinalis	Sea squirt
Pediculus humanus corporis	Human lice

(b)

Vitamin B_1	Thiamine	
Vitamin B_2	Riboflavin	
Vitamin B_3	Niacin	
Vitamin B_5	Pantothenic acid	
Vitamin B_6	Pyridoxine	

(c)

Figure 6: Example tables with synonymous columns

Other syntactic features. As additional negative signals, we describe the syntactic difference between two names using character types and lengths, to leverage observations such as if two names only differ by a number, they are unlikely to be synonyms (e.g., "ford mustang" and "ford mustang 2015").

Quality evaluation. After producing feature vectors for all pairs of candidate names, we train a boosted regression tree model [13] to predict synonymity for each name pair. Based on our calibration of classification scores using a holdout set of labeled data, we threshold to produce over 200 million synonym pairs with very high precision.

We take a sample from the produced name pairs and perform manual labeling to assess result quality. The result is shown in Table 1. Recall here is defined as simply the number of synonyms produced for each sampled name, where tier-1 recall only counts miss-spellings and name variations ("Canon 600d" and "Cannon 600d"), tier-2 only counts subset/superset synonyms ("Canon 600d" and "Canon EOS 600d"), and tier-3 counts semantic synonyms that do not belong to the two previous categories ("Canon 600d" and "EOS rebel t3i").

Precision	Tier-1 recall	Tier-2 recall	Tier-3 recall
0.973	1.43	2.12	1.12

Table 1: Quality evaluation of synonyms. Recall is defined as the avg. number of synonyms produced per entity.

3 Web Table Services

We describe the key techniques that enable the web table search and web table answer services. Both services rely on the *web table extraction and understanding* component which we describe first. The web table search service takes a keyword query as input and returns a *ranked list of web tables* relevant to the query. Since the tables are already extracted, the key remaining challenge is to provide high quality ranking over those tables. Ranking features used in web page ranking or prior work on web table search are not adequate for our purpose [19]. We develop novel ranking features to address this challenge. We describe them in the second subsection. The web table answer service also takes a keyword query (a general web search query) but returns *a single table* if it is the best possible answer (among all possible types of answers as well as top algorithmic search result), otherwise it returns nothing. The main challenge here is to provide table answers with a high precision and good coverage. We develop a novel approaches to address this challenge which we describe in the last subsection.

(a)

Leica Q specifications

Price	
MSRP	$4250
Body type	
Body type	Large sensor compact
Body material	Magnesium alloy
Sensor	
Max resolution	6000 x 4000

(b)

Name	Type[3]	County[4]	Population		Land area[5]	
			2014 estimate [6]	2010 Census [5]	sq mi	km²
Aberdeen	First	Grays Harbor	16,255	16,896	10.65	27.6
Airway Heights	Code	Spokane	6,545	6,114	5.12	13.3
Algona	Code	King	3,133	3,014	1.30	3.4
Anacortes	Code	Skagit	16,232	15,778	11.76	30.5
Arlington	Code	Snohomish	18,808	17,926	9.30	24.1
Asotin	Code	Asotin	1,290	1,251	1.05	2.7
Auburn	Code	King Pierce[Note 1]	76,347	70,180	29.62	76.7
Bainbridge Island	Code	Kitsap	23,293	23,025	27.61	71.5
Battle Ground	Code	Clark	18,930	17,571	7.86	20.4
Bellevue	Code	King	136,426	122,363	33.48	86.7

(c)

Figure 7: Different types of HTML tables. (a) is a layout table, (b) is an attribute-value table, (c) is a relational table.

3.1 Prior Work on Web Tables

Web table extraction: There is significant work on web table extraction in the literature [6, 5, 21, 20, 14, 11]. One of the key challenges is to distinguish between the different types of tables, viz. layout, relational and attribute-value tables. Most of the above works formulate the problem as a machine learning classification task and design features for the task. Examples of features that distinguish relational tables from other ones are cell value consistency along columns and the number of empty cells [6, 5, 21]. We adopt a similar approach in our paper. While there is overlap between our features and those proposed in earlier work, we also employ novel features like header features to improve accuracy.

Web table search: Venetis et. al. developed a seminal system for keyword search on web tables [19]. They annotate each table with column labels that describe the class of entities that appear in that column (using a isA database). Then they look for matches between the query keywords and the annotations as well as the column names. This results in much better quality than simply piggybacking on Google's ranking of web pages. Their approach is not adequate for our services for multiple reasons. For the web table search service, some classes of queries require us to find matches inside the data cells (e.g., row-subset queries, entity-attribute queries). The above work does not consider such matches. Second, for the web table answer service, there is a hard requirement of 98% precision while maintaining good coverage. The above approach is not optimized for this requirement.

3.2 Web Table Extraction and Understanding Component

The goal of this component is to extract tables from Bing's web crawl corpus and perform table understanding and annotation. These tables are subsequently used in web table search and web table answer services. We first obtain the raw HTML tables (i.e., elements enclosed by the <table></table> tags) by parsing each document in the crawl corpus into a DOM tree. The main challenge is that not all the raw HTML tables contain valuable data; a majority of them are used for layout purposes (e.g., page layout, form layout). Figure 7(a) shows an example of a layout table. We need to identify and discard these tables. Among the valuable tables, there are two main types of tables, namely *relational* and *attribute-value* tables. A relational table is one where each row corresponds to a named entity and the columns correspond to different attributes of the entities [5, 6]. Figure 7(c) shows an example of a relational table where each row corresponds to a city (in Washington) and the columns correspond to different attributes like the county the city belongs to, the city's population and the city's land area. On the other hand, an attribute-value table contains different attributes and its values of a single entity [11]. Figure 7(b) shows an attribute-value table for the entity 'Leica Q'. We need to distinguish between the two

types of tables in order to understand their semantics which in turn is necessary to provide high quality table search and table answer services. These are challenging problems as they cannot be accomplished via simple rules [5].

We first present our approach to distinguish between the different types of tables. Like prior work, we formulate the problem as a machine learning classification task and propose features for the task [6, 5, 21, 20]. We improve upon those works by proposing novel features like header features. We describe all the features below for the sake of completeness.

Distinguishing among different types of tables We identify several features to distinguish the different types of tables. We categorize the features as follows:

• *Column-wise homogeneity*: One of the main features that distinguishes relational tables from attribute-value tables is column-wise homogeneity. The values in a column of a relational table are typically homogeneous as it contains values of different entities *on the same attribute*. For a string-valued attribute, all cells have string values and their string lengths are similar to each other (e.g., three leftmost columns in table in Figure 7(c)). Similarly, for a numeric attribute, all cells have numeric values (e.g., four rightmost columns in table in Figure 7(c)). A column in a relational table typically does not contain a mix of string and numeric values. On the other hand, the second (counting from left) column of an attribute-value table typically contains a mix of string and numeric values as it contains values on *different attributes*. For example, the second column in the table in Figure 7(b) contains such a mix. We capture the above observations by proposing the following features: (i) fraction of string-valued cells in first and second columns (ii) fraction of numeric-valued cells in first and second columns (iii) mean and standard deviation of string lengths of cells in first and second columns (iv) mean and standard deviation of string lengths of cells averaged across all columns. A high value on either (i) or (ii) represents homogeneity; we capture that by computing a derived feature that computes the max between (i) and (ii).

• *Row-wise homogeneity*: Layout tables are also often column-wise homogeneous, so column-wise homogeneity alone cannot distinguish between relational and layout tables. We use row-wise homogeneity for that purpose. All rows in a relational table typically have the same number of cells and they are typically non-empty. On the other hand, rows in a layout table might have different number of non-empty cells. For example, in the layout table in Figure 7(a), the first, second and third rows have 2, 3 and 1 non-empty cells respectively. We compute the standard deviation of the number of cells (and non-empty cells) in different rows as features.

• *Row and column counts*: Row and column counts also help distinguish between the different types of tables. Layout tables often have a very small number of rows and/or a very small number of columns (e.g., the layout table in Figure 7(a) in Figure has only 3 rows). Attribute-value tables typically have 2 columns and a small number of rows. On the other hand, relational table often have much larger number of rows and more than 2 columns. We compute number of number of rows and number of columns as features.

• *Presence of header*: If a table contains a header row that is explicitly marked by `<th>` or `<thead>` tags and it "aligns" with the rest of the table (i.e., has the same number of columns as other rows), it is most likely a relational table. This is especially true if there is a column where all the cell values are numeric except the cell value in the `<th>` or `<thead>` row (which has a string value). We compute boolean features to capture the above insights.

We manually labeled about 5000 tables as relational, layout and attribute-value and trained a decision forest model on those labeled examples. We tuned our model for high precision. Our relation classifier has a precision of 96%. We focus on relational tables in our project as we believe that these tables are most useful for joining with business data tables in spreadsheet scenarios; we plan to include attribute-value tables in the future. The rest of discussion focuses on relational tables.

This component also tries to "understand" the relational tables. Each relational table typically has one column that contains the names of the entities that correspond to the rows of the table [19, 20]. We refer to it as the *entity column* of the table. For example, in the table in Figure 7(c)), the leftmost column is the entity column. It is important to pinpoint the entity column for high-quality search. Consider two entity-attribute

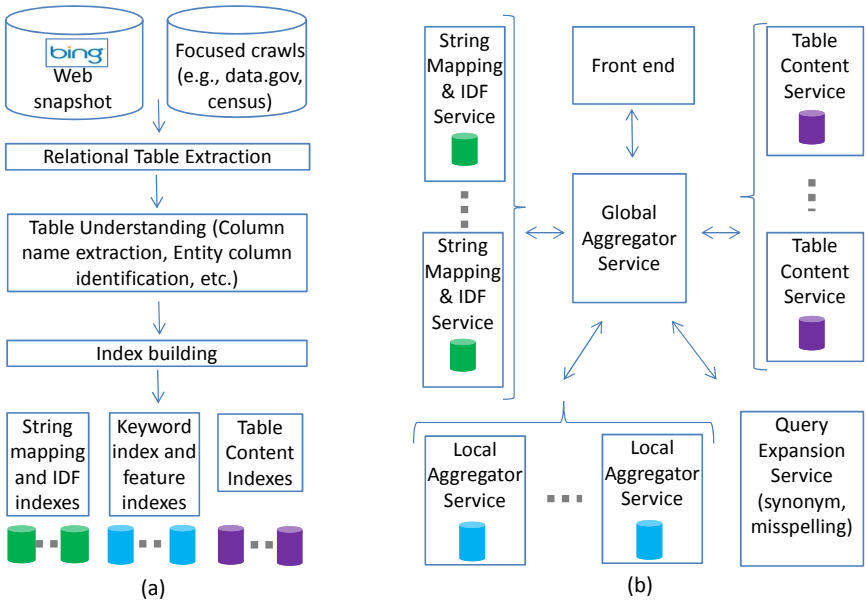


Figure 8: System architectures for (a) Table extraction and understanding and (b) Table search service.

queries: {`aberdeen population 2014`} and {`grays harbor population 2014`}. Both queries "match" the first data row in the table. However, it is a correct match for the first query (as it contains the 2014 population of Aberdeen) but not for the second query (as it does not contain the 2014 population of Grays Harbor). This can be ascertained only if we know the entity column. We identify the entity column using machine learning techniques. Another example of table understanding is column name extraction, especially when the column names are not marked explicitly via `<th>` or `<thead>` tags. Figure 8(a) shows the various subcomponents of this component.

3.3 Web Table Search Service

The web table search service takes a keyword query as input and returns a ranked list of web tables relevant to the query. The architecture as well as the query processing of the web table search service are similar to that of a general web search engine. The architecture is shown in Figure 8(b). The indexed items are web tables instead of web pages. The inverted index over web tables as well indexes containing features about web tables (e.g., number of rows, PageRank) are distributed over many computers, each corresponding to a subset of the web table corpus. Each of those computers runs a local aggregator service that traverses the inverted index and performs top-k ranking among the table subset in that computer. When a query arrives, the global aggregator service distributes the query to the local aggregator services to search simultaneously. It then consolidates all the local top-k results and computes the global top-k. Finally, it invokes the table content services (which stores the full content of each table) to generate query-dependent snippets for the top-k tables.

The key challenge here is to provide a ranking of high quality. Ranking features used in web page ranking or prior work on web table search are not adequate for our purpose [19]. The above features are adequate when the user searches based on the description of the tables; here, we only need to look for keyword matches in the description fields (e.g., page title, table caption) and column names. However, there are some classes of queries which require us to find matches inside the data cells as well as the *alignment* of those matches (e.g., in the same column). One such class is *row-subset queries* where the ideal answer is a *subset of rows in a bigger table* and that subset can be obtained by filtering on the value of an attribute. Consider the query 'largest software companies in usa'. The table in Figure 2(b) contains the largest software companies from all over the world (including several from USA) and is hence relevant. However, 'usa' is not mentioned anywhere in the

page except in the cells of the 'Headquarters' column. To identify the above table as a relevant one, we need to compute a table alignment feature that indicates 'usa' occurs multiple times in different rows of the same (non-entity) column of the table. Prior works do not compute such features and hence fail to return such a table. We incorporate such features in our table search engine.

Another such class is *entity-attribute queries*. Consider a query 'aberdeen population' where the user is looking for a table containing population of Aberdeen. The ideal table should contain 'aberdeen' in a cell in the entity column and 'population' as a column name of a non-entity column. Once again, the above approaches fail to return the relevant table. We incorporate such table alignment features in our service. To compute these features, we need to know the row and column indexes of keyword hits in data cells and column indexes of keyword hits in column names. A document search engine does not store such fine grained information; we design our inverted index to store such information.

3.4 Web Table Answer Service

The web table answer service takes a keyword query (a general web search query) and returns a single table if it the best possible answer (among all possible types of answers as well as top algorithmic search result). Otherwise, it does not return any answer. The main challenge is to have high precision and still having significant coverage.

One of the main challenges is that a perfect match of the query with a table does not guarantee that the table is the best possible answer. Table answer may not be the best type of answer at all; the top algorithmic search result or some other type of answer (e.g., an entity information card or a passage answer) might be better. Consider the query 'washington state climate'. The table with caption "Climate data for Washington State (1895-2015)" in `en.wikipedia.org/wiki/Climate_of_Washington` is a perfect match as Bing returns that page at second position and both the table caption and page title perfectly matches with the query. But a passage answer is a better and more compact answer; both Bing and Google returns a passage answer for this query. We address this challenge by following a two-step approach. In the first step, we determine whether table answer is the *best type* of answer; this is irrespective of whether such a table answer exists or not. If yes, we try to find such a table in the second step. We briefly describe the two steps.

Determine whether table answer is best answer type: We identify several classes of queries for which table answer is the best type of answer. One such class is *list-intent queries*. Such a query seeks for two or more entities. An example is 'largest software companies in usa' as shown in Figure 2(b). Typically a table containing all the desired entities and their important attributes is the best possible answer. Such queries contain the desired entity type in plural; we obtain the list of possible desired entity types from Bing's knowledge graph. For example, the desired entity type in the above query is 'company'. However, this is not a sufficient condition. For example, 'us companies outsourcing' is not a list-intent query although it contains an entity type in plural. We observe that for list-intent queries, the rest of the query, specifically the pre-modifier (the part preceding the desired entity type) and post-modifier (the part following it), either specifies constraints on the entities (to narrow down the desired entities) or a ranking criterion. For example, 'software' and 'in usa' are constraints and 'largest' in the ranking criterion. The constraint can be either an adjective or an entity that is "compatible" with the desired entity type. For example, 'software' is an adjective which is compatible with company while 'usa' is a entity compatible with company. 'Usa' is compatible with company as it is a location entity and companies are often constrained by location. On the other hand, a film or album entity will not be compatible with company as companies are typically not constrained by films or albums. The challenge is to check whether the pre-modifier and post-modifier satisfy the above criteria. We address this challenge in two steps. We first create a dictionary of all adjectives compatible with any entity type; we obtain this from Probase [22]. We also create a dictionary of all entities compatible with any entity type; we first manually curate constraining entity types that are compatible with any query entity type (e.g., location entity type is compatible with company) and then obtain the entities of the former types from Bing's knowledge graph. We also curate a list of ranking keywords. In the first step,

we identify matches of compatible adjectives, compatible entities and ranking keywords in the premodifier and postmodifier. Subsequently, we use a probabilistic graphical model to discover the holistic set of matches that best "covers" the query. If the coverage exceed a certain threshold, we classify the query as list-intent query.

Another class of queries for which table answer is the best type of answer is *superlative queries* (e.g., 'largest software company in usa'). This class is closely related to the class of list-intent queries; we use similar techniques to identify these queries. Under certain conditions, table answer is the best type of answer for *entity-attribute queries* as well; we identify this class of queries as well. The output of this step is a decision whether the query belongs to one of the above classes and, if yes, the parsed query.

Finding table answer: Given a parsed query belonging to one of the above classes, this step tries to find a table answer. We first obtain a set of candidate tables among which we try to find the table answer: these are the tables extracted from the top k ($k \leq 10$) pages returned by Bing for the query. Finding a table answer within the candidate set is challenging. One challenge is that there are multiple tables within the page, and often many of them are similar from the perspective of keyword match. Consider the query "tom cruise movies". There are two tables from `www.movies.com/actors/tom-cruise/tom-cruise-movies/p284917` in the candidate set: one containing the movies that Tom Cruise acted in and the other containing the actors that Tom Cruise has worked with. Both tables have hits for all the keywords. But only the first table is the right answer since the desired entity type is movie. We address this challenge by leveraging the fine-grained understanding of roles of keywords in query such as desired entity type, constraining entity, constraining concept and ranking keyword. In this example, the desired entity type matches with the column name of the entity column of the first table but there is no match with the entity column of the second one. We enrich our ranking features with above roles to make such distinctions.

Another challenge is that the table answer not only needs to be the best table among the candidate tables but also better than the top algorithm result in Bing. For example, for college ranking queries, the top Bing answer is usually the right page from US News. However, sometimes the ranking list in the US News page is not formatted as a table or the table extractor fails to extract the table from the page. We might be able to find a table among the other candidate tables that perfectly matches user's intent but it is from a less well-known source. We do not want to show such a table answer above the proven top result.. We identify such cases by using the click features (click count, click through rate) of both the page containing the candidate table and the most clicked Bing search result for that query.

4 Conclusion

In this paper, we provide a brief overview of the data services we have been developing by leveraging Bing's data assets. Specifically, we described two data services: synonym service and web table service. Both services have been successfully integrated into various Microsoft products and services. One of the key learnings is that commercial applications require data services with very high precision (high nineties), otherwise it would result in user dissatisfaction. While it is usually straightforward to obtain about 80% precision (for both synonym and web table services), it is hard to obtain 95+% precision (while still having good coverage). We need novel approaches to accomplish that level of precision as we described in the paper.

Acknowledgements

We thank Tao Cheng, Dong Xin and Venkatesh Ganti for their core research contributions to the Synonym project. We thank David Simpson, Guihong Cao, Han Wang, Yi Li and Jiang Wu from Bing who collaborated closely with us to ship the web table answers in Bing. We thank Minghui (Jason) Xia from Bing whose team integrated our synonyms data into Bing's entity linking technology. We thank James Finnigan, Konstantin Zoryn, Jean-Sebastien Brunner and Dennis Churin from SQL Server Information Services team who shipped

web table search service in Excel PowerQuery. Finally, we thank Arnd Christian Konig and Vivek Narasayya for their feedback on the initial draft of the paper.

References

[1] Bing Synonyms API. http://datamarket.azure.com/dataset/bing/synonyms, 2012.

[2] Search public data (Power Query) - Excel. https://support.office.com/en-us/article/Search-public-data-Power-Query-12e0d27e-2e91-4471-b422-c20e75c008cd?ui=en-US&rs=en-US&ad=US, 2013.

[3] Ricardo Baeza-Yates, Carlos Hurtado, and Marcelo Mendoza. Query recommendation using query logs in search engines. In *Proceedings of the 2004 international conference on Current Trends in Database Technology*, 2004.

[4] Ricardo Baeza-Yates and Alessandro Tiberi. Extracting semantic relations from query logs. In *Proceedings of KDD*, 2007.

[5] Michael J. Cafarella, Alon Y. Halevy, Daisy Zhe Wang, Eugene Wu, and Yang Zhang. Webtables: exploring the power of tables on the web. *PVLDB*, 1(1):538–549, 2008.

[6] Michael J. Cafarella, Alon Y. Halevy, Yang Zhang, Daisy Zhe Wang, and Eugene Wu. Uncovering the relational web. *WebDB*, 2008.

[7] Kaushik Chakrabarti, Surajit Chaudhuri, Tao Cheng, and Dong Xin. A framework for robust discovery of entity synonyms. In *SIGKDD*, 2012.

[8] Chia-Hui Chang, Mohammed Kayed, Moheb Ramzy Girgis, and Khaled Shaalan. A survey of web information extraction systems. *Transactions on Knowledge and Data Engineering*, 2006.

[9] Surajit Chaudhuri, Manoj Syamala, Tao Cheng, Vivek Narasayya, and Kaushik Chakrabarti. Data services for e-tailers leveraging web search engine assets. In *ICDE*, 2013.

[10] Tao Cheng, Hady W. Lauw, and Stelios Paparizos. Entity synonyms for structured web search. *Transactions on Knowledge and Data Engineering*, 2011.

[11] Eric Crestan and Patrick Pantel. Web-scale knowledge extraction from semi-structured tables. In *WWW*, 2010.

[12] Hang Cui, Ji-Rong Wen, Jian-Yun Nie, and Wei-Ying Ma. Probabilistic query expansion using query logs. In *Proceedings of WWW*, 2002.

[13] J. Friedman. Greedy function approximation: a gradient boosting machine. *Annals of Statistics*, 2001.

[14] Wolfgang Gatterbauer, Paul Bohunsky, Marcus Herzog, Bernhard Krüpl, and Bernhard Pollak. Towards domain-independent information extraction from web tables. In *WWW*, 2007.

[15] Yeye He, Kaushik Chakrabarti, Tao Cheng, and Tomasz Tylenda. Automatic discovery of attribute synonyms using query logs and table corpora. In *Proceedings of WWW*, 2016.

[16] Jianhua Lin. Divergence measures based on the shannon entropy. *Transactions on Information Theory*, 1991.

[17] Sunita Sarawagi. Information extraction. *Foundations and Trends in Databases*, 2008.

[18] Bilyana Taneva, Tao Cheng, Kaushik Chakrabarti, and Yeye He. Mining acronym expansions and their meanings using query click log. In *WWW*, 2013.

[19] Petros Venetis, Alon Y. Halevy, Jayant Madhavan, Marius Pasca, Warren Shen, Fei Wu, Gengxin Miao, and Chung Wu. Recovering semantics of tables on the web. *PVLDB*, 4(9):528–538, 2011.

[20] Jingjing Wang, Haixun Wang, Zhongyuan Wang, and Kenny Q. Zhu. Understanding tables on the web. In *ER*, 2012.

[21] Yalin Wang and Jianying Hu. A machine learning based approach for table detection on the web. In *WWW*, 2002.

[22] Wentao Wu, Hongsong Li, Haixun Wang, and Kenny Q. Zhu. Probase: A probabilistic taxonomy for text understanding. In *SIGMOD*, 2012.

Attributed Community Analysis: Global and Ego-centric Views

Xin Huang[†], Hong Cheng[‡], Jeffrey Xu Yu[‡]
[†] *University of British Columbia,* [‡]*The Chinese University of Hong Kong*
xin0@cs.ubc.ca, {hcheng,yu}@se.cuhk.edu.hk

Abstract

The proliferation of rich information available for real world entities and their relationships gives rise to a type of graph, namely attributed graph, where graph vertices are associated with a number of attributes. The set of an attribute can be formed by a series of keywords. In attributed graphs, it is practically useful to discover communities of densely connected components with homogeneous attribute values. In terms of different aspects, the community analysis tasks can be categorized into global network-wide and ego-centric personalized. The global network-wide community analysis considers the entire network, such that community detection, which is to find all communities in a network. On the other hand, the ego-centric personalized community analysis focuses on the local neighborhood subgraph of given query nodes, such that community search. Given a set of query nodes and attributes, community search in attributed graphs is to locally detect meaningful community containing query-related nodes in the online manner. In this work, we briefly survey several state-of-the-art community models based on various dense subgraphs, meanwhile also investigate social circles, that one special kind of communities are formed by friends in 1-hop neighborhood network for a particular user.

1 Introduction

Nowadays with rich information available for real world entities and their relationships, graphs can be built in which vertices are associated with a set of attributes describing the properties of the vertices. The attributed graphs exist in many application domains such as web, social networks, collaboration networks, biological networks and communication networks and so on. Community(cluster), as a group of densely inter-connected nodes sharing similar properties, naturally exists in real-world networks [24]. In this work, we investigate communities in two aspects of global network-wide and ego-centric personalized. From the global network-wide analysis, we study the task of community detection that is to identify all communities in a network [13, 17, 23]. On the other hand, in the ego-centric personalized community analysis, we studied the problem of community search that is to find meaningful communities containing query-related nodes in local subgraph. Since the communities defined by different nodes in a network may be quite different, community search with query nodes opens up the prospects of user-centered and personalized search, with the potential of the answers being more meaningful to a user[9]. Recently, several papers [19, 9, 11, 22, 15, 5, 4, 1] have studied community search on graph structure for ego-centric personalized community analysis.

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

In Section 2, we focus on community detection in attributed graphs. For discovering all communities in attributed graph, [24, 25, 2] model the problem as graph clustering, which aims to partition the graph into several densely connected components with homogeneous attribute values. We proposed a novel graph clustering algorithm, SA-Cluster, which combines structural and attribute similarities through a unified distance measure. SA-Cluster finds all clusters by considering the full attribute space. However, in high-dimensional attributed graphs[10], the high-dimensional clusters are hard to interpret, or there is even no significant cluster with homogeneous attribute values in the full attribute space. If an attributed graph is projected to different attribute subspaces, various interesting clusters embedded in subspaces can be discovered. Therefore, based on the unified distance measure, we extend the method of SA-Cluster to propose a novel cell-based algorithm SCMAG to discover clusters embedded in subspaces, with similar attribute values and cohesive structure[10].

In Section 3, we focus on community search in attributed graphs. Unlike community detection, community search focus on the local neighborhood of given query-related nodes. Given a set of query nodes and attributes, community search on attribute graph is to detect a densely inter-connected communities containing all required query nodes and attributes in the online manner. First, we introduce one of best known query applications on attribute graph as team formation [12, 14, 6]. Team formation is to find a group of individuals satisfying all skilled required in a task with low communication cost. Then we show how to generalize the problem of team formation into community search. Next, we briefly summarize several community models based on various dense subgraphs, such as quasi-clique[4], densest subgraph[22], k-core[19, 15, 5, 1] and k-truss[9, 11]. Finally, we investigate social circles, and analyze its power in social contagion. In social network, for a particular user, social circles are defined as communities in her 1-hop neighborhood network, a network of connections between her friends. The structure of social circles can be modeled as connected component, k-core and k-truss. [20] shows the probability of contagion in social contagion process is tightly controlled by the number of social circles.

2 Community Detection on Attributed Graphs

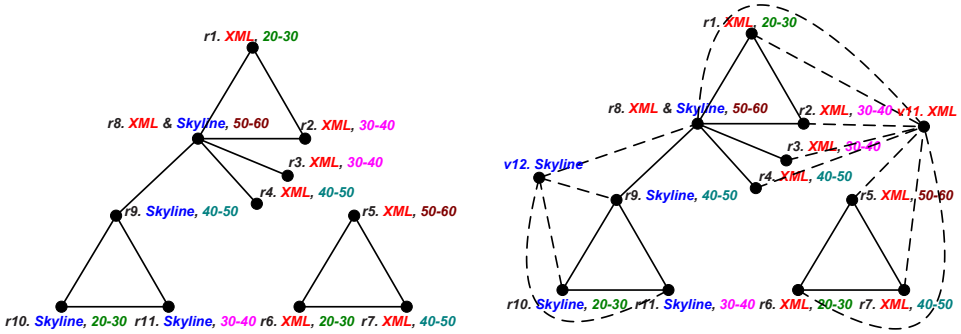
In this section, we study the community detection on attributed graphs, under the semantics of both full attribute space and attribute subspace. We first formulate the problem of graph clustering on attributed graphs by considering both structural connectivity and attribute similarities. Then, we design a unified distance measure to combine structural and attribute similarities. Finally, we briefly review the key ideas of community detection algorithms, as SA-Cluster for graph clustering on full space attributes [24] and SCMAG for graph subspace clustering[10].

2.1 Attributed Graphs

An undirected, unweighted simple graph is represented as $G = (V, E)$ with $|V|$ vertices and $|E|$ edges. When the vertices are associated with attributes, the network structure can be modeled as a new type of attributed graph as follow.

Definition 1 (Attributed Graph): An attributed graph is denoted as $G = (V, E, \Lambda)$, where V is the set of vertices, E is the set of edges, and $\Lambda = \{a_1, \ldots, a_m\}$ is the set of attributes associated with vertices in V for describing vertex properties. A vertex $v \in V$ is associated with an attribute vector $[a_1(v), \ldots, a_m(v)]$ where $a_j(v)$ is a set of attribute values of vertex v on attribute a_j.

Figure 1 shows an example of a coauthor graph where a vertex represents an author and an edge represents the coauthor relationship between two authors. In addition, there are an author ID, research topic and age range associated with each author, which are considered as attributes to describe the vertex properties. For example, the author r_8 works on two topics of XML and Skyline. The problem of community detection is to find all



(a) Attributed Graph (b) Attribute Augmented Graph

Figure 1: A Coauthor Network with Two Attributes "Research Topic" and "Age Range"

communities on the attributed graph, such as the example in Figure 1(a), based on both structural and attribute similarities. Therefore, we formulate the problem as the graph clustering on attributed graph in the following.
Attributed graph clustering is to partition an attributed graph G into k disjoint subgraphs $\{G_i = (V_i, E_i, \Lambda)\}_{i=1}^{k}$, where $V = \bigcup_{i=1}^{k} V_i$ and $V_i \bigcap V_j = \emptyset$ for any $i \neq j$. A desired clustering of an attributed graph should achieve a good balance between the following two objectives: (1) vertices within one cluster are close to each other in terms of structure, while vertices between clusters are distant from each other; and (2) vertices within one cluster have similar attribute values, while vertices between clusters could have quite different attribute values.

2.2 Attribute Augmented Graph

In the following, we used an *attribute augmented graph* to represent attributes explicitly as *attribute vertices and edges* proposed by [24].

Definition 2 (Attribute Augmented Graph): Given an attributed graph $G = (V, E, \Lambda)$ with a set of attributes $\Lambda = \{a_1, \ldots, a_m\}$. The domain of attribute a_i is $Dom(a_i) = \{a_{i1}, \ldots, a_{in_i}\}$ with a size of $|Dom(a_i)| = n_i$. An attribute augmented graph is denoted as $G_a = (V \cup V_a, E \cup E_a)$ where $V_a = \{v_{ij}\}_{i=1,j=1}^{m,\ n_i}$ is the set of attribute vertices and $E_a \subseteq V \times V_a$ is the set of attribute edges. An attribute vertex $v_{ij} \in V_a$ represents that attribute a_i takes the j^{th} value. An attribute edge $(v_i, v_{jk}) \in E_a$ iff $a_{jk} \in a_j(v_i)$, i.e., vertex v_i takes the value of a_{jk} on attribute a_j. Accordingly, a vertex $v \in V$ is called a structure vertex and an edge $(v_i, v_j) \in E$ is called a structure edge.

Figure 1(b) is an attribute augmented graph on the coauthor network example. Two attribute vertices v_{11} and v_{12} representing the topics "XML" and "Skyline" are added. Authors with corresponding topics are connected to the two vertices respectively in dashed lines. We omit the attribute vertices and edges corresponding to the age attribute, for the sake of clear presentation. In the attributed graph clustering problem, we need to discuss two main issues: (1) a distance measure, and (2) a clustering algorithm below.

2.3 A Unified Random Walk Distance

We use the neighborhood random walk model on the attribute augmented graph G_a to compute a unified distance between vertices in V. The random walk distance between two vertices $v_i, v_j \in V$ is based on the paths consisting of both structure and attribute edges. Thus it effectively combines the structural proximity and attribute similarity of two vertices into one unified measure. The transition probability matrix P_A on G_a is defined as follows.

A structure edge $(v_i, v_j) \in E$ is of a different type from an attribute edge $(v_i, v_{jk}) \in E_a$. The m attributes in Λ may also have different importance. Therefore, they may have different degree of contributions in random walk distance. Without loss of generality, we assume that a structure edge has a weight of ω_0, attribute edges corresponding to a_1, a_2, ..., a_m have an edge weight of ω_1, ω_2, ..., ω_m, respectively. In the following, we will define the transition probabilities between two structure vertices, between a structure vertex and an attribute vertex, and between two attribute vertices. First, the transition probability from a structure vertex v_i to another structure vertex v_j through a structure edge is

$$p_{v_i,v_j} = \begin{cases} \dfrac{\omega_0}{|N(v_i)| * \omega_0 + \omega_1 + \ldots + \omega_m}, & if (v_i, v_j) \in E \\ 0, & otherwise \end{cases} \tag{1}$$

where $N(v_i)$ represents the set of structure vertices connected to v_i.

The transition probability from a structure vertex v_i to an attribute vertex v_{jk} through an attribute edge is

$$p_{v_i,v_{jk}} = \begin{cases} \dfrac{\omega_j}{|N(v_i)| * \omega_0 + \omega_1 + \ldots + \omega_m}, & if (v_i, v_{jk}) \in E_a \\ 0, & otherwise \end{cases} \tag{2}$$

The transition probability from an attribute vertex v_{ik} to a structure vertex v_j through an attribute edge is

$$p_{v_{ik},v_j} = \begin{cases} \dfrac{1}{|N(v_{ik})|}, & if (v_{ik}, v_j) \in E_a \\ 0, & otherwise \end{cases} \tag{3}$$

The transition probability between two attribute vertices v_{ip} and v_{jq} is 0 as there is no edge between attribute vertices.

$$p_{v_{ip},v_{jq}} = 0, \forall v_{ip}, v_{jq} \in V_a \tag{4}$$

The transition probability matrix P_A is a $|V \cup V_a| \times |V \cup V_a|$ matrix, where the first $|V|$ rows and columns correspond to the structure vertices and the rest $|V_a|$ rows and columns correspond to the attribute vertices. For the ease of presentation, P_A is represented as

$$P_A = \begin{bmatrix} P_{V_1} & A_1 \\ B_1 & O \end{bmatrix} \tag{5}$$

where P_{V_1} is a $|V| \times |V|$ matrix representing the transition probabilities defined by Equation (1); A_1 is a $|V| \times |V_a|$ matrix representing the transition probabilities defined by Equation (2); B_1 is a $|V_a| \times |V|$ matrix representing the transition probabilities defined by Equation (3); and O is a $|V_a| \times |V_a|$ zero matrix.

Definition 3 (Random Walk Distance Matrix): Let P_A be the transition probability matrix of an attribute augmented graph G_a. Given L as the length that a random walk can go, $c \in (0, 1)$ as the random walk restart probability, the unified neighborhood random walk distance matrix R_A is

$$R_A = \sum_{l=1}^{L} c(1-c)^l P_A^l \tag{6}$$

2.4 SA-Cluster Algorithm

SA-Cluster adopts the *K-Medoids* clustering framework. After initializing the cluster centroids and calculating the random walk distance at the beginning of the clustering process, it repeats the following four steps until convergence.

1. Assign vertices to their closest centroids;
2. Update cluster centroids;
3. Adjust attribute edge weights $\{\omega_1, \ldots, \omega_m\}$;
4. Re-calculate the random walk distance matrix R_A.

Different from traditional K-Medoids, SA-Cluster has two additional steps (i.e., steps 3-4): in each iteration, the attribute edge weights $\{\omega_1, \ldots, \omega_m\}$ are automatically adjusted to reflect the clustering tendencies of different attributes. Interested readers can refer to [24] for the proposed mechanism for weight adjustment.

The time complexity of SA-Cluster is O($t \cdot L \cdot |V \cup V_a|^3$), where t is the number of iterations in the clustering process, and $O(L \cdot |V \cup V_a|^3)$ is the cost of computing the random walk distance matrix R_A. In order to improve the efficiency and scalability of SA-Cluster, [25] proposes an efficient algorithm Inc-Cluster to incrementally update the random walk distances given the edge weight increments. Complexity analysis shows that Inc-Cluster can improve SA-Cluster by approximately t times. For further speed up Inc-Cluster, [2] designs parallel matrix computation techniques on a multicore architecture.

2.5 Subspace Clustering in High-dimensional Attributed Graphs

Although SA-Cluster can differentiates the importance of attributes with an attribute weighting strategy, it cannot get rid of irrelevant attributes completely, especially when the dimension of attribute is high, i.e., $|\Lambda| = m$ is large. The high-dimensional clusters are hard to interpret, or there is even no significant cluster with homogeneous attribute values in the full attribute space. If an attributed graph is projected to different attribute subspaces, various interesting clusters embedded in subspaces can be discovered which, however, may not exhibit in the full attribute space. In the following, we will study the problem of subspace clustering in high-dimensional attributed graphs. We first define the subspace criterion of good subspace clusters in terms of homogeneous properties and cohesive structure. Then, we propose a novel cell-based subspace clustering algorithm SCMAG.

2.5.1 Criterion of Subspace Clusters

For the discovered clusters embedded in subspaces, should not only have homogeneous attribute values, but also have dense connections, i.e., correspond to communities with homogeneous properties and cohesive structure.
Attribute Criterion. Given a attribute subspace $\mathcal{S} \subseteq \Lambda$, the subspace entropy and interest are defined as follows.

Definition 4 (Subspace Entropy): Given a set of attributes $\mathcal{S} = \{a_1, \ldots, a_k\} \subseteq \Lambda$, the subspace entropy of $\mathcal{S}$ is defined as

$$H(a_1, \ldots, a_k) = -\sum_{A_1 \in Dom(a_1)} \cdots \sum_{A_k \in Dom(a_k)} p(A_1, \ldots, A_k) \log p(A_1, \ldots, A_k) \tag{7}$$

where $p(A_1, \ldots, A_k)$ is the percentage of graph vertices whose attribute value vector is $[A_1, \ldots, A_k]$.

In addition, we want the attributes of a subspace to be correlated. If the attributes are independent of each other, the subspace does not give more information than looking at each attribute independently. We measure the correlation of a subspace $\mathcal{S}$ using mutual information between all individual dimensions of the subspace as below.

$$I(\{a_1, \ldots, a_k\}) = \sum_{i=1}^{k} H(a_i) - H(a_1, \ldots, a_k)$$

We consider a subspace $\mathcal{S} = \{a_1, \ldots, a_k\}$ as an interesting subspace, if $\mathcal{S}$ is more strongly correlated than any of its subsets $\mathcal{S}' \subseteq \mathcal{S}$. To measure the increase in correlation of a subspace, we define the *interest* of a subspace.

Definition 5 (Subspace Interest): Given a set of attributes $\mathcal{S} = \{a_1, \ldots, a_k\} \subseteq \Lambda$, the subspace interest of $\mathcal{S}$ is defined as the minimum increase in correlation of $\mathcal{S}$ over its $(k-1)$-dimensional subsets.

$$interest(a_1, \ldots, a_k) = I(\{a_1, \ldots, a_k\}) - \max_i I(\{a_1, \ldots, a_k\} - \{a_i\})$$

Therefore, a good subspace for clustering should have low subspace entropy and high subspace interest.
Structural Criterion. Given a subspace $\mathcal{S} = \{a_1, \ldots, a_k\}$ and each attribute a_i has n_i values, the k-dimensional space is partitioned to form a grid. The vertices with same attribute vector fall into the same cell of grid, under this k-dimensional space. Thus, for a good space for clustering, we identify the cells with high coverage and connectivity, according to the following definition.

Definition 6 (Coverage and Connectivity): Given a cell u in a subspace, the coverage of u is measured by the number of vertices in u, i.e., $V(u) = |u|$. The connectivity of u is measured by the sum of random walk scores of all pairs of vertices, divided by the cell size

$$D(u) = \frac{\sum_{v_i, v_j \in u} \widetilde{Q}_{VV}(v_i, v_j)}{|u|},$$

where $\widetilde{Q}_{VV}(v_i, v_j)$ is the normalized structural similarity between v_i and v_j.

2.5.2 A review of SCMAG

Based on the criteria for interesting subspace with good clustering tendency and coverage subspace with dense connectivity, the cell-based algorithmic framework of SCMAG is described as follow. We will first find the subspaces with good clustering tendency, and then identify cells in the subspace with high coverage and high connectivity. Adjacent qualified cells will be merged to form a maximal cluster in the subspace.

Follow by the framework of SA-Cluster, we first construct the attribute augmented graph by Definition 2. Then, we use the random walk with restart to unify the structural closeness and attribute similarity into a single measure. Based on the random walk score, we design a novel cell combining strategy on dimensions of attributes. Moreover, to distinguish the multi-values in an attribute, we choose one attribute value with the largest attribute similarity between the value and vertex as the unique one. Thus, each vertex is associated with an attribute vector containing a single value in each attribute. Finally, we iteratively find subspace with low subspace entropy and high subspace interest, and detect clusters by merging adjacent dense cells to satisfy high coverage and dense connectivity. The entire procedure is shown as follow.

1. Construct the attribute augmented graph, and calculate the random walk distance;

2. Identify similar attribute values to be adjacent;

Table 1: Clusters in attribute subspace {Citation, H-index, G-index, Venue} on bibliographic graph, where each vertex represents an author and an edge represents the author collaboration. Each author has 12 attributes, such as Topic, Citation, H-index, Sociability and so on[10].

Cluster 1 Database	**Cluster 2 Software Engineering & Scientific Computing**	**Cluster 3 Hardware & Architecture**	**Cluster 4 Algorithms & Theory**
Rakesh Agrawal	C.A.R. Hoare	A. L. Sangiovanni-Vincentelli	Rajeev Motwani
Hector Garcia-Molina	Leslie Lamport	Sharad Malik	Robert E. Tarjan
Jeffrey D. Ullman	Thomas A. Henzinger	Sartaj K. Sahni	Christos Papadimitriou
Jennifer Widom	Rajeev Alur	Lothar Thiele	Prabhakar Raghavan
Christos Faloutsos	David Harel	Sudhakar M. Reddy	David R. Karger
Jim Gray	Joseph Halpern	Jason Cong	Richard M. Karp
David J. DeWitt	Amir Pnueli	Robert Brayton	Jon M. Kleinberg
Michael Stonebraker	Moshe Vardi	Miodrag Potkonjak	Leslie Valiant
Ramakrishnan Srikant	Edmund Clarke	Massoud Pedram	Oded Goldreich
Serge Abiteboul	Robin Milner	Janak H. Patel	Moni Naor

3. Assign vertices into cells of the grid by handling multi-valued attributes;
4. Find good subspaces with low subspace entropy and high subspace interest;
5. Find clusters in the identified subspace by merging adjacent dense cells to satisfy high coverage and dense connectivity;

Case study. Table 1 shows that SCMAG discovers 4 clusters from different research fields on bibliographic graph in the subspace {*Citation, H-index, G-index, Venue*}, and list 10 representative authors in each cluster. The subspace combination of *Citation, H-index* and *G-index* is interesting, as these three attributes are positively correlated – H-index and G-index are computed from citations.

3 Community Search on Attributed Graphs

Given a set of query nodes and attributes, community search on attributed graphs is to detect meaningful community containing query nodes and satisfying attribute constraints in the online manner. As an ego-centric personalized analysis, community search is different from community detection, which focuses on the local neighborhood subgraph of query-related nodes. In the following, we first introduce one of best known query application on attributed graphs as team formation, and show how to generalize it into community search on attributed graphs. Then, we will discuss several state-of-the-art community models based on various dense subgraphs, including special community models of social circles.

3.1 Team Formation

Task-driven Team formation [14]. Assume that in attributed graph $G(V, E, \Lambda)$, each vertex is associated with different skill attributes. Given a task T that requires a set of skills, the problem of team formation is to find a group of individuals $X \subseteq V$ who can function as a team to accomplish task T, such that every required skill in T is exhibited by at least one individual in X. Additionally, the members of team X should define a subgraph or a tree in G with low communication cost. The communication cost measures how effectively the team members can collaborate: the lower the communication cost, the better the quality of the team. [14] measures team

communication cost in terms of diameter or spanning tree. We formulate the problem of diameter based team formation as below.

Definition 7 (Graph Diameter): The diameter of a graph G is defined as the maximum length of a shortest path in G, i.e., $\mathsf{diam}(G) = \max_{u,v \in G}\{\mathsf{dist}_G(u, v)\}$, where $\mathsf{dist}_G(u, v)$ is the length of a shortest path between u and v in G.

Definition 8 (Diameter based Team Formation): Given an attributed graph $G(V, E, \Lambda)$ and a task $T = \{w_1, ..., w_k\} \subseteq \bigcup_{a \in \Lambda} Dom(a)$, find a subgraph $H \subseteq G$ such that satisfies

1. $\forall w \in T$, $\exists v \in H$ and $a \in \Lambda, s.t, w \in a(v)$;
2. $\mathsf{diam}(H)$ is minimized.

This problem has been shown to be NP-complete. However, there exists a 2-approximation algorithm, which can find a subgraph H that satisfies all required skills and has the diameter no greater than 2 times of the optimal one.

3.2 A Formulation of Community Search

In the diameter based team formation, the diameter metric may not measure the communication cost well, because this simple function is instability: a slight change in the graph may result in a radical change in the solution, due to the weak connectivity[6]. Therefore, to enforce the dense connectivity constraints on the formed team is necessary. On the other hand, in some application scenarios, we may need to specify leaders in a team, since leaders need to iteratively communicate with each team member to monitor and coordinate the project[12]. Thus, the given leaders(vertices) must be contained in the reported team. As a result, we can generalize team formation with leader constraints into the problem of diverse attributed community search on attributed graph as follow.

Definition 9 (Diverse Attributed Community Search): Given an attributed graph $G(V, E, \Lambda)$, a set of attribute values $T = \{w_1, ..., w_k\} \subseteq \bigcup_{a \in \Lambda} Dom(a)$ and a set of query nodes $Q \subseteq V(G)$, find a connected subgraph $H \subseteq G$ such that satisfies

1. $Q \subseteq V(H)$;
2. $\forall w \in T$, $\exists v \in H$ and $a \in \Lambda$, $s.t, w \in a(v)$;
3. H is densely connected, and the communication cost is minimum.

As we can see, the problem of diverse attributed community search tends to find a densely connected subgraph containing all query nodes and achieving the coverage of diverse attributes, with the minimum communication cost. In Definition 9, either a set of attributes T or a set of query nodes Q can be empty. If the set of attributes are empty as $T = \emptyset$, the problem of community search on attributed graph is equivalent to the problem of find densely connected community in a simple graph $G(V, E)$. If the set of query nodes are empty as $Q = \emptyset$, the problem of community search on attributed graph is equivalent to the problem of team formation without leader constraints in Definition 8.

3.3 Dense Subgraph based Community Models

In this section, we will introduce several novel community models in a simple graph $G(V, E)$ without attributes. These state-of-the-art community models are based on different dense subgraph definitions, such as quasi-clique[4], densest subgraph[22], k-core[19, 15, 5, 1] and k-truss[9, 11]. These community models can be further developed and extended for applying on attributed graph $G(V, E, \Lambda)$.

Quasi-Clique Community. Cui et. al[4] propose a α-adjacency-γ-quasi-k-clique community model. A γ-quasi-k-clique of a simple graph G is defined as a k-node subgraph of G with at least $\lfloor \gamma \frac{k(k-1)}{2} \rfloor$ edges, where $0 \leq \gamma \leq 1$. A γ-quasi-k-clique is a relaxation of a k-clique. Two γ-quasi-k-cliques are α-adjacent and can be union if they share at least α common vertices. Given a query node, the community search problem is to find all α-adjacency-γ-quasi-k-cliques containing it. Several heuristic approaches are proposed for speed up the NP-hard query processing.

Query-biased Densest Subgraph Community. Wu et al. [22] studied the query biased densest connected subgraph (QDC) problem for avoiding subgraphs irrelevant to query nodes in the discovered community. The community is defined based on a connected graph containing given query nodes, and it optimizes a fundamentally different function called query biased edge density, which is calculated as the overall edge weight averaged over the weight of nodes in a community.

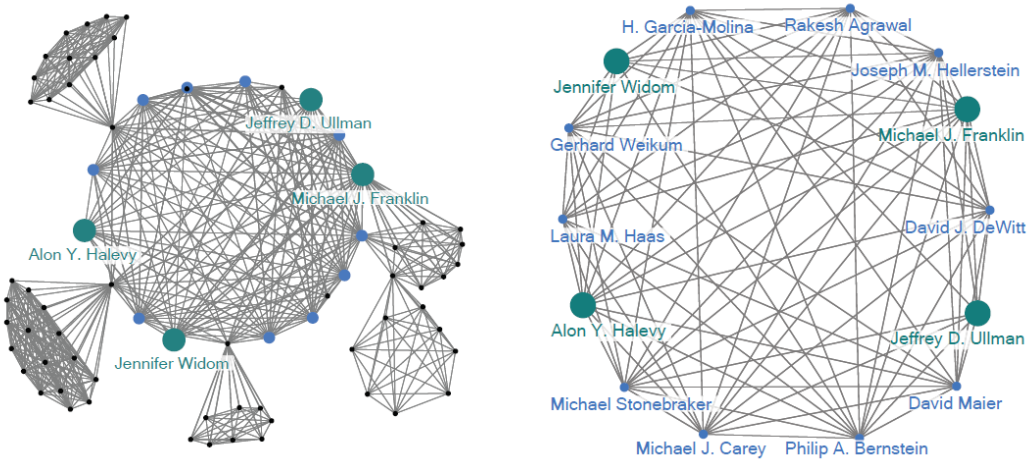
K-core Community. Several community models build up on the structure of k-core [19, 15, 5, 1]. A k-core is a subgraph of G that requires each node has at least k neighbors within this subgraph [18]. Sozio et al. [19] proposed a k-core based community model, called Cocktail Party, with the distance and size constraints. Cocktail Party community model finds the k-core with largest k as the density optimization, and uses the furthest query distance as the communication cost function. Cui et al. [5] find a k-core community for a query node using local search. In addition, Li et al. [15] propose influential community model that finds top-r communities with the highest influence scores over the entire graph, without considering query nodes.

K-truss Community. A k-truss is a subgraph of G that requires each edge be contained at least (k-2) triangles within this subgraph [3]. In a social network, a triangle indicates two friends have a common friend, which shows a strong relationship among three friends. Intuitively, the more common friends two people have, the stronger their relationship. In a k-truss, each pair of friends is "endorsed" by at least (k-2) common friends[11]. Thus, a k-truss with a large value of k signifies strong inner-connections between members of the subgraph. The community proposed by [9] and [11] both are build upon the connected k-truss. [9] proposes a k-truss community model based on triangle adjacency, to find all overlapping communities of one query node. The closest truss community [11] aims to find a connected k-truss subgraph with the largest k that contains Q, and has the minimum diameter among such subgraphs. Here, the minimum graph diameter is used as the communication cost constraint. In comparison of the k-core community and the k-truss community, conceptually, k-truss is a more cohesive definition than k-core, as k-truss is based on triangles whereas k-core simply considers node degree.

Case study. Figure 2(b) shows a closest truss community [11] detected on DBLP network using the query $Q = \{$"Alon Y. Halevy", "Michael J. Franklin", "Jeffrey D. Ullman", "Jennifer Widom"$\}$ and $T = \emptyset$. It has 14 authors, 81 edges and the edge density of 0.89. The community does not include any authors in a 9-truss [3] that are far away from and loosely connected with queried authors in Figure 2(a), which shows the superiority of closest truss community.

3.4 Social Circles and Social Contagion

In this section, we will study one special kind of community in social networks as social circles. For one query user, social circles are communities in query users 1-hop neighborhood network, a network of connections between her friends. Simply, in terms of graph structure, for a user with a small number of friends, a connected component is strongly enough to represent a social circle; Whereas, for a user with a large number of friends,



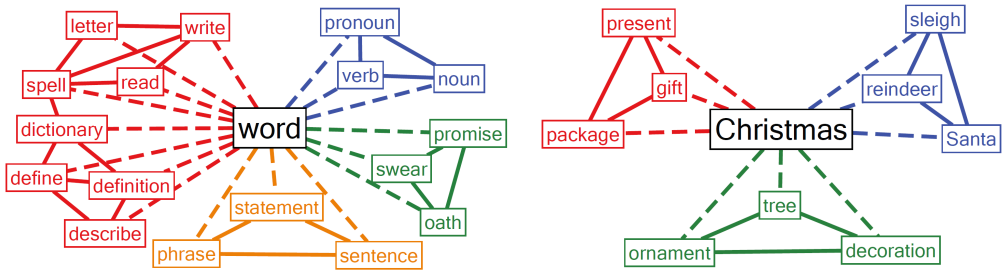
(a) 9-truss [3]

(b) closest truss community [11]

Figure 2: Community search on DBLP network without attributes using query $Q = \{$"Alon Y. Halevy", "Michael J. Franklin", "Jeffrey D. Ullman", "Jennifer Widom"$\}$ and $T = \emptyset$

since the structure of her neighborhood network becomes complex, a connected k-core and a connected k-truss as cohesive structure are much better social circle models. Interested readers can refer to more community models on attribute graphs [16, 21].

In the following, we will show how these social circles affects the process of information diffusion on social contagion. Ugander et al. [20] study two social contagion processes in Facebook: the process that a user joins Facebook in response to an invitation email from an existing Facebook user, and the process that a user becomes an engaged user after joining. They find that the probability of contagion is tightly controlled by the number of social circles in a users neighborhood, rather than by the number of friends in the neighborhood. A social circle represents a distinct social context of a user, and the multiplicity of social contexts is termed structural diversity [20]. A user is much more likely to join Facebook and become engaged if he or she has a larger structural diversity, i.e., a larger number of distinct social contexts. [7, 8] studied the problem of find k users with the highest structural diversity in graphs, which can be beneficial to a wide range of application domains, for example, political campaign, the promotion of health practices, marketing, and so on.



(a) Word

(b) Christmas

Figure 3: Top-2 structural diversity based on connected 2-core in word association network. Here "word" and "Christmas" respectively has the top-2 highest structural diversity score as 4 and 3.

Case study. Figure 3 shows that top-2 results in the word association network using connected 2-core as the structural bone of social circles. Two words "word" and "Christmas" have the highest two structural diversity scores of 4 and 3. As we can see, in Figure 3, each vertex in the 2-core component has at least two neighbor words. Specifically, the word "word" in Figure 3 (a) has 4 distinct contexts of associated words with different meanings. For example, {"swear", "oath", "promise"} represent the synonym of "words" as "promise", and {"verb", "noun", "pronoun"} are different types of "word". The word "Christmas" has three distinct contexts of associated words, as shown in Figure 3 (b), {"reindeer", "sleigh", "Santa"} describe the "Santa", {"present", "gift", "package"} represent the "Christmas gifts" and {"tree", "ornament", "decoration"} are related to the "Christmas tree".

4 Future Work and Conclusion

In this paper, we study two problems of community detection and community search in attributed networks, respectively in terms of global network-wide analysis and ego-centric personalized analysis aspects. For community detection, we design a unified distance measure to combine structural and attribute similarities on attribute graphs. Based on that, we propose two community detection algorithms SA-Cluster and SCMAG for respectively considering the full space and subspace of attributes. For community search, we give a formal problem definition of community search on attributed graphs by generalizing from the problem of team formation. Several dense subgraph based community model are surveyed here for a comparison. Since all these dense subgraph based community models only consider structures in simple graphs without attributes, it would be interesting to extend the models and algorithms to attributed graphs for community search. Given the recent surge of interest k-core and k-truss in probabilistic graphs, an exciting question is how k-core and k-truss models generalizes to probabilistic graphs. The challenge is to develop extensions that are widely useful and tractable.

References

[1] N. Barbieri, F. Bonchi, E. Galimberti, and F. Gullo. Efficient and effective community search. *Data Mining and Knowledge Discovery*, 29(5):1406–1433, 2015.

[2] H. Cheng, Y. Zhou, X. Huang, and J. X. Yu. Clustering large attributed information networks: an efficient incremental computing approach. *Data Mining and Knowledge Discovery*, 25(3):450–477, 2012.

[3] J. Cohen. Trusses: Cohesive subgraphs for social network analysis. Technical report, National Security Agency, 2008.

[4] W. Cui, Y. Xiao, H. Wang, Y. Lu, and W. Wang. Online search of overlapping communities. In *SIGMOD*, pages 277–288, 2013.

[5] W. Cui, Y. Xiao, H. Wang, and W. Wang. Local search of communities in large graphs. In *SIGMOD*, pages 991–1002, 2014.

[6] A. Gajewar and A. D. Sarma. Multi-skill collaborative teams based on densest subgraphs. In *SDM*, pages 165–176. SIAM, 2012.

[7] X. Huang, H. Cheng, R.-H. Li, L. Qin, and J. X. Yu. Top-k structural diversity search in large networks. *PVLDB*, 6(13):1618–1629, 2013.

[8] X. Huang, H. Cheng, R.-H. Li, L. Qin, and J. X. Yu. Top-k structural diversity search in large networks. *The VLDB Journal*, 24(3):319–343, 2015.

[9] X. Huang, H. Cheng, L. Qin, W. Tian, and J. X. Yu. Querying k-truss community in large and dynamic graphs. In *SIGMOD*, pages 1311–1322, 2014.

[10] X. Huang, H. Cheng, and J. X. Yu. Dense community detection in multi-valued attributed networks. *Information Sciences*, 314:77–99, 2015.

[11] X. Huang, L. V. Lakshmanan, J. X. Yu, and H. Cheng. Approximate closest community search in networks. *PVLDB*, 9(4):276–287, 2015.

[12] M. Kargar and A. An. Discovering top-k teams of experts with/without a leader in social networks. In *Proceedings of the 20th ACM international conference on Information and knowledge management*, pages 985–994. ACM, 2011.

[13] A. Lancichinetti and S. Fortunato. Community detection algorithms: a comparative analysis. *Physical review E*, 80(5):056117, 2009.

[14] T. Lappas, K. Liu, and E. Terzi. Finding a team of experts in social networks. In *KDD*, pages 467–476. ACM, 2009.

[15] R.-H. Li, L. Qin, J. X. Yu, and R. Mao. Influential community search in large networks. *PVLDB*, 8(5), 2015.

[16] J. J. McAuley and J. Leskovec. Learning to discover social circles in ego networks. In *NIPS*, pages 548–556, 2012.

[17] M. E. Newman. Fast algorithm for detecting community structure in networks. *Physical review E*, 69(6):066133, 2004.

[18] S. B. Seidman. Network structure and minimum degree. *Social networks*, 5(3):269–287, 1983.

[19] M. Sozio and A. Gionis. The community-search problem and how to plan a successful cocktail party. In *KDD*, pages 939–948, 2010.

[20] J. Ugander, L. Backstrom, C. Marlow, and J. Kleinberg. Structural diversity in social contagion. *Proceedings of the National Academy of Sciences*, 109(16):5962–5966, 2012.

[21] Y. Wang and L. Gao. An edge-based clustering algorithm to detect social circles in ego networks. *Journal of computers*, 8(10):2575–2582, 2013.

[22] Y. Wu, R. Jin, J. Li, and X. Zhang. Robust local community detection: On free rider effect and its elimination. *PVLDB*, 8(7), 2015.

[23] J. Yang and J. Leskovec. Overlapping community detection at scale: a nonnegative matrix factorization approach. In *WSDM*, pages 587–596, 2013.

[24] Y. Zhou, H. Cheng, and J. X. Yu. Graph clustering based on structural/attribute similarities. *PVLDB*, 2(1):718–729, 2009.

[25] Y. Zhou, H. Cheng, and J. X. Yu. Clustering large attributed graphs: An efficient incremental approach. In *ICDM*, pages 689–698, 2010.

Ten Years of Knowledge Harvesting: Lessons and Challenges

Gerhard Weikum[1], Johannes Hoffart[2], Fabian Suchanek[3]

[1] Max Planck Institute for Informatics, Saarbrücken, Germany
[2] Ambiverse GmbH, Saarbrücken, Germany
[3] Télécom ParisTech University, Paris, France
E-mail: weikum@mpi-inf.mpg.de, johannes@ambiverse.com, suchanek@telecom-paristech.fr

Abstract

This article is a retrospective on the theme of knowledge harvesting: automatically constructing large high-quality knowledge bases from Internet sources. We draw on our experience in the Yago-Naga project over the last decade, but consider other projects as well. The article discusses lessons learned on the architecture of a knowledge harvesting system, and points out open challenges and research opportunities.

1 Large High-Quality Knowledge Bases

Turning Internet content, with its wealth of latent-value but noisy text and data sources, into crisp "machine knowledge" that can power intelligent applications is a long-standing goal of computer science. Over the last ten years, *knowledge harvesting* has made tremendous progress, leveraging advances in scalable information extraction and the availability of curated knowledge-sharing sources such as Wikipedia. Unlike the seminal projects on manually crafted knowledge bases and ontologies, like Cyc [28] and WordNet [15], knowledge harvesting is automated and operates at Web scale.

Automatically constructed knowledge bases – KB's for short – have become a powerful asset for search, analytics, recommendations, and data integration, with intensive use at big industrial stakeholders. Prominent examples are the Google Knowledge Graph, Facebook's Graph Search, Microsoft Satori as well as domain-specific knowledge bases in business, finance, life sciences, and more.

These achievements are rooted in academic research and community projects starting ten years ago, most notably, DBpedia [2], Freebase [6], KnowItAll [14], WikiTaxonomy [35] and Yago [42]. More recent major projects along these lines include BabelNet [32] ConceptNet [41], DeepDive [40], EntityCube (aka. Renlifang) [34], KnowledgeVault [10], Nell [7] Probase [51], Wikidata [48], XLore [49].

The largest of the KB's from these projects contain many millions of entities (i.e., people, places, products etc.) and billions of facts about them (i.e., attribute values and relationships with other entities). Moreover, entities are organized into a taxonomy of semantic classes, sometimes with hundred thousands of fine-grained types. All this is often represented in the form of *subject-predicate-object (SPO) triples*, following the RDF data model, and some of the KB's – most notably DBpedia – are central to the Web of Linked Open Data [19].

For illustration, here are some examples of SPO triples about Steve Jobs:

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

```
Steve_Jobs type entrepreneur          entrepreneur subtypeOf businessperson
Steve_Jobs coFounded Apple            Steve_Jobs hasFriend Steve_Wozniak
Steve_Jobs hasDaughter Lisa_Brennan   Apple_Lisa namedAfter Lisa_Brennan
Steve_Jobs diedOf Pancreatic_Cancer   Steve_Jobs fanOf Bob_Dylan
```

The most obvious use case for this kind of encyclopedic knowledge is to support search engines (for both Internet and enterprise search) on queries about entities. For example, when receiving the query "jobs apple", the system can identify "jobs" as a possible target entity and use the KB to improve its answers (transparently to the user, and in addition to pursuing other interpretations of the user's intent, e.g., looking for job offers at Apple). Other benefits arise from aggregating observations (on queries, clicks, posts, etc.) on a per-entity basis and combining them with semantic types or entity facts for analytics and recommendations.

Once computers have background knowledge on the real world, they have better ways of tapping into vague and noisy contents like natural language texts, social media and Internet data. This is an asset for language understanding, data cleaning and more. Moreover, we can think of this duality of knowledge acquistion and content understanding as a virtuous cycle: it enables the computer to obtain more knowledge, better knowledge, deeper knowledge. Figure 1 illustrates this point.

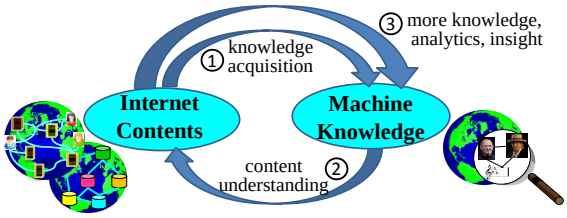


Figure 1: Knowledge Acquision and Content Understanding.

Our own endeavor on knowledge harvesting has its roots in research on semantic search starting in 2004. Later it became the *Yago-Naga project*, and led to the first release of the Yago KB (`yago-knowledge.org`) in February 2007. The salient strength of Yago is its rich type system with hundred thousands of fine-grained classes. When IBM Watson won the Jeopardy quiz show, it harnessed Yago's taxonomy for semantic type checking [26]. Later Yago releases added temporal and spatial knowledge [22], multilingual dimensions [38] and commonsense properties [17, 45]. Yago is now a joint project of the Max Planck Institute for Informatics and the Télécom ParisTech University. It is the only publicly available KB with statistical quality assurance: at least 95% accuracy (i.e., correct triples) based on sampling and Wilson confidence intervals.

This article intends to review the knowledge harvesting work of the past decade, pointing out lessons learned as well as open challenges and research opportunities. We draw on our Yago-Naga project as a primary source of experience, but aim to reflect the general research avenue.

2 Lessons Learned

2.1 Extraction Sources and Methods: Low-Hanging Fruit First

There is a wide spectrum of *information extraction (IE)* methods that knowledge harvesting can be based on, including regular expression matching, probabilistic graphical models, constraint reasoning and more. These can be applied to a wide spectrum of potential input sources, ranging from specialized databases (e.g., on movies or music) and web tables all the way to news articles and social media. From a purely scientific perspective, it may seem desirable to tap into every possible source with a single unified method and as little supervision as possible. Our experience is that this does not work - not if the goal is to build a high-quality knowledge base, with precision close to what a team of human curators would achieve.

Choice of Sources: We have followed – and strongly advocate – a layered approach where we pick low-hanging fruit first: high-quality input sources with limited noise in content structure, and robust methods for high-quality output. Specifically, we first tapped into semi-structured elements of Wikipedia: category names, infoboxes, lists, headings, etc. For specific kinds of knowledge, we integrated the best available curated sources: WordNet for taxonomic relations among semantic classes, and GeoNames for spatial entities. If we had been keen to increase Yago's coverage of movies and songs, we would have tapped into sources like IMDB (or LinkedMDB), Musicbrainz, etc. Generally, entities of specific types can often be harvested from dedicated sources or via specific identifier systems [44].

Once we obtained a KB core, we were able to harness this to distantly supervise extractions from other sources. In doing so, we could still "cherrypick" the more suitable sources: there is no point in obtaining poor extractions from super-noisy inputs like social media if the same knowledge can be distilled from easier inputs such as biographies. Also, we can leverage redundancy and statistics: seeing the same facts many times in different sources. This is why we called our approach *knowledge harvesting*, as opposed to running IE on each and every, arbitrarily difficult, input.

Extraction Methods: Similar engineering principles apply to the extraction methods. We started with simple, robust methods like regular expression matching (applied to token sequences in Wikipedia) combined with simple linguistic analysis, most notably, noun phrase parsing. Then we leveraged semantic type checking, building on the rich taxonomic knowledge distilled from Wikipedia categories and WordNet classes. This way we could keep precision at near-human quality, and were able to feed the obtained knowledge as seeds into more advanced distantly supervised methods: *reasoning with consistency constraints* over evidence-weighted candidate assertions [31, 43].

Our constraint reasoning is based on approximately solving Weighted MaxSat problems, which is equivalent to MAP inference for probabilistic graphical models. We believe that explicit constraint reasoning can be tuned more easily, though. A key aspect is the manual crafting of consistency constraints: functional dependencies, inclusion dependencies, temporal constraints, and more. For example, we can specify that a person (such as `Lisa_Brennan`) can have only one father, and that a person can found a company or compose a song only while being alive (i.e., between birth and death dates). Although this approach requires a modest amount of human supervision, we never encountered this to be a bottleneck.

Other papers with similar lessons include [8, 9, 30, 54].

2.2 Data Representation: Triples, Triploids, Quads, and Beyond

Like other KB projects, we decided to represent facts in the form of SPO triples, following the RDF data model.

Canonicalization: An important design decision, not shared by all of the major KB's, is to aim for database-style rigor in capturing the S, P and O roles of triples. We wanted S to be only *canonicalized entities* (or classes when P is `subtypeOf`), P to be only explicitly specified relations (with type signatures), and O to be only entities or literals such as dates or numbers (or classes for `type` or `subtypeOf`). Here, canonicalization means that we can uniquely identify each entity, and that all facts that refer to the same entity are attached to the same S value, regardless of the surface names under which the entity is discovered. For example, we insist that the two sentences *"Jobs is one of the co-founders of Apple."* and *"Steve founded Apple, together with his friend Woz."* result in the same fact `Steve_Jobs coFounded Apple`. Conversely, we have to carefully distinguish occurrences of "Woz" meaning Steve Wozniak against occurrences that abbreviate the game "Wizard of Oz". This issue calls for *named entity disambiguation* (aka. entity linking) [39], which we initially integrated into constraint reasoning [43] while later developing a general-purpose stand-alone solution [21]. Occurrences of "Steve" that our methods cannot map to a unique entity with high confidence would be disregarded.

Triples vs. Triploids: Our design emphasizes *precision* at the expense of *recall*. Several other KB's show facts with different S values even if they refer to the same entity, such as `Jobs founded Apple`, `Steve coFounded Apple`, `SteveJobs coFounded Apple`, `Woz coFounderOf Apple`, and so on. Here, the P

values are not properly canonicalized either. Unlike some KB projects that see this as an advantage, resulting in a larger KB, we believe that noisy redundancy and ambiguity is the recipe for inconsistency and degraded quality in downstream applications.

In retrospect, we still stand by this design choice, but we would be open to *additionally* harvesting non-canonicalized assertions, using so-called Open IE methods [29]. We would then treat S, P and O as textual phrases, to explicitly distinguish them from canonicalized triples. As this is no longer within the RDF standard, we refer to this case as *triploids*. Here are some examples, including mixed cases with partial canonicalization:

```
"Steve" "revolutionized" "music industry"     "Steve" invented iPod
Steve_Jobs "dated" Joan_Baez                  Steve_Jobs "admired" "Dylan"
```

The rationale for tolerating the co-existence of triples and triploids is to increase the coverage of the KB and support more use cases. Search applications could work well even with triploids, whereas other cases require rigorous reasoning and could thus be restricted to proper triples. We call the hybrid representation an *Extended Knowledge Graph (XKG)* [53], essentially a richly labeled graph with canonicalized or textual labels for nodes and edges. An XKG could be incrementally turned into a KG in a pay-as-you-go manner, using methods for post-hoc canonicalization of entity names and predicate phrases [16].

Beyond Triples: It is good practice to associate data with its provenance. To this end, we attached to each SPO triple metadata about the extraction source, method and confidence. As there could be multiple sources, we refer to this aspect as "knowledge witnesses". How do we represent this? The Semantic Web community advocates so-called *quads* for this purpose: adding a fourth dimension to each triple which encodes its provenance. However, this is insufficient as we need to capture multiple witnesses, extraction dates, confidence scores, etc. So we used a technique akin to reification: each SPO triple is given an identifier, and these identifiers can in turn be used as S values in additional (metadata) triples.

Following the idea of the virtuous cycle in Figure 1, we also harvested *spatial* and *temporal knowledge* about (certain kinds of) SPO triples [22]. For example, for a triple like `Steve_Jobs isCEOof Apple`, we distill the respective time intervals for the validity of this fact from additional sources [18, 50]. Likewise, events can be positioned in space and time, e.g., by capturing that `Steve_Jobs announced iPhone` happened on January 9, 2007 in San Francisco. This is represented in the KB as follows (with identifiers prefixing the triples):

```
id1:  ( Steve_Jobs isCEOof Apple )
id2:  ( id1 validDuring [1997-07-09, 2011-08-24] )
id3:  ( Steve_Jobs announced iPhone )
id4:  ( id3 happenedOn 2007-01-09 )     id5:  ( id3 happenedIn San_Francisco )
```

While this representation may appear elegant, it is unwieldy for querying. Simple queries such as asking for iPhone-related events in 2007 become fairly complex. We even invented extensions to the RDF query language Sparql to express search conditions with fact identifiers:

```
Select ?x, ?y Where {
     ?id:  ?x ?y iPhone .  ?id happendOn 2007-##-## .  }
```

SPOTLX Tuples: These considerations made us rethink our choice for RDF, and eventually led to the model of *SPOTLX tuples* for the Yago2 release in 2011 [22]. Each fact was expressed as a six-tuple with Subject, Predicate, Object, Time, Location and teXt (or conteXt), plus additional metadata attributes. The X component allows textual witnesses for facts, which could be queried jointly with the facts in the KB. For example, to find iPhone-related events in 2007 in the Bay Area which involved "standing ovations", we could use SQL over SPOTLX tuples, including abstract data types for time, space and text. The RDF representation has still been kept in parallel, for easy data exchange and interoperability in the Linked Open Data world.

2.3 Data Storage and Query Processing: Join, Relax, Rank and Scale

Since we initially focused on SPO triples, we desired efficient support for the Sparql query language. Because of the fine-grained nature of RDF data, this calls for extensive join processing, a typical example being:

```
Select ?x Where {
     ?x bornIn ?t .  ?t inCountry ?c .  ?c locatedIn Europe .
     ?x performed ?s .  ?s type song .  ?s composedBy Bob_Dylan .  }
```

This query finds European artists who covered Bob Dylan. It consists of 6 triple patterns: a 6-way self-join over the schema-free SPO table. As there was no high-performance RDF engine at that time, we built our own: the RDF-3X system with emphasis on join optimization [33].

This served us well for some time, but the transition from triples to quads and SPOTLX tuples (see above) led us to move to a standard relational engine, namely, PostgreSQL. However, as our KB grows and we pursue new KB-driven applications, we reconsider this decision. Our recent endeavor to support entity-centric large-scale text analytics [24], employs a cloud-based platform using Spark with Cassandra for storing the KB. This key-value storage solution gives full flexibility to represent knowledge tuples, decent query processing performance, and an easy way to scale out. However, this platform is far from ideal for the performance of many-way joins and SPOTLX queries. So we may have to keep revisiting this design choice.

Ranking of Query Answers: In addition to query performance, an important concern for us has been the need to rank query answers and to support query relaxation. These requirements arise as users who explore the KB are not familiar with its structure, terminology and content. Broad queries return many answers; so we need ranking to identify the most informative ones. An example is the query about iPhone-related events given above. With KB's being part of the highly heterogeneous Linked-Data ecosystem, this issue becomes even more demanding. Therefore, we developed IR-style statistical ranking models for query answers from triple patterns and complex Sparql queries [12, 27].

Query Relaxation: Even with perfect answer ranking, querying a KB still poses a high burden even for skilled users like analysts or journalists, due to the potential mismatch between the user's and the KB's vocabulary and structure. Reconsider the query about European artists covering Bob Dylan. Albeit perfectly formulated, it may still return very few answers or none at all, simply because the predicates in the query – `inCountry`, `performed`, `composedBy` – could be sparsely populated. Perhaps, cities are more often in the `locatedIn` relation which should better be transitively applied. Songs may better be found by using two predicates `hasAlbum` between artists and albums, and `hasTracks` between albums and songs. Instead of the `composedBy` predicate, the inverse relation `composed` between artists and songs may be the preferred way when populating the KB. Or perhaps neither of the two predicates is much in use; the KB could instead have triploids with P phrases like "composer of", "created", "his masterpiece".

This suggests alternative query formulations, which we call *query relaxations*. A good search interface should automatically generate one or more relaxations as needed. However, as we deviate from the user's original formulation, answers may be treated with lower confidence, and we have to merge results from multiple relaxations – another case for answer ranking.

Over the last ten years, our understanding for these issues in KB search and exploration has gradually improved. The form-based interface of our recent Trinity system [53] supports such relaxations and rankings. [4] gives a comprehensive survey on semantic search.

2.4 Data Evolution: The Knowledge Awakens

Hardly anything lasts forever. In a KB, attribute values (e.g., city populations) and relationships of entities (e.g., the CEO of a company or the spouse of a person) change over time. Even the set of entities under consideration is not fixed: new entities are being created all the time (e.g., new songs, sports matches, newborn children of celebrities) and need to be added to the KB. Also, existing entities could be irrelevant for a KB, but become

prominent at a certain point. Examples are when an unknown "garage band" or "garage company" starts having success. If Wikipedia had existed in April 1976, it would probably not have included Apple for insufficient notability.

So a KB should be continuously updated, for example, by subscribing to change feeds from Wikipedia and other data streams. DBpedia tried this [20], but abandoned it for its complexity. Yago instead used an approach with periodic rebuilding of the entire KB. Freebase and Wikidata have update processes in place, but seem to critically rely on human curation.

One difficulty that prevents a straightforward solution is that sometimes new facts can be simply added to the KB whereas others need to invalidate and overwrite previously included facts. Comprehensive versioning of all triples or tuples would alleviate this issue, but comes at the cost of making querying and exploration more complex. The general data evolution problem for KB's still appears to be widely open.

Active Knowledge: For highly dynamic and specialized knowledge, the Yago-Naga project explored an approach for linking KB items with external databases and web services. For example, the chart positions of a song and the box office counts of a movie change so rapidly that it is hardly meaningful to materialize these values in the KB. Instead, one should have automatic linking across knowledge repositories and to web service calls that return up-to-date values on demand. We developed techniques towards this form of *active knowledge* [36]; more work is needed along these lines.

Emerging Entities: One aspect of knowledge evolution for which we have a reasonable success story is *emerging entities* [23]. When identifying entity names in input sources (text, web tables, etc.), we attempt to disambiguate them onto the already known entities in the KB. However, we always consider an additional virtual candidate: none of the known entities. When the evidence and our methods suggest that we observe an *out-of-KB entity*, we capture it under its surface name along with its context. After some time, we obtain a repository of such emerging entities. Then we run clustering techniques to group them, and involve users to confirm this canonicalization. Finally, the emerging entities with sufficient support and confirmation can be added to the KB as first-class citizens. To alleviate humans from labor-intensive curation, it is important to present new entity candidates with informative context [25].

A specific case for out-of-KB entities is constructing ad-hoc KB's on the fly. Consider a new corpus of documents becoming available, for example, the Panama Papers or a batch of articles on specific health issues such as Zika infections. The goal is to automatically build a domain-specific KB that helps journalists and analysts to obtain an overview and drill down into finer issues. Our startup `ambiverse.com` pursues applications along these lines [13, 1].

3 Challenges and Opportunities

3.1 Knowledge Base Coverage

Some of the existing KB's are huge, but no KB will ever be complete. In fact, one can observe all kinds of knowledge gaps.

Locally incomplete knowledge is when certain O values are missing for a given S and P value – for example, when we know some movies of a director but not all of them. A variant of this is when for a given P value, we have O values for some S but not all of them – for example, knowing spouses of some people but missing out on many other married people. The difficulty here is not just to fill these gaps, but to realize when and where gaps exist. In other words, when can we assume a locally closed world, and how can we find evidence for an open world that can provide additional facts? [37] offers further discussion along these lines.

A second challenge arises from **long-tail entities** and **long-tail classes**. There are many lesser known musicians, regional politicians and good but not exactly famous scientists. How can we identify these in the Internet, and harvest facts about them? Long-tail classes pose a similar problem: despite some KB's having hundred thousands of classes, one could always add more interesting ones. For example, what if we got interested in

a class of `GratefulDeadFans` (i.e., fans of the rock band, which would have Steve Jobs as a member) or `HippieRetroConcerts`? Where in the class taxonomy do we fit these in, and how can we find their members (which is difficult even among the entities in the KB)? [11] offers further discussion along these lines.

Third and last, there is a big deficit in terms of **missing salient facts** about entities. KB's have been built in an opportunistic manner, mostly relying on Wikipedia. If Wikipedia does not have the information or if a fact is stated only in sophisticated form in the article's text, all the KB's miss out on it. For example, what is notable about Johnny Cash, the late singer? One key fact is that he recorded a live album on a free concert in a US state prison – in 1968 when this was a sensation if not a scandal. KB's contain the name of the album, but not the special circumstances. What is notable about the Nick Cave album "Abbatoir Blues"? Many KB's contain this album, listing its individual songs. No KB points out, though, that the song "Let the Bells Ring" is about Johnny Cash and that the song "O Children" is used in one of the Harry Potter movies. Part of the problem is the poor coverage of predicate types: KB's are missing predicates like `songIsAbout`. But this alone cannot fix the issue with the Folsom Prison concert. There is a great research opportunity here. Similar points can be made for *spatial knowledge* and *temporal knowledge*.

One may argue that Open IE could fill these gaps, and one should add more textual descriptions to an Extended Knowledge Graph. This would be a remedy, but only in a superficial sense as humans would then have to read and interpret a large amount of noisy text to look up facts. The fundamental trade-off between precision and recall cannot be eliminated this way.

3.2 Commonsense, Rules and Socio-Cultural Knowledge

Automatically constructed KB's have largely focused on harvesting encyclopedic fact knowledge. However, to power semantic search and other intelligent applications (e.g., smart conversational bots in social media), computers need a much broader understanding of the world: properties of everyday objects, human activities, plausibility invariants and more. This ambitious goal suggests several research directions.

We need to distill **commonsense** from Internet sources. This is about properties of objects like size, color, shape, parts or substance of which an object is made of, etc., and knowledge on which objects are used for which activities as well as when and where certain activities typically happen. For example, a rock concert involves musicians, instruments – almost always including drums and guitars, speakers, a microphone for the singer; the typical location is a stage – often but not necessarily in a large hall, and so on. This background knowledge could improve the interpretation of (spoken) user questions, and also image and video search when user queries include abstractions or emotions that cannot be directly matched by captions, tags or other text. Today's search engines return poor results on queries such as "exhausted band at hippie concert" (where the user hopes to find footage of performances by the Grateful Dead or the Doors). Prior work on acquiring commonsense includes ConceptNet [41] and our recent project WebChild [45, 46]. However, there is still a long way to go for computers to learn what every child knows.

Rules that capture invariants over certain kinds of facts is another key element for advancing the intelligent behavior of computer systems. For example, a rule about scientists and their advisors could be that the advisor is a professor at the scientist's alma mater – as of the time when the scientist graduated. Such rules may have exceptions, but could be a great asset to answer more queries and to infer additional facts for "KB completion". Even soft rules can be useful: an example could be that folk guitarists are usually also singers. To acquire this intensional knowledge, rule mining methods have been developed and applied to large KB's – one notable work being our AMIE project [17]. However, the state of the art exhibits major limitations. Rules are restricted in their logical form to Horn clauses or at least clauses. This does not allow rules with existential quantifiers or with disjunctions in the rule head, for example, stating that every human person has a mother and that every human is male or female. The seminal Cyc project [28] had its focus on commonsense rules, even including higher-order logics, but exclusively relied on human experts to manually specify them. Another major challenge with automatic rule mining arises from the open world assumption that underlies KB's and the bias in observations

from Internet sources. For example, if the KB has no entrepreneur who founded more than 3 companies, this should not imply a cardinality constraint. Likewise, if the facts in the KB suggest a rule that every founder of an IT company has become a billionair, this may be caused by the bias in the KB construction (e.g., by harvesting only successful entrepreneurs from Wikipedia) and does not entail that the rule is valid in the open world.

Yet another dimension where today's KB's fall short is the **socio-cultural context** of facts or rules. Even seemingly objective statements on discoveries and inventions often depend on the background and viewpoint of a certain group of people. For example, in the US most people would state that the computer was invented by Eckert and Mauchley, whereas a German would give the credit to Konrad Zuse and a British would insist on Alan Turing (or perhaps Charles Babbage). This is not just geographical context; teenagers, for example, may largely think of Steve Jobs as the (re-) inventor of the (mobile) computer. For commonsense knowledge, it is even more critical to capture socio-cultural contexts. This also requires more thought on appropriate representations (certainly beyond SPO triples).

3.3 Interactive Search and Exploration

KB's have become so large and heterogeneous, in terms of structure and terminology, that users struggle with formulating queries – even when supported by a form-based or faceted UI (e.g., [3]). Even worse, a major use case of KB's is to serve as background reference when data scientists or business analysts combine and explore other datasets or online media. For example, a life scientist or political scientist may rapidly collect tens of interesting datasets for a specific study, but would then drastically lose her productivity when trying to join different data items and search for patterns, trends and insight.

This calls for new modes of interactive search and exploration of KB's and associated datasets. We believe the most effective way of relieving the user from the necessity to cope with the complex structure of the data, is by means of **natural language** for question answering and other interactions. User inputs such as "Which European singers covered Bob Dylan?" can be translated into structured Sparql queries. There has been substantial work on this task recently (e.g., [5, 47, 52]). However, this is still restricted to simple questions that return a single fact or a list of entities. Questions with multi-relation hops (i.e., chain joins in DB jargon) or asking for composite answers is beyond scope. Even more demanding is coping with colloquial inputs, where the user first describes her information need and context and then poses an underspecified question. An example would be: "Steve was a big fan of Bob Dylan. Didn't he date a folk singer to get closer to him? Who was that?"

Even if these issues were solved, merely translating each question into a single one-shot query would not be sufficient. The generated query may not capture the user intent, it may return unexpected, unwanted or just too few answers, and users may be utterly puzzled on how to formulate the very first question in order to start their knowledge exploration. So we need to extend question answering to interactive dialogs, with the system explaining answers and guiding the user towards better questions.

3.4 Knowledge Base Life-Cycle

It has been a major endeavor to build a comprehensive KB. However, it is an even more daunting task to maintain, grow and improve it over a long time horizon. Discovering and adding emerging entities is just the tip of the iceberg. Especially in combination with expanding the scope towards commonsense, rules and socio-cultural context, coping with new knowledge, obsolete knowledge and context-specific knowledge poses enormous challenges. The crux will be to minimize the cost of human curation, while keeping quality assurance a key priority.

References

[1] Ambiverse: Text to Knowledge. https://www.ambiverse.com

[2] S. Auer et al.: DBpedia: A Nucleus for a Web of Open Data. ISWC 2007

[3] H. Bast, F. Bäurle, B. Buchhold, E. Haussmann: Semantic full-text search with Broccoli. SIGIR 2014

[4] H. Bast, B. Buchhold, E. Haussmann: *Semantic Search on Text and Knowledge Bases*. Foundations and Trends in Information Retrieval 10(2-3):119-271

[5] J. Berant, A. Chou, R. Frostig, P. Liang: Semantic Parsing on Freebase from Question-Answer Pairs. EMNLP 2013

[6] K.D. Bollacker et al.: Freebase: a Collaboratively Created Graph Database for Structuring Human Knowledge. SIGMOD 2008

[7] A.J. Carlson et al.: Toward an Architecture for Never-Ending Language Learning. AAAI 2010

[8] L. Chiticariu, Y. Li, F.R. Reiss: Rule-Based Information Extraction is Dead! Long Live Rule-Based Information Extraction Systems! EMNLP 2013

[9] O. Deshpande et al.: Building, Maintaining, and Using Knowledge Bases: a Report from the Trenches. SIGMOD 2013

[10] X. Dong et al.: Knowledge vault: A web-scale approach to probabilistic knowledge fusion. KDD 2014

[11] X. Dong: How Far Are We From Collecting the Knowledge of the World. Keynote at WebDB 2016, http://lunadong.com/talks/tailKnowledge.pdf

[12] S. Elbassuoni, M. Ramanath, R. Schenkel, G. Weikum: Searching RDF Graphs with SPARQL and Keywords. IEEE Data Eng. Bull. 33(1): 16-24 (2010)

[13] P. Ernst et al.: DeepLife: An Entity-Aware Search, Analytics and Exploration Platform for Health and Life Sciences. ACL 2016

[14] O. Etzioni et al.: Unsupervised Named-Entity Extraction from the Web: an Experimental Study. Artificial Intelligence 165(1): 91-134

[15] C. Fellbaum, G. Miller (Editors): *WordNet: An Electronic Lexical Database*. MIT Press, 1998

[16] L. Galárraga, G. Heitz, K. Murphy, F.M. Suchanek: Canonicalizing Open Knowledge Bases. CIKM 2014

[17] L. Galárraga, C. Teflioudi, K. Hose, F.M. Suchanek: Fast rule mining in ontological knowledge bases with AMIE+. VLDB J. 24(6): 707-730 (2015)

[18] T. Ge, Y. Wang, G. de Melo, H. Li, B. Chen: Visualizing and Curating Knowledge Graphs over Time and Space. ACL 2016

[19] T. Heath, C. Bizer: *Linked Data: Evolving the Web into a Global Data Space*. Morgan & Claypool, 2011

[20] S. Hellmann, C. Stadler, J. Lehmann, S. Auer: DBpedia Live Extraction. OTM Conferences 2009

[21] J. Hoffart et al.: Robust Disambiguation of Named Entities in Text. EMNLP 2011

[22] J. Hoffart, F.M. Suchanek, K. Berberich, G. Weikum (2013): YAGO2: A Spatially and Temporally Enhanced Knowledge Base from Wikipedia. Artificial Intelligence 194: 28-61

[23] J. Hoffart, Y. Altun, G. Weikum: Discovering emerging entities with ambiguous names. WWW 2014

[24] J. Hoffart, D. Milchevski, G. Weikum: STICS: searching with strings, things, and cats. SIGIR 2014

[25] J. Hoffart et al.: The Knowledge Awakens: Keeping Knowledge Bases Fresh with Emerging Entities. WWW 2016

[26] IBM Journal of Research and Development 56(3/4), Special Issue on "This is Watson" (2012)

[27] G. Kasneci et al.: NAGA: Searching and Ranking Knowledge. ICDE 2008

[28] D.B. Lenat: CYC: A Large-Scale Investment in Knowledge Infrastructure. Comm. ACM 38(11): 32-38

[29] Mausam et al.: Open Language Learning for Information Extraction. EMNLP-CoNLL 2012

[30] T. Mitchell et al.: Never-Ending Learning. AAAI 2015

[31] N. Nakashole, M. Theobald, G. Weikum: Scalable knowledge harvesting with high precision and high recall. WSDM 2011

[32] R. Navigli, S.P. Ponzetto: BabelNet: The automatic construction, evaluation and application of a wide-coverage multilingual semantic network. Artif. Intell. 193: 217-250 (2012)

[33] T. Neumann, G. Weikum: RDF-3X: a RISC-style engine for RDF. PVLDB 1(1): 647-659 (2008)

[34] Z. Nie, J.-R. Wen, W.-Y. Ma: Statistical Entity Extraction From the Web. Proceedings of the IEEE 100(9): 2675-2687 (2012)

[35] S.P. Ponzetto, M. Strube: Deriving a Large-Scale Taxonomy from Wikipedia. AAAI 2007

[36] N. Preda et al.: Active knowledge: dynamically enriching RDF knowledge bases by web services. SIGMOD 2010

[37] S. Razniewski, F.M. Suchanek, W. Nutt: But Do We Actually Know? AKBC 2016

[38] T. Rebele et al.: YAGO: a multilingual knowledge base from Wikipedia, Wordnet, and Geonames. ISWC 2016

[39] W. Shen, J. Wang, J. Han: Entity Linking with a Knowledge Base: Issues, Techniques, and Solutions. IEEE Trans. Knowl. Data Eng. 27(2): 443-460 (2015)

[40] J. Shin et al.: Incremental Knowledge Base Construction Using DeepDive. PVLDB 8(11): 1310-1321 (2015)

[41] R. Speer, C. Havasi: Representing General Relational Knowledge in ConceptNet 5. LREC 2012

[42] F.M. Suchanek, G. Kasneci, G. Weikum: YAGO: a Core of Semantic Knowledge. WWW 2007

[43] F.M. Suchanek, M. Sozio, G. Weikum: SOFIE: a self-organizing framework for information extraction. WWW 2009

[44] A. Talaika, J. Biega, A. Amarilli, F.M. Suchanek: IBEX: Harvesting Entities from the Web Using Unique Identifiers. WebDB 2015

[45] N. Tandon, G. de Melo, F.M. Suchanek, G. Weikum: WebChild: harvesting and organizing commonsense knowledge from the web. WSDM 2014

[46] N. Tandon, G. de Melo, A. De, G. Weikum: Knowlywood: Mining Activity Knowledge From Hollywood Narratives. CIKM 2015

[47] C. Unger, A. Freitas, P. Cimiano: An Introduction to Question Answering over Linked Data. Reasoning Web 2014

[48] D. Vrandecic, M. Krötzsch: Wikidata: a free collaborative knowledgebase. Commun. ACM 57(10): 78-85 (2014)

[49] Z. Wang et al.: Xlore: A large-scale english-chinese bilingual knowledge graph. ISWC 2013

[50] Y. Wang, M. Dylla, M. Spaniol, G. Weikum: Coupling Label Propagation and Constraints for Temporal Fact Extraction. ACL 2012

[51] W. Wu, H. Li, H. Wang, K.Q. Zhu: Probase: a Probabilistic Taxonomy for Text Understanding. SIGMOD 2012

[52] M. Yahya et al.: Natural Language Questions for the Web of Data. EMNLP-CoNLL 2012

[53] M. Yahya, D. Barbosa, K. Berberich, Q. Wang, G. Weikum: Relationship Queries on Extended Knowledge Graphs. WSDM 2016

[54] C. Zhang, J. Shin, C. Ré, M.J. Cafarella, F. Niu: Extracting Databases from Dark Data with DeepDive. SIGMOD 2016

Towards a Game-Theoretic Framework for Text Data Retrieval

ChengXiang Zhai
Department of Computer Science, University of Illinois at Urbana-Champaign
Email: czhai@illinois.edu

Abstract

The task of text data retrieval has traditionally been defined as to rank a collection of text documents in response to a query. While this definition has enabled most research progress so far, it does not model accurately the actual retrieval task in a real search engine application, where users tend to be engaged in an interactive process with multipe queries, and optimizing the overall performance of a search engine system on an entire search session is far more important than its performance on an individual query. This paper presents a new game-theoretic formulation of the text data retrieval problem where the key idea is to model text retrieval as a process of a search engine and a user playing a cooperative game, with a shared goal of satisfying the user's information need (or more generally helping the user complete a task) while minimizing the user's effort and the operation cost of the retrieval system. Such a game-theoretic framework naturally optimizes the overall utility of an interactive retrieval system over a whole search session, thus breaking the limitation of the traditional formulation that optimizes ranking of documents for a single query and can optimize the collaboration of a search engine and its users, thus maximizing the "combined intelligence" of a system and users. Although the framework was motivated by text data retrieval, it is actually quite general and potentially applicable to all kinds of information service systems.

1 Introduction

Text data are those data that are encoded in human natural languages such as English and Chinese. They are unique in that they are generated by humans and also meant to be consumed by humans. As a result, they play a very important role in our life. For example, most human knowledge is encoded in the form of text data; scientific knowledge almost exclusively exists in scientific literature. Because natural languages are the tools of communication by humans, we can usually describe other media such as video or images using text data, facilitating understanding of videos and images. From data mining perspective, because of the rich semantic content in text data, we can expect to discovery all kinds of knowledge from text data, especially knowledge about people's preferences and opinions, which are often best reflected in the text data produced by them and may be hard to obtain from other kinds of machine-generated data.

The amount of text has recently grown dramatically due to the digitalization of information and the widespread deployment of tools for people to easily produce and consume text online. This creates a significant challenge for humans to consume and make use of all the text data in a timely manner Logically, in order to make use of

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

"big text data," we would first need to select the most relevant text data from a large collection of text documents and filter out many non-relevant text data; this not only helps filter out noise, but also significantly reduces the size of the data that we actually have to digest [26]. The most useful tool for supporting this first step in text data applications is a text data retrieval system, or a search engine. Indeed, Web search engines are now essential tools in our daily life, and they help us find relevant documents from the Web quickly, effectively addressing the problem of information overload. There are also many other examples of search engine applications such as product search, social media search, and scientific literature search. In general, wherever we have text data, we would need a search engine.

Since the accuracy of a search engine directly affects our productivity, it is very important to study how to improve the accuracy of all search engines so that we can save people's time. It is especially beneficial to develop general techniques that can improve *all* search engines since such techniques can be immediately used in all kinds of search engine applications to generate benefit in all application domains. For this reason, the optimization of general text retrieval models has been a long-standing important fundamental reseach question in the information retrieval community (see, e.g., [23, 16, 20]), and many general informtion retrieval models have been proposed and tested (see [25] for a detailed review of all these models).

So far, the task of text data retrieval has been mostly defined as to generate an optimal ranking of documents in response to a user's query. The ranking is usually based on a scoring function defined on a query-document pair to generate a relevance score. The theoretical justification for such a ranking formulation is the Probability Ranking Principle (PRP) [15], which states that returning a ranked list of documents in descending order of probability that a document is relevant to the query is the optimal strategy under two assumptions: (1) The utility of a document (to a user) is independent of the utility of any other document. (2) A user would browse the results sequentially. The intuition captured by PRP is the following: if a user sequentially examines one doc at each time, wed like the user to see the very best ones first, which makes much sense.

While the PRP has enabled most research progress in text retrieval so far, it does not model accurately the actual retrieval task in a real text retrieval application since neither assumption actually holds in practice; even the extended PRP for interactive retrieval [8] is still limited due to the assumption of sequential browsing. Moreover, retrieval is in general an interactive process often with multipe queries formulated, and thus optimizing the overall performance of a text retrieval system on an entire search session is far more important than its performance on an individual query. Unfortunately, the current formulation of the retrieval task makes it impossible to optimize over an entire session.

To address these limitations, we present a novel game-theoretic formulation of the text data retrieval problem where the key idea is to model text retrieval as a process of a search engine and a user playing a cooperative game, with a shared goal of satisfying the user's information need (or more generally helping the user complete a task) while minimizing the user's overall effort and the operation cost of the retrieval system. Such a game-theoretic framework offers two important benefits. First, it naturally suggests optimization of the overall utility of an interactive retrieval system over a whole search session, thus breaking the limitation of the traditional formulation of optimizing the ranking of documents for a single query. Second, it models the interactions between users and a search engine, and thus can optimize the collaboration of a search engine and its users, maximizing the "combined intelligence" of a system and users [4].

2 Text Retrieval as Cooperative Game Playing: Basic Idea

How can we optimize all search engines in a general way? This question is not well defined until we clearly define what a search engine is, and what is an optimal search engine. The previous section has made it clear that defining a retrieval task as optimizing a ranked list in response to a query has many limitations. However, what is the most general way to define the task of text retrieval that would address all the limitations then? We argue that the most general way to define a retrieval task is to define it as a search engine system playing a sequential

cooperative game with the user with a shared goal of helping the user finish the information seeking task with minimum overall user effort and minimum operation cost of the system.

In the simplest setting, we will consider just one user, though the idea can be easily generalized to include multiple users. In such a case, the game has two players: player 1 is the search engine, while player 2 is the user. The rules of the game are as follows:

1. Players take turns to make "moves" (just as in a board game like chess). A move is an action taken by user/system.

2. The first move is usually made by the user in the case of search, i.e., the entering of a query by the user (though in a recommender system, the system can also make the first move).

3. For each move of the user, the system makes a response move, i.e., shows an interactive interface; for each move of the system (i.e., each interactive interface), the user makes a response move (i.e., takes an action on the interface).

4. The game is over when the user has finished the information seeking task or decided to abandon the search (failed to find the needed information).

The objective of the game is to help the user complete the (information seeking) task with minimum overall effort and minimum operating cost for the search engine.

Such a game-theoretic view of the retrieval problem can be easily understood by analyzing the interaction process of the current search engine with a user as shown in Figure 1. In general, we may assume that the user

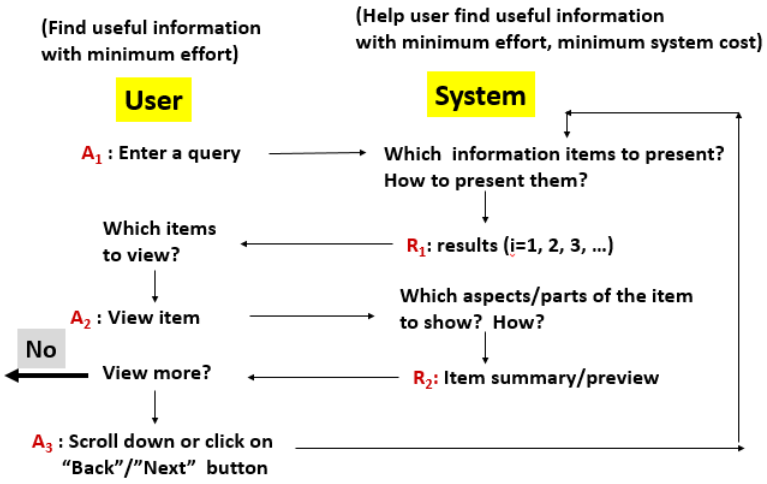


Figure 1: Retrieval process as a sequential game.

has the goal of finding useful information with minimum effort, while the system has the goal of helping user find useful information with minimum effort as well as minimum system cost. The user's query is the very first move of the game, made by the user (denoted as Action A_1). In response to this move, the system now has to make a decision over what action/move to take. The decision can be "which information items to present" and "how to present these items" in response to the user's query. Now suppose the system has decided to show a certain search result denoted by R_1 in an interface. The user would look at the interface and decide his/her next move. That is, the user would decide which items to view. The user then chooses an item to view (action A_2). In response to this move, the system would then further decide which move to make in response to the new action taken by the user. The decision questions here can include which aspects/parts of the selected information item

by the user should be shown to the user, and how to show them? In a regular retrieval system, such decisions are often ignored as the engine often just returns the document to the user, and the user would browse the document. However, if we are to optimize the response from the engine, we could imgine to display only the most relevant part of a long document with possibility to navigate into other parts, or show a summary that can further support navigation into the original content. Furthermore, if the system shows only the most relevant part of a document, the system can also adaptively optimize the presentation of the remaining part if the user chooses to scroll up or down (e.g., if the user has spent a lot of time reading the displayed relevant part, the system can infer the displayed content is interesting to the user, whereas if the use didn't really spend time to read it and directly chose to scroll up/down, the system would know to avoid displaying similar content). When viewed as a game, such opportunities for optimization would be obvious. After the user finishes examining a result, the user would again face a decision: should the user view more? If the user decides to view more, the user may decide to click on a "Next" button, which we again would view as a new move by the user. From a system's perspective, it could do exactly the same as what it did when it responded to the original qeury, i.e., the system would again decide which items to show and how to show them. This means after the user clicks on the "Next" button, the results shown to the user can be very different from the original next page since by this time, the system would have known much more about the user's need (based on analyzing the user's behavior on the first page). The process can go on and on until the user decides that he/she has viewed sufficient items and decides to stop.

There are many benefits of formulating the retrieval problem (i.e., the task of a search engine) as playing such a cooperative game (and thus also defining an optimal search engine as one that plays such a game with an optimal strategy). First, this gives us a formal framework to naturally integrate research in user studies, evaluation, retrieval models, and efficient implementation of retrieval systems since the optimization of the objective of such a game involves research in all these areas. Second, it provides a unified roadmap for identifying unexplored important retrieval research topics. Third, it emphasizes on optimization of an entire session instead of that on a single query (i.e., optimizing the chance of winning the entire game). Finally, it enables optimization of the collaboration of machines and users, i.e., maximizing collective intelligence [4].

The new formulation also raises many interesting new questions which we will discuss later in this paper. First, how should we design a text retrieval game? Specifically, how should we design moves for the user and the system, how should we design the objective of the game, and how can we go beyond search to support access and task completion? Second, how can we formally define the optimization problem and compute the optimal strategy for the IR system? Specifically, how do we characterize a text retrieval game, which category of games does a text retrieval game fit, to what extent can we directly apply existing game theory, and what new challenges must be solved? Finally, how should we evaluate such a system? In the rest of the paper, we will briefly address some of these questions with a focus on presenting a formal framework for optimizing the retrieval decisions in such a game.

3 Text Retrieval as Cooperative Game Playing: Formal Framework

It is important to point out that the game of retrieval is not a zero-sum game, thus it is different from a game such as chess in this sense. However, it shares similarity with chess in that they both involve optimization of sequential decisions over a horizon of multiple moves. Indeed, just as in chess where it sometimes makes sense to sacrifice a piece in order to win a game, it also makes sense for a search engine to sometimes take a "local loss" in order to gain more in the overall session. An example is when the query is ambiguous, a search engine can ask the user to clarify whether the word "Jaguar" in the query means a car or an animal. This move is not optimal because it is not as useful to the user as if the system simply takes a guess of the sense of "jaguar" and provides some search results, but it helps to optimize the results in all future moves once the system knows the intended sense of "jaguar."

The challenge is, however, how do we mathematically optimize such sequential decisions? How do we know

when the search engine should ask a question and when it shouldn't? To address these questions, we must first formalize our decision problem by introducing notations to denote all the important variables. This is illustrated in Figure 2.

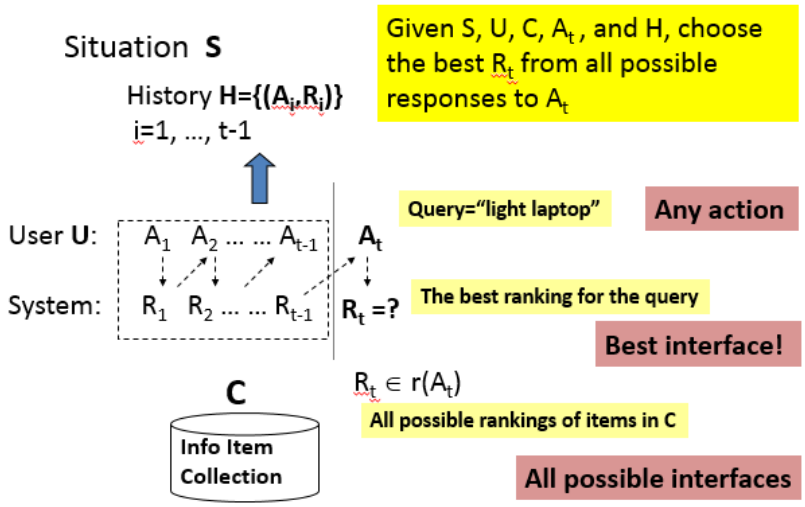


Figure 2: Formalization of the retrieval decision in a retrieval game.

We use A_i to denote an action taken by the user U, and R_i the corresponding response (system action) by the system. At time t, the system needs to choose a response R_t in response to the current user action A_t. For convenience, we would use $H = \{(A_i, R_i)\}$ to denote all the history information we can observe about the user-system interactions. We denote the collection of information items to be searched by C and further use a generic variable S to denote the current search scenario which can also be interpreted as representing the general context of the search (e.g., time and location of the search), which may also be a factor affecting the system's decision. For example, there may be some general preferences in the Holiday shopping season in a year or when an important international event has just happened.

With these variables, the retrieval decision problem can now be framed as: Given situation S, user U, collection C, history H, and the current action A_t, choose the best response R_t from all the possible responses to action A_t, which we denote by $r(A_t)$.

Note that such a formulation is much more general than the current formulation of the retrieval problem (which is to optimize ranking of documents for a query) since a user's action can be, theoretically speaking, any action that a user can take, including not just entering a query, viewing a document, or clicking on the "Next" button, but also any keystroke, any cursor movement, and many others. Also, the system's response to a user action can also easily go beyond the normal responses such as showing a ranked list of documents to potentially include *any* interaction interface that the system can show to the user, which further opens up many possibilities of actions that a user can take for further interaction with the system. Naturally, the traditional formulation of the retrieval problem can be easily seen as a special case where the user's current action A_t would be the query entered by the user, the response of the system R_t is a ranked list of documents in collection C, and the set $r(A_t)$ consists of all possible rankings of these documents.

With the introduced notations, we can now use Bayesian decision theory to formally frame the problem as a statistical decision problem, which we illustrate in Figure 3. The formulation can be regarded as a generalization of the risk minimization framework for retrieval [24]. In this figure, we see that our observed variables include S, U, H, C, and A_t, and the space of actions to choose is given by $r(A_t) = \{r_1, ..., r_n\}$, where r_i is a potential

action that the system can take. In order to assess which action is a good action to take, we introduce a loss function $L(r_i, M, S)$ which depends on the candidate action r_i, the situation S, and also a new variable M, which is a user model variable that encodes "everything" that we need to know about the user at the point for deciding which action to take. Intuitively, the system should choose a response that would minimize the value of the loss function. That is, we want to choose an r_i that has the smallest $L(r_i, M, S)$ (as compared with other possible responses).

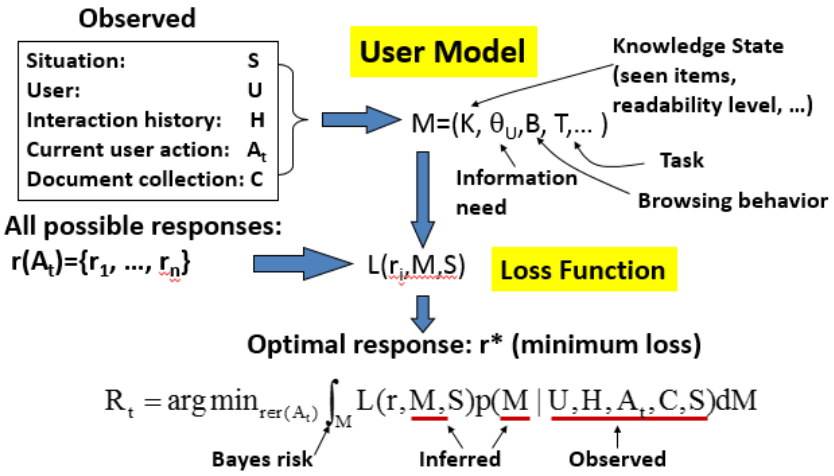


Figure 3: Bayesian decision theory applied to optimization of retrieval decisions.

However, we have note yet said what is exactly the user model variable M. Theoretically speaking, M can potentially encode all the detailed information that we might know directly or indirectly about the user. As a minimum, though, M would contain information about the user's information need, which initially can only be inferred based on the user's query, but can be updated if we know more information about the user. This was denoted by θ_U in the figure. M may also contain information about the user's knowledge status (which is denoted by K and can contain the items already viewed by the user, i.e., "known information" to the user), the user's reading level (which can be inferred based on the documents that the user has read/skipped), a user's browsing behavior, and any information about the user's task.

If M is clearly specified, our decision problem would be relatively easy as we just need to compute $L(r_i, M, S)$ for every r_i and choose the one minimizing our loss function. However, in general, we cannot clearly specify M, and thus the best we can do is to infer M based on all the observed variables using the posterior distribution $p(M|U, H, A_t, C, S)$, which captures our belief about M after we observe all the observables. Considering this uncertainty, the solution to our problem can be defined as the response that minimizes the expected loss (also called expected risk, or Bayes risk) as shown in the figure.

The computation of the Bayes risk involves the computation of an integral over the space of all possible user models, which is clearly intractable. It is thus interesting to look at an approximation of this integral by considering the mode (highest value) of the posterior distribution of the user model M, which we denote by M^*. Since $M^* = \arg\max_M p(M|U, H, A_t, C, S)$, with this approximation, we have

$$R_t \approx \arg\min_{r \in r(A_t)} L(r, M^*, S) p(M^*|U, H, A_t, C, S) \quad (8)$$

$$= \arg\min_{r \in r(A_t)} L(r, M^*, S) \quad (9)$$

This suggests the following two-step procedure for computing the optimal response: 1) Compute an updated user model M^* based on all the currently available information, i.e., compute $M^* = \arg\max_M p(M|U, H, A_t, C, S)$.

2) Given M^*, choose an optimal response to minimize the loss, i.e., compute $R_t = \arg\min_{r \in r(A_t)} L(r, M^*, S)$. We illustrate the game-playing process when the system is taking such a simplified strategy in Figure 4 where the system maintains a user model variable M as an internal state and dynamically updates it in each iteration and then chooses an optimal response to minimize the loss function.

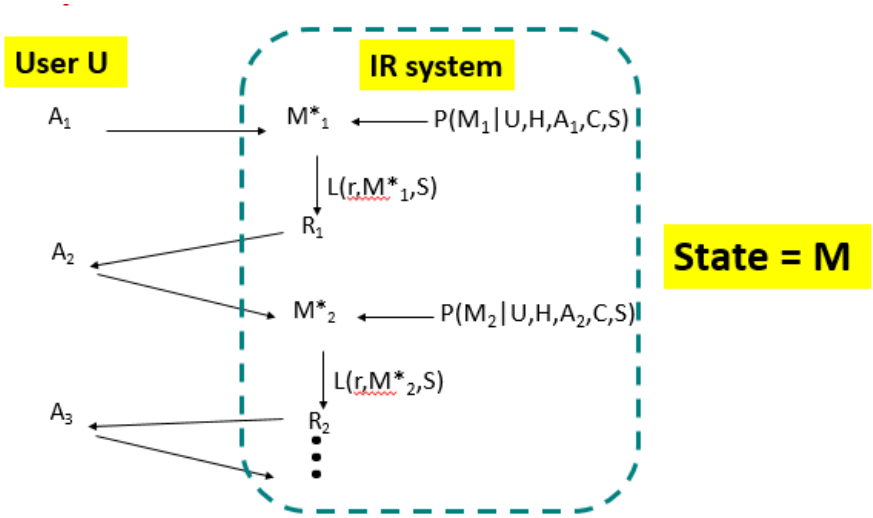


Figure 4: Optimal interactive retrieval based on dynamic updating of user model.

This process suggests that the sequential decision problem here can be naturally modeled with Partially Observable Markov Decision Process (POMDP) by treating M as the state and the update of M as transitions between the states. Indeed, Markov Decision Process (MDP), particularly POMDP, has been recently explored for information retrieval with encouraging results (see e.g., [12, 13]), but the modeling of M in the existing work is still quite limited. When POMDP is used to model interactive retrieval, we would be able to naturally use Reinforcement Learning to enable a retrieval system to learn an optimal decision strategy from its interactions with many users. The idea of search engine as a learning agent was also suggested in the work [9]. We anticipate to see more progress in applying POMDP and reinforcement learning to optimize interactive retrieval in the near future.

An especially interesting direction may be to introduce more detailed user behavior modeling into a retrieval decision framework so that M can be enriched with specific models capturing a user behavior when interacting with the retrieval system. From game-theoretic perspective, it is essential to model the user's decision process. In this direction, the recent work on economics in interactive IR (e.g., [1]) is very promising and can shed light on how we can refine M to model user behavior and further connect the behavior with the loss function to influence the choice of system response.

As often happens in sequential decision making, there is a tradeoff between exploration and exploitation. For example, it may be beneficial to diversify the retrieval results initially in hope of learning more completely about a user's information need, but over-diversification would incur cost because it may decrease the relevance of the current results (as we also include results that may be only marginally relevant). Recent work has explored this problem by using multi-armed Bandit for optimizing online learning to rank (e.g., [10]) and content display and aggregation (e.g., [7]). The game-theoretic framework can capture the exploration and exploitation tradeoff by defining the loss function with consideration of the expected future actions taken by the user.

The evaluation of a system designed based on the game-theoretic framework is challenging. While it is always possible to use A/B test in a real operational system to evaluate different decision strategies of the system based on user responses, it is unclear how we can create a reusable test collection that can be repeatedly used

to compare different game-playing algorithms (especially new algorithms yet to be developed) in a controlled manner. One possibility is to create many simulators of different kinds of users based on assumed user behavior and information needs, which can then be combined with a standard retrieval test collection to enable simulation-based evaluation.

4 Instantion of the Game-Theoretic Retrieval Framework

The framework presented in the previous section is very general, but it is mostly a conceptual framework. Thus in order to derive specific models and algorithms that can be implemented in a search engine, we must instantiate all the components in the framework, including (1) actions (i.e., "moves") that can be taken by a user and a system (A_i and $r(A_i)$), (2) user model M, (3) conditional distribution of M given all the observables, $p(M|U, H, A_t, C, S)$, and (4) the loss function $L(r, M, S)$. Clearly, there are many different ways to define these variables; each would lead to a potentially different specific strategy for a retrieval system to use for playing such a cooperative game (i.e., a strategy for optimizing the sequential retrieval decisions in a whole interaction session with a user). A thorough discussion of all these possibilities is beyond the scope of this paper, but we will briefly discuss some possibilities, mostly to suggest even more possibilities are possible.

4.1 Instantiation of "moves" in the game

We first look at the question how to define the moves in such a cooperative game, i.e., the actions that the user and the system can take respectively. It is easy to see that when viewed with this new framework, a current search engine can only play a very simple game with its users with extremely limited moves. Indeed, a user's actions are mostly restricted to entering a query, clicking on a result, scrolling up and down on a page, and clicking other buttons to navigate into additional results. The system's response is also mostly restricted to either produce a ranked list or passively show the requested results in the ranked list already pre-computed by the search engine based on the user's query. This is indeed a very simple game with very limited moves! The new framework suggests that the game can be much more sophisticated with many more possible moves. It is especially useful to include those moves that do not require much effort from the user, but woud help clarify the user's information need since this would help optimize human-computer collaboration by more actively engaging the user in a "dialogue" where the user and system help each other to help the user finish the information seeking task with minimum user effort and minimum system operation cost.

To see some specific moves that can be added to such a game, let us first consider the restrictive set of user actions the current search engine supports. A careful analysis would reveal that even for these actions, the system could have had many more "moves" to make than what is supported by the current search engine. For example, when a user requests the next page, the results on the next page can be completely reranked based on what the system has learned from the user's actions on the current page as done in some systems such as UCAIR [17] and SurfCanyon (http://www.surfcanyon.com/). The system can even go further to summarize the next page by using what the user has seen on the first page as the context. Similarly, when the user scrolls down on a page, the system would also have an opportunity to adaptively customize how to present the rest of the page; for example, the system can have likely non-relevant parts collapsed to save screen space or "decorate" the page with additional navigation buttons as appropriate, which would enable the user to make more moves (each navigation can be viewed as one possible move by the user). Of course, when we offer a user more actions to take, it would further provide more opportunities for the system to respond in different ways.

The framework also enables us to model user actions at different levels and thus allow a system to play the game at different levels. For example, the low level actions of users can include key strokes, mouse clicking and movement, or even eye tracking. When viewing a user's action at this level, we could identify opportunities for responding to a key stroke when the user enters a query with suggestions for query autocompletion. In this

sense, query autocompletion or query spelling correction is a natural strategy that can be derived from the game-theoretic framework of text retrieval. If we view query input as a medium-level user action, then we may also view an entire query session with multiple queries entered by the user (e.g., searching for hotels) as one "high-level" action by the user, enabling the system to make a "high-level" response by recommending information relevant to the user's task behind the search session (e.g., the task can be travel), such as recommeding attractions in the city of the hotel. Such a flexibility in modeling user-system interactions makes the framework very general and allows us to selectively model those actions that are most useful for a particular application scenario.

Another interesting example of a response to a user's action of entering query is to ask the user a clarification question such as "did you mean jaguar as a car or animal?" When looking at the optimization at the level of an individual query, such a response is clearly non-optimal as it does not provide any useful information to the user and is thus worse than presenting a ranked list of pages using any standard retrieval function. However, when viewed globally from the perspective of optimizing the entire session, it is possible that the local loss can easily be balanced by the gain in the future interactions due to the clear understanding of the intended sense of "jaguar." In some sense, this is similar to sacrificing a piece in chess to win the game where the loss of a piece would incur an immediate (short-term) loss, but it helps win the game (gains in the long run), and the game-theoretic view of the retrieval problem would allow us to use an algorithm to decide when the system may consider asking such a clarification question (e.g., when the current results are poor and the user has already reformulated the query multiple times).

How to improve search accuracy for long-tail queries is one of the most important and challenging questions faced today in optimizing a search engine. This is because the system does not see many instances of a query, making the machine learning approach ineffective. (In contrast, the popular queries benefit significantly from the many clickthroughs from the users that the system can collect.) To address this problem, we may introduce new moves to allow a user to provide "explanatory feedback" when the search results are poor and the user can't easily reformulate the query either. An example of such a move can be to allow a user to click on a button to select from a menu with choices such as 1) I want documents similar to this one except for not matching "X" where the user would type in "X," or 2) I want documents similar to this one, but also further matching "Y" where the user would type in "Y." Such a move is very natural for a user and does not require much effort from the user, but can be very effective for helping the system accurately learn the user's information need.

4.2 Instantiation of user model M

As already discussed, formally modeling a user's state through the variable M is a very important component in the proposed framework, and a major novelty of the framework over existing ways to model the retrieval process. In general, the formal user model M is intended to capture all the essential knowledge about a user's status for the purpose of optimizing the system's response especially for personalization [21]. We can thus imagine many components that can be included in M. The most important component is the model of the current information need of the user. This component is usually implemented as a term vector [18] or a word distribution in the language model [14] in a current search engine, allowing for matching with a document vector or document word distribution to generate a score of relevance. However, more sophisticated representation is clearly possible, including, e.g., tracking multiple subtopics of interest to the user or negative models for information not interesting to the user.

Another component that is very easy to maintain is a model of the information items that have already been seen or viewed by the user. This component is important for assessing the novelty of any information item that may be displayed to the user in the future. One can also model the reading level of a user, which can be important if the user is an elementary school student since it helps the system avoid presenting topically relevant materials that are beyond the reading level of the user.

The user model M can further include information about the user's browsing behavior. For example, does the user tend to view many results on a page or even go to the next page, or the user often just views the top

three to five results? Having this knowledge clearly helps optimizing the design of the interface shown to the user when responding to the user's query. Of course, M can also include many other aspects about the user such as the user's task (which can often be inferred based on the multiple queries entered by the user; for example finding flight tickets and booking hotel can suggest a task of travel). The model can even include an estimate of the patience level of the user, which can also affect the optimization of the system's response.

It is easy to see that M can potentially include all kinds of findings about user preferences and behaviors from user studies that may be relevant to optimizing a search task. This user model thus provides a formal and principled way to integrate relevant findings by human-computer interaction researchers into a search engine to optimize its utility. The user modeling component of the framework can also be regarded as a way to formalize some existing theories about user modeling in information retrieval, notably the Anomalous State of Knowledge (ASK) theory [3] and the Cognitive Information Retrieval Theory [11].

4.3 Inference of user model

The inference of user model M is based on the posterior distribution $p(M|U, H, A_t, C, S)$, which enables inference of M based on everything the system has available so far about the user and his/her interactions. The instantiation of this distribution can be based on findings from user studies or machine learning using user interaction log data for training. The current search engines mostly focused on estimating and updating the information need model, which is only part of the general user model M. Future search engines must also infer and update many other variables about the user such as the inference about the user's task, exploratory vs. fixed item search, reading level, and browsing behavior. Some existing work can be leveraged for making such inferences (e.g., reading level [5], modeling decision point [22]).

4.4 Instantiation of loss function

In general, the loss function $L(r, M, S)$ should combine measures of 1) *Utility* of response r for a user modeled as M to finish the user's task in situation S; 2) *Effort* of a user modeled as M in situation S; and 3) *Cost* of system performing r. The tradeoff between these three aspects would inevitably vary across users and situations. The utility of response r can be defined as a sum of the *immediate* utility of r to the user and the *future* utility derived from future interactions of the user enabled by response r, which depends on the user's interaction behavior.

Formalization of the utility function requires research on evaluation, task modeling, and user behavior modeling. The traditional evaluation measures, such as Mean Average Precision (MAP) and Normalized Discounted Cumulative Gain (NDCG), tend to use very simple user behavior model (i.e., sequential browsing) and use a straightforward combination of effort and utility. They would need to be extended to incorporate more sophisticated user behavior models (e.g., in the line of [6, 19, 2]) to more accurately model the utility and user effort and enable more flexible tradeoff.

The system operation cost can be modeled based on the expected consumption of computing resources such as CPU time, memory, and disk space by computing the response r. In a more general sense, the cost may even include the cost on hiring humans to help answer a query (crowdsourcing), which may then justify a response r that is based on human computation, rather than a retrieval algorithm.

4.5 The Interface card model

The interface card model [27] is a general instantiation of the game-theoretic framework where a system response is defined very generally as any interaction interface, and the retrieval decision is thus reduced to the decision on which interface to choose. The objective function to be optimized reflects the expected surplus of presenting a particular interaction interface which is calculated based on the reward (gain) of relevant information obtained by

the user adjusted by the cost due to the effort that the user has to make in order to gain the relevant information. The expectation is taken with respect to a distribution of all possible actions that a user can take. With further assumptions, the model is shown to be able to generalize PRP by relaxing both assumptions made in PRP (i.e., sequential browsing and independent utility of documents). Moreover, the model is shown to be able to automatically determine the optimal layout of a navigation interface in adaptation to the system's confidence in inference about the user's information need and the screen size of the display. This is a very promising model because it opens up a new direction of computationally optimizing the layout of an interface, which can have broad impact beyond a retrieval system and may be especially useful for optimizing the interface design for mobile phone users where the screen size is very small, thus requiring efficient use of the limited space.

5 Conclusions

In this paper, we addressed some fundamental questions about how to optimize text data retrieval systems in a general way and proposed to define the retrieval task as having the system to play a cooperative retrieval game with the user with a shared objective to complete the user's task with minimum overall user effort and minimum system operation cost. We used Bayesian decision theory to formally frame the problem as a statistical decision problem with a formal model of users as a key component for optimizing retrieval decisions. The game-theoretic framework can potentially integrate research in user studies, evaluation, retrieval models, and efficient implementation of text retrieval systems in a single unified principled framework and also serves as a roadmap for identifying unexplored important IR research topics. The two important benefits of the framework are 1) natural optimization of the utility over an entire session instead of that on a single query and 2) optimization of the collaboration of machines and users (thus maximizing collective intelligence). We also briefly discussed how to instantiate all the major components of the framework so as to derive more specific models that can be implemented in a search engine and introduced the Interface Card Model as a specific example which can optimize the interface design for navigation into relevant items. We show that the game-theoretic framework can model user actions in a very general way and at different levels of granularity.

To fully implement a system based on such a new framework, there are obviously many challenges to solve, and we would need to integrate research in multiple areas including 1) formal modeling of users and tasks, 2) modeling and measuring system cost, 3) machine learning, particularly reinforcement learning to enable the system to effectively update user model, 4) efficient algorithms to enable fast response by the system, and 5) evaluation methodology for evaluating such an interactive system. While the framework was proposed for the problem of text data retrieval where involvement of users in the loop is especially important, we believe that the general idea of the framework and many components may also be broadly applicable to other interactive systems where optimization of human-machine collaboration is important.

References

[1] L. Azzopardi. Modelling interaction with economic models of search. In *Proceedings of ACM SIGIR 2014*, pages 3–12, New York, NY, USA, 2014. ACM.

[2] F. Baskaya, H. Keskustalo, and K. Järvelin. Modeling behavioral factors ininteractive information retrieval. In *Proceedings of ACM CIKM 2013*, CIKM '13, pages 2297–2302, New York, NY, USA, 2013. ACM.

[3] N. J. Belkin. Anomalous states of knowledge as a basis for information retrieval. *The Canadian Journal of Information Science*, (5):133–143, 1980.

[4] N. J. Belkin. Intelligent information retrieval: Whose intelligence? In *Proceedings of the Fifth International Symposium for Information Science*, 1996.

[5] K. Collins-Thompson, P. N. Bennett, R. W. White, S. de la Chica, and D. Sontag. Personalizing web search results by reading level. In *Proceedings of ACM CIKM 2011*, pages 403–412, 2011.

[6] A. P. de Vries, G. Kazai, and M. Lalmas. Tolerance to irrelevance: A user-effort evaluation of retrieval systems without predefined retrieval unit. In *Coupling Approaches, Coupling Media and Coupling Languages for Information Retrieval - RIAO 2004*, pages 463–473, 2004.

[7] F. Diaz. Integration of news content into web results. In *Proceedings of the Second ACM International Conference on Web Search and Data Mining*, WSDM '09, pages 182–191, New York, NY, USA, 2009. ACM.

[8] N. Fuhr. A probability ranking principle for interactive information retrieval. *Inf. Retr.*, 11(3):251–265, June 2008.

[9] K. Hofmann. Fast and reliable online learning to rank for information retrieval, 2013. Doctoral Dissertation.

[10] K. Hofmann, S. Whiteson, and M. de Rijke. Balancing exploration and exploitation in learning to rank online. In *Proceedings of ECIR 2011*, pages 251–263, 2011.

[11] P. Ingwersen. Cognitive perspectives of information retrieval interaction: elements of a cognitive ir theory. *Journal of Documentation*, 52:3–50, 1996.

[12] X. Jin, M. Sloan, and J. Wang. Interactive exploratory search for multi page search results. In *Proceedings of the 22Nd International Conference on World Wide Web*, WWW '13, pages 655–666, New York, NY, USA, 2013. ACM.

[13] J. Luo, S. Zhang, and H. Yang. Win-win search: Dual-agent stochastic game in session search. In *Proceedings of ACM SIGIR 2014*, pages 587–596, New York, NY, USA, 2014. ACM.

[14] J. Ponte and W. B. Croft. A language modeling approach to information retrieval. In *Proceedings of the ACM SIGIR'98*, pages 275–281, 1998.

[15] S. E. Robertson. The probability ranking principle in ıᵲ *Journal of Documentation*, 33(4):294–304, Dec. 1977.

[16] G. Salton and M. McGill. *Introduction to Modern Information Retrieval*, McGraw-Hill, New York, 1983

[17] X. Shen, B. Tan, and C. Zhai. Implicit user modeling for personalized search. In *Proceedings of CIKM 2005*, 2005.

[18] A. Singhal and C. Buckley and M. Mitra. Pivoted document length normalization, In *Proceedings of ACM SIGIR 1996*, pages 21–29, 1996.

[19] M. D. Smucker and C. L. Clarke. Time-based calibration of effectiveness measures. In *Proceedings of ACM SIGIR 2012*, pages 95–104, New York, NY, USA, 2012. ACM.

[20] K. Sparck Jones and P. Willett, editors. *Readings in Information Retrieval*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1997.

[21] J. Teevan, S. T. Dumais, and E. Horvitz. Potential for personalization. *ACM Trans. Comput.-Hum. Interact.*, 17(1):4:1–4:31, Apr. 2010.

[22] P. Thomas, A. Moffat, P. Bailey, and F. Scholer. Modeling decision points in user search behavior. In *Proceedings of the 5th Information Interaction in Context Symposium*, IIiX '14, pages 239–242, New York, NY, USA, 2014. ACM.

[23] C. J. van Rijsbergen. *Information Retrieval*, Butterworths, London, 1979.

[24] C. Zhai and J. Lafferty. A risk minimization framework for information retrieval. *Information Processing and Management*, 42(1), pages 31-55, 2006.

[25] C. Zhai. *Statistical Language Models for Information Retrieval*. Synthesis Lectures on Human Language Technologies. Morgan & Claypool Publishers, 2008.

[26] C. Zhai and S. Massung. *Text Data Management and Analysis: A Practical Introduction to Information Retrieval and Text Mining*. ACM and Morgan & Claypool Publishers, 2016.

[27] Y. Zhang and C. Zhai. Information retrieval as card playing: A formal model for optimizing interactive retrieval interface. In *Proceedings of ACM SIGIR 2015*, pages 685–694, New York, NY, USA, 2015. ACM.

Neural Enquirer: Learning to Query Tables in Natural Language

Pengcheng Yin
Language Technologies Institute
pcyin@cs.cmu.edu

Zhengdong Lu
Noah's Ark Lab, Huawei Technologies
Lu.Zhengdong@huawei.com

Hang Li
Noah's Ark Lab, Huawei Technologies
HangLi.HL@huawei.com

Ben Kao
The University of Hong Kong
kao@cs.hku.hk

Abstract

We propose NEURAL ENQUIRER *— a neural network architecture for answering natural language (NL) questions based on a knowledge base (KB) table. Unlike existing work on end-to-end training of semantic parsers [13, 12],* NEURAL ENQUIRER *is fully "neuralized": it finds distributed representations of queries and KB tables, and executes queries through a series of neural network components called "executors". Executors model query operations and compute intermediate execution results in the form of table annotations at different levels.* NEURAL ENQUIRER *can be trained with gradient descent, with which the representations of queries and the KB table are jointly optimized with the query execution logic. The training can be done in an end-to-end fashion, and it can also be carried out with stronger guidance, e.g., step-by-step supervision for complex queries.* NEURAL ENQUIRER *is one step towards building neural network systems that can understand natural language in real-world tasks. As a proof-of-concept, we conduct experiments on a synthetic QA task, and demonstrate that the model can learn to execute reasonably complex NL queries on small-scale KB tables.*

1 Introduction

Natural language dialogue and question answering often involve querying a knowledge base [14, 3]. The traditional approach involves two steps: First, a given query $\tilde{Q}$ is semantically parsed into an "executable" representation, which is often expressed in certain logical form $\tilde{Z}$ (e.g., SQL-like queries). Second, the representation is executed against a knowledge base from which an answer is obtained. For queries that involve complex semantic constraints and logic (e.g., "*Which city hosted the longest Olympic Games before the Games in Beijing?*"), semantic parsing and query execution become extremely complex. For example, carefully hand-crafted features and rules are needed to correctly parse a complex query into its logical form (see example shown in the lower-left corner of Figure 1). This complexity often results in poor accuracy of the system. To partially overcome this difficulty, recent works [7, 10, 13] attempt to "backpropagate" query execution results to revise the semantic representation of a query, which is an example of learning from grounding [6, 9]. This approach, however, is greatly hindered by the fact that traditional semantic parsing mostly involves rule-based features and symbolic manipulation, and is subject to intractable search space incurred by the great flexibility of natural language.

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

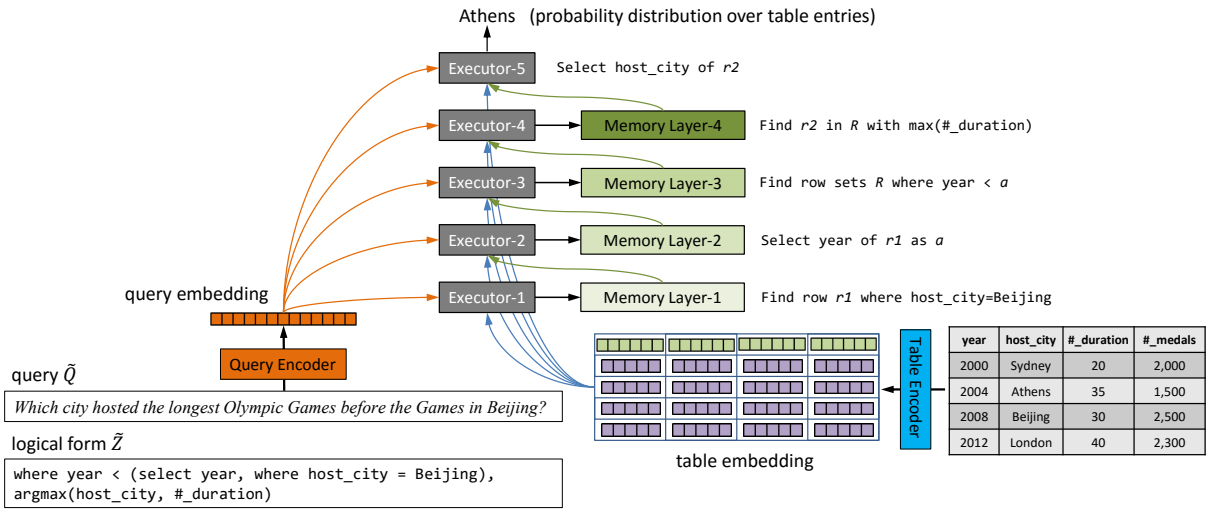


Figure 1: An overview of NEURAL ENQUIRER with five executors

Neural network-based models have enjoyed much successes in natural language processing, particularly in machine translation and syntactic parsing. These successes are attributable to direct and strong supervision. The recent work on learning to execute simple program codes with LSTM [17] pioneers in the direction on learning to parse structured objects through executing it in a purely neural way, while the more recent work on Neural Turing Machines (NTMs) [8] introduces more modeling flexibility by equipping the LSTM with external memory and various means for interacting with it.

Inspired by the above-mentioned research, we aim to design a neural network system that learns to understand queries and execute them on a knowledge base table from examples of queries and answers. We propose NEURAL ENQUIRER, a fully neuralized, end-to-end differentiable system that jointly models semantic parsing and query execution. NEURAL ENQUIRER encodes queries and KB tables into distributed representations, and executes compositional queries against the KB through a series of differentiable executors. The model is trained using query-answer pairs, where the distributed representations of queries and the KB are optimized together with the query execution logic in an end-to-end fashion. As the first step along this line of research, we evaluate NEURAL ENQUIRER using a synthetic question-answering task as a proof-of-concept, and demonstrate that our proposed model is capable of learning to execute complex compositional natural language questions on small-scale KB tables.

2 Model

Following [13], we study the problem of question answering on a single KB table. Specifically, given an NL query Q and a KB table $\mathcal{T}$, NEURAL ENQUIRER executes Q against $\mathcal{T}$ and outputs a ranked list of answers. The execution is done by first using *Encoders* to encode the query and table into distributed representations, which are then sent to a cascaded pipeline of *Executors* to derive the answer. Figure 1 gives an illustrative example (with five executors). It consists of the following components:

Query Encoder (Section 2.1), which abstracts the semantics of an NL query and encodes it into a query embedding.

Table Encoder (Section 2.2), which derives a table embedding by encoding entries in the table into distributed vectors.

Executor (Section 2.3), which executes the query against the table and outputs *annotations* that encode intermediate execution results. Annotations are stored in the memory of each layer to be accessed by the executor

of the next layer. Since complex compositional queries can be answered in multiple steps of computation, each executor models a specific type of operation conditioned on the query. Figure 1 illustrates the operation each executor is assumed to perform in answering the example query $\tilde{Q}$. Different from classical semantic parsing approaches which require a predefined set of all possible logical operations, NEURAL ENQUIRER *learns* the logic of executors via end-to-end training using query-answer pairs. By stacking several executors, our model is able to answer complex queries that involve multiple steps of computation.

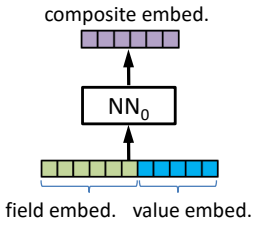
2.1 Query Encoder

Query Encoder converts an NL query Q into a query embedding $\mathbf{q} \in \mathbb{R}^{d_{\mathcal{Q}}}$. Let $\{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_T\}$ be the embeddings of words in Q, where $\mathbf{x}_t \in \mathbb{R}^{d_{\mathcal{W}}}$ is from an embedding matrix $\mathbf{L}$. We employ a bidirectional Gated Recurrent Unit (GRU) [2] to summarize the sequence $\{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_T\}$ in forward and reverse orders. $\mathbf{q}$ is formed by concatenating the last hidden states in the two directions.

It is worth noting that Query Encoder can find the representation of a rather general class of symbol sequences, agnostic to the actual representation of the query (e.g., natural language, SQL, etc). The model is able to learn the semantics of input queries through end-to-end training, making it a generic model for query understanding and query execution.

2.2 Table Encoder

Table Encoder converts a KB table $\mathcal{T}$ into a distributed representation, which is used as an input to executors. Suppose $\mathcal{T}$ has M rows and N columns. In our model, the n-th column is associated with a *field name* (e.g., `host_city`). Each cell value is a word (e.g., *Beijing*) in the vocabulary. We use w_{mn} to denote the cell value in row m column n, and $\mathbf{w}_{mn}$ to denote its embedding. Let $\mathbf{f}_n$ be the embedding of the field name for column n. For each entry (cell) w_{mn}, Table Encoder computes a $\langle$field, value$\rangle$ composite embedding $\mathbf{e}_{mn} \in \mathbb{R}^{d_{\mathcal{E}}}$ by fusing $\mathbf{f}_n$ and $\mathbf{w}_{mn}$ using a single-layer Neural Network:



$$\mathbf{e}_{mn} = \mathsf{NN}_0(\mathbf{f}_n, \mathbf{w}_{mn}) = \tanh(\mathbf{W} \cdot [\mathbf{f}_n; \mathbf{w}_{mn}] + \mathbf{b}),$$

where $[\cdot;\cdot]$ denotes vector concatenation. The output of Table Encoder is an $M \times N \times d_{\mathcal{E}}$ tensor that consists of $M \times N$ embeddings, each of length $d_{\mathcal{E}}$.

We remark that our Table Encoder is different from classical knowledge embedding models (e.g., TransE [5]). While traditional methods learn the embeddings of entities (cell values) and relations (field names) in an unsupervised fashion via minimizing certain reconstruction errors, embeddings in Table Encoder are optimized via supervised learning in end-to-end QA tasks.

2.3 Executor

NEURAL ENQUIRER executes an input query on a KB table through layers of execution. Each layer consists of an executor that, after learning, performs certain operation (e.g., `select`, `max`) relevant to the input query. An executor outputs intermediate execution results, referred to as *annotations*, which are saved in the external memory of the executor. A query is executed sequentially through a stack of executors. Such a cascaded architecture enables the model to answer complex, compositional queries. An example is given in Figure 1 in which descriptions of the operation each executor is assumed to perform for the query $\tilde{Q}$ are shown. We will demonstrate in Section 4 that the model is capable of learning the operation logic of each executor via end-to-end training.

As illustrated in Figure 2, an executor at Layer-ℓ (denoted as Executor-ℓ) consists of two major neural network components: a *Reader* and an *Annotator*. The executor processes a table row-by-row. The Reader reads

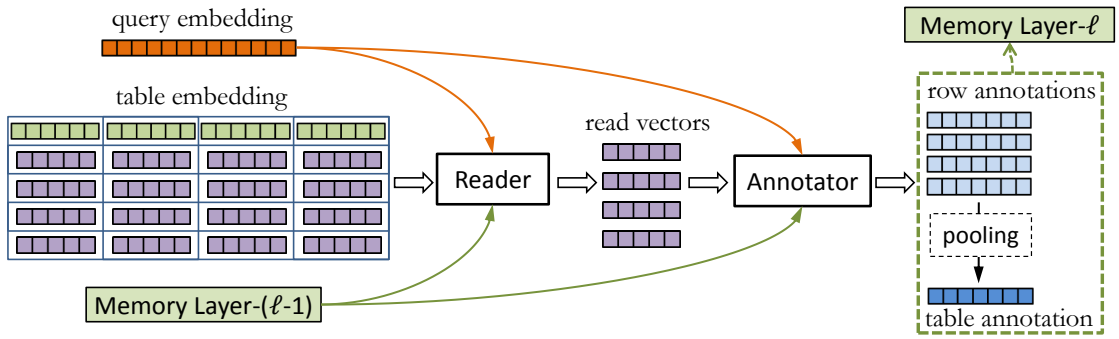


Figure 2: Overview of an Executor-ℓ

in data from each row m in the form of a *read vector* $\mathbf{r}_m^\ell$, which is then sent to the Annotator to perform the actual execution. The output of the Annotator is a *row annotation* $\mathbf{a}_m^\ell$, which captures the row-wise local computation result. Once all row annotations are obtained, Executor-ℓ generates a *table annotation* $\mathbf{g}^\ell$ to summarize the global computation result on the whole table by pooling all row annotations. All the row and table annotations are saved in the memory of Layer-ℓ: $\mathcal{M}^\ell = \{\mathbf{a}_1^\ell, \mathbf{a}_2^\ell, \ldots, \mathbf{a}_M^\ell, \mathbf{g}^\ell\}$. Intuitively, row annotations handle operations that require only row-wise, local information (e.g., `select`, `where`), while table annotations model superlative operations (e.g., `max`, `min`) by aggregating table-wise, global execution results. A combination of row and table annotations enables the model to perform a wide variety of query operations in real world scenarios.

2.3.1 Reader

As illustrated in Figure 3, for the m-th row with N ⟨field, value⟩ composite embeddings $\mathcal{R}_m = \{\mathbf{e}_{m1}, \mathbf{e}_{m2}, \ldots, \mathbf{e}_{mN}\}$, the Reader fetches a read vector $\mathbf{r}_m^\ell$ from $\mathcal{R}_m$ via an attentive reading operation:

$$\mathbf{r}_m^\ell = f_{\mathrm{R}}^\ell(\mathcal{R}_m, \mathcal{F}_\mathcal{T}, \mathbf{q}, \mathcal{M}^{\ell-1}) = \sum_{n=1}^{N} \tilde{\omega}(\mathbf{f}_n, \mathbf{q}, \mathbf{g}^{\ell-1}) \mathbf{e}_{mn}$$

where $\mathcal{M}^{\ell-1}$ denotes the content of memory Layer-$(\ell-1)$, and $\mathcal{F}_\mathcal{T} = \{\mathbf{f}_1, \mathbf{f}_2, \ldots, \mathbf{f}_N\}$ is the set of field name embeddings. $\tilde{\omega}(\cdot)$ is the normalized attention weights given by:

$$\tilde{\omega}(\mathbf{f}_n, \mathbf{q}, \mathbf{g}^{\ell-1}) = \frac{\exp(\omega(\mathbf{f}_n, \mathbf{q}, \mathbf{g}^{\ell-1}))}{\sum_{n'=1}^{N} \exp(\omega(\mathbf{f}_{n'}, \mathbf{q}, \mathbf{g}^{\ell-1}))} \tag{10}$$

where $\omega(\cdot)$ is modeled as a Deep Neural Network (denoted as $\mathsf{DNN}_1^{(\ell)}$). Since each executor models a specific type of computation, it should only attend to a subset of entries that are pertinent to its execution. This is modeled by the Reader. Our approach is related to the content-based addressing of Neural Turing Machines [8] and the attention mechanism in neural machine translation models [2].

2.3.2 Annotator

The Annotator of Executor-ℓ computes row and table annotations based on read vectors fetched by the Reader. The results are stored in the ℓ-th memory layer $\mathcal{M}^\ell$ accessible to Executor-$(\ell+1)$. The last executor is the only exception, which outputs the final answer.

[Row annotations] Capturing row-wise execution result, the annotation $\mathbf{a}_m^\ell$ for row m in Executor-ℓ is given by

$$\mathbf{a}_m^\ell = f_{\mathrm{A}}^\ell(\mathbf{r}_m^\ell, \mathbf{q}, \mathcal{M}^{\ell-1}) = \mathsf{DNN}_2^{(\ell)}([\mathbf{r}_m^\ell; \mathbf{q}; \mathbf{a}_m^{\ell-1}; \mathbf{g}^{\ell-1}]). \tag{11}$$

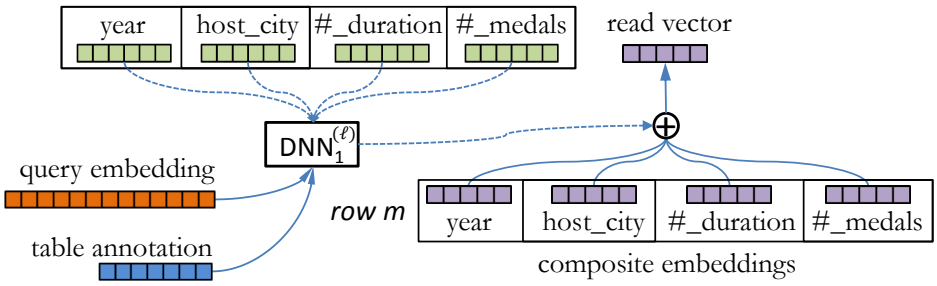


Figure 3: Illustration of the Reader in Executor-ℓ

DNN$_2^{(\ell)}$ fuses the corresponding read vector $\mathbf{r}_m^\ell$, the results saved in the previous memory layer (row and table annotations $\mathbf{a}_m^{\ell-1}$, $\mathbf{g}^{\ell-1}$), and the query embedding $\mathbf{q}$. Intuitively, row annotation $\mathbf{a}_m^{\ell-1}$ and table annotation $\mathbf{g}^{\ell-1}$ summarize the local and global status of execution up to Layer-$(\ell-1)$, respectively. DNN$_2^{(\ell)}$ then performs the actual query execution by combing these annotations with the read vector and query embedding, and outputs a row annotation $\mathbf{a}_m^\ell$ that encodes the local execution result on row m.

[Table annotations] Capturing global execution state, a table annotation summarizes all row annotations via a global max pooling operation:

$$\mathbf{g}^\ell = f_{\text{MAXPOOL}}(\mathbf{a}_1^\ell, \mathbf{a}_2^\ell, \ldots, \mathbf{a}_M^\ell) = [g_1, g_2, \ldots, g_{d_\mathcal{G}}]^\top \tag{12}$$

where $g_k = \max(\{\mathbf{a}_1^\ell(k), \mathbf{a}_2^\ell(k), \ldots, \mathbf{a}_M^\ell(k)\})$ is the maximum value among the k-th elements of all row annotations.

2.3.3 Last Layer Executor

Instead of computing annotations based on read vectors, the last executor in NEURAL ENQUIRER directly outputs the probability of an entry w_{mn} in table $\mathcal{T}$ being the answer a:

$$p(a = w_{mn}|Q, \mathcal{T}) = \frac{\exp(f_{\text{ANS}}^\ell(\mathbf{e}_{mn}, \mathbf{q}, \mathbf{a}_m^{\ell-1}, \mathbf{g}^{\ell-1}))}{\sum_{m'=1,n'=1}^{M,N} \exp(f_{\text{ANS}}^\ell(\mathbf{e}_{m'n'}, \mathbf{q}, \mathbf{a}_{m'}^{\ell-1}, \mathbf{g}^{\ell-1}))} \tag{13}$$

where $f_{\text{ANS}}^\ell(\cdot)$ is modeled as a DNN (DNN$_3^{(\ell)}$). Note that the last executor, which is devoted to returning answers, could still carry out execution in DNN$_3^{(\ell)}$.

3 Learning

NEURAL ENQUIRER can be trained in an *end-to-end* (N2N) fashion on QA tasks. During training, both the representations of queries and tables, as well as the execution logic captured by the weights of executors are learned. Given a set of $N_\mathcal{D}$ query-table-answer triples $\mathcal{D} = \{(Q^{(i)}, \mathcal{T}^{(i)}, y^{(i)})\}$, we learn the model parameters by maximizing the log-likelihood of gold-standard answers:

$$\mathcal{L}_{\text{N2N}}(\mathcal{D}) = \sum_{i=1}^{N_\mathcal{D}} \log p(a = y^{(i)}|Q^{(i)}, \mathcal{T}^{(i)}) \tag{14}$$

In end-to-end training, each executor discovers its operation logic from training data in a purely data-driven fashion, which could be difficult for complex queries requiring four or five sequential operations.

year	host_city	#_participants	#_medals	#_duration	#_audience	host_country	GDP	country_size	population
2008	Beijing	4,200	2,500	30	67,000	China	2,300	960	130

Figure 4: An example table in the synthetic QA task (only one row shown)

Query Type	Example Queries (Q) with Annotated SQL-like Logical Forms (Z)
SELECT_WHERE	▷ Q: *How many people participated in the game in Beijing?* Z: `select #_participants, where host_city = Beijing` ▷ Q: *In which country was the game hosted in 2012?* Z: `select host_country, where year = 2012`
SUPERLATIVE	▷ Q: *When was the lastest game hosted?* Z: `argmax(host_city, year)` ▷ Q: *How big is the country which hosted the shortest game?* Z: `argmin(country_size, #_duration)`
WHERE_SUPERLATIVE	▷ Q: *How long is the game with the most medals that has fewer than 3,000 participants?* Z: `where #_participants < 3,000, argmax(#_duration, #_medals)` ▷ Q: *How many medals are in the first game after 2008?* Z: `where #_year > 2008, argmin(#_medals, #_year)`
NEST	▷ Q: *Which country hosted the longest game before the game in Athens?* Z: `where year<(select year,where host_city=Athens),argmax(host_country,#_duration)` ▷ Q: *How many people watched the earliest game that lasts for more days than the game in 1956?* Z: `where #_duration<(select #_duration,where year=1956),argmin(#_audience,#_year)`

Table 1: Example queries in our synthetic QA task

This can be alleviated by softly guiding the learning process via controlling the attention weights $\tilde{w}(\cdot)$ in Eq. (10). By enforcing $\tilde{w}(\cdot)$ to bias towards a field pertaining to a specific operation, we can "coerce" the executor to figure out the logic of this operation relative to the field. For example, for Executor-1 in Figure 1, by biasing the attention weight of the `host_city` field towards 1.0, only the value of `host_city` will be fetched and sent to the Annotator. In this way we can "force" the executor to learn the `where` operation to find the row whose `host_city` is *Beijing*. This method will be referred to as *step-by-step* (SbS) training. Formally, this is done by introducing additional supervision signal to Eq. (14):

$$\mathcal{L}_{\text{SbS}}(\mathcal{D}) = \sum_{i=1}^{N_{\mathcal{D}}} \Big(\log p(a = y^{(i)} | Q^{(i)}, \mathcal{T}^{(i)}) + \alpha \sum_{\ell=1}^{L-1} \log \tilde{w}(\mathbf{f}^{\star}_{i,\ell}, \cdot, \cdot) \Big) \tag{15}$$

where α is a tuning weight, and L is the number of executors. $\mathbf{f}^{\star}_{i,\ell}$ is the embedding of the field known *a priori* to be used by Executor-ℓ in answering the i-th example.

4 Experiments

In this section we evaluate NEURAL ENQUIRER on synthetic QA tasks with NL queries of varying compositional depths.

4.1 Synthetic QA Task

We present a synthetic QA task with a large number of QA examples at various levels of complexity to evaluate the performance of NEURAL ENQUIRER. Starting with "artificial" tasks accelerates the development of novel deep models [15], and has gained increasing popularity in recent research on modeling symbolic computation using DNNs [8, 17].

Our synthetic dataset consists of query-table-answer triples $\{(Q^{(i)}, \mathcal{T}^{(i)}, y^{(i)})\}$. To generate a triple, we first randomly sample a table $\mathcal{T}^{(i)}$ of size 10×10 from a synthetic schema of Olympic Games. The cell values of $\mathcal{T}^{(i)}$ are drawn from a vocabulary of 120 location names and 120 numbers. Figure 4 gives an example table. Next, we

	MIXED-25K				MIXED-100K		
	SEMPRE	N2N	SbS	N2N-OOV	N2N	SbS	N2N-OOV
SELECT_WHERE	93.8%	96.2%	99.7%	90.3%	99.3%	100.0%	97.6%
SUPERLATIVE	97.8%	98.9%	99.5%	98.2%	99.9%	100.0%	99.7%
WHERE_SUPERLATIVE	34.8%	80.4%	94.3%	79.1%	98.5%	99.8%	98.0%
NEST	34.4%	60.5%	92.1%	57.7%	64.7%	99.7%	63.9%
Overall Accuracy	65.2%	84.0%	96.4%	81.3%	90.6%	99.9%	89.8%

Table 2: Accuracies on MIXED datasets

sample a query $Q^{(i)}$ generated using NL templates, and obtain its gold-standard answer $y^{(i)}$ on $\mathcal{T}^{(i)}$. Our task consists of four types of NL queries, with examples given in Table 1. We also give the logical form template for each type of query. The templates define the semantics and compositionality of queries. We generate queries at various compositional depths, ranging from simple SELECT_WHERE queries to more complex NEST ones. This makes the dataset have similar complexity as a real-world one, except for the relatively small vocabulary. The queries are flexible enough to involve complex matching between NL phrases and logical constituents, which makes query understanding nontrivial: (1) the same field is described by different NL phrases (e.g., *"How big is the country ..."* and *"What is the size of the country ..."* for the `country_size` field); (2) different fields may be referred to by the same NL pattern (e.g, *"in China"* for `host_country` and *"in Beijing"* for `host_city`); (3) simple NL constituents may be grounded to complex logical operations (e.g., *"after the Games in Beijing"* implies comparing between the values of `year` fields).

To simulate the read-world scenario where queries of various types are issued to the model, we construct two MIXED datasets, with $25K$ and $100K$ training examples respectively, where four types of queries are sampled with the ratio $1:1:1:2$. Both datasets share the same testing set of $20K$ examples, $5K$ for each type of query. We enforce that no tables and queries are shared between training/testing sets.

4.2 Setup

[Tuning] We adopt a model with five executors. The lengths of hidden states for GRU and DNNs are 150, 50. The numbers of layers for $\mathsf{DNN}_1^{(\ell)}$, $\mathsf{DNN}_2^{(\ell)}$ and $\mathsf{DNN}_3^{(\ell)}$ are 2, 3, 3. The length of word embeddings and annotations is 20. α is 0.2. We train the model using ADADELTA [18] on a Tesla K40 GPU. The training converges fast within 2 hours.
[Metric] We evaluate in terms of accuracy, defined as the fraction of correctly answered queries.
[Models] We compare the results of the following settings:

- **Sempre** [13] is a state-of-the-art semantic parser and serves as the baseline;
- **N2N**, our model trained using *end-to-end* setting (Sec 4.3);
- **SbS**, our model trained using *step-by-step* setting (Sec 4.4);
- **N2N-OOV**, a variant of the N2N model to deal with out-of-vocabulary words (Sec 4.5)

4.3 End-to-End Evaluation

Table 2 summarizes the results of SEMPRE and NEURAL ENQUIRER under different settings. We show both the individual performance for each query type and the overall accuracy. We evaluate SEMPRE only on MIXED-25K because of its long training time even on this small dataset (about 3 days).

In this section we discuss the results under end-to-end (N2N) training setting. On MIXED-25K, the relatively low performance of SEMPRE indicates that our QA task, although synthetic, is highly nontrivial. Surprisingly, NEURAL ENQUIRER outperforms SEMPRE on all query types, with a marginal gain on *simple* queries

Q_1: *How long was the Games with the most medals that had fewer than 3,000 participants?*

Z_1: `where #_participants < 3,000, argmax(#_duration, #_medals)`

Q_2: *Which country hosted the longest Games before the Games in Athens?*

Z_2: `where year < (select year, where host_city = Athens), argmax(host_country, #_duration)`

Figure 5: Weights visualization of queries Q_1 and Q_2

(SELECT_WHERE, SUPERLATIVE), and significant improvement on *complex* queries (WHERE_SUPERLATIVE, NEST). We posit that the low performance of SEMPRE on complex queries is likely due to the intractable search space incurred by the flexibility of its float parsing algorithm. On MIXED-100K, our model registers an overall accuracy of 90.6%. These results show that in our QA task, NEURAL ENQUIRER is very effective in answering compositional NL queries, especially those with complex semantics compared with the state-of-the-art system.

To further understand why our model is capable of answering compositional queries, we study the attention weights of Readers (Eq. 10) for intermediate executors, and the answer probability (Eq. 13) the last executor outputs for each table entry. These statistics are obtained on MIXED-100K. We sample two queries (Q_1 and Q_2) in the testing set that our model answers correctly and visualize their corresponding values in Figure 5. To better understand the query execution process, we also give the logical forms (Z_1 and Z_2) of the two queries. Note that the logical forms are just for reference purpose and unknown by the model. We find that each executor actually *learns* its execution logic in N2N training, which is in accordance with our assumption. The model executes Q_1 in three steps, with each of the last three executors performs a specific type of operation. For each row, Executor-3 takes the value of the `#_participants` field as input, while Executor-4 attends to the `#_medals` field. Finally, Executor-5 outputs a high probability for the `#_duration` field in the 3-rd row. The attention weights for Executor-1 and Executor-2 appear to be meaningless because Q_1 requires only three steps of execution, and the model learns to defer the meaningful execution to the last three executors. Comparing with the logical form Z_1 of Q_1, we can deduce that Executor-3 "executes" the `where` clause in Z_1 to find row sets R satisfying the condition, and Executor-4 performs the first part of `argmax` to find the row $r \in R$ with the maximum value of `#_medals`, while Executor-5 outputs the value of `#_duration` in r.

Compared with the relatively simple Q_1, Q_2 is more complicated. According to Z_2, Q_2 involves an additional nest sub-query to be solved by two extra executors, and requires a total of five steps of execution. The last three executors function similarly as in answering Q_1, yet the execution logic for the first two executors (devoted to solving the sub-query) is a bit obscure, since their attention weights are scattered instead of being perfectly centered on the ideal fields as highlighted in red dashed rectangles. We posit that this is because during the end-to-end training, the supervision signal propagated from the top layer has decayed along the long path down to the first two executors, which causes vanishing gradients.

4.4 With Additional Step-by-Step Supervision

To alleviate the vanishing gradient problem when training on complex queries like Q_2, we train the model using step-by-step (SbS) setting (Eq. 15), where we encourage each intermediate executor to attend to the field that is known *a priori* to be relevant to its execution logic. Results are shown in Table 2 (column SbS). With stronger supervision signal, the model significantly outperforms the N2N setting, and achieves perfect accuracy on MIXED-100K. This shows that NEURAL ENQUIRER is capable of leveraging the additional supervision signal given to intermediate layers in SbS training. Let us revisit the query Q_2 in SbS setting. In contrast to the result in N2N setting (Figure 5) where the attention weights for the first two executors are obscure, now the weights are perfectly skewed towards each relevant field with a value of 1.0, which corresponds with the highlighted ideal weights.

4.5 Dealing with Out-Of-Vocabulary Words

One of the major challenges for applying neural network models to NLP applications is to deal with out-of-vocabulary (OOV) words (e.g., new entities for QA). Surprisingly, we find that a simple variant of NEURAL ENQUIRER is able to handle unseen entities almost without loss of accuracy.

Specifically, we divide words in the vocabulary into *entity* words and *operation* words. Embeddings of entity words (e.g., *Beijing*) function like indices to facilitate the matching between the entities in queries and tables during query execution, and therefore are not updated once randomly initialized; while those of operation words, i.e., all non-entity words (e.g., numbers, *longest*, *before*, etc), carry semantic meanings relevant to execution and will be optimized in training. Therefore, after randomly initializing the embedding matrix $\mathbf{L}$, we only update the embeddings of operation words in training, while keeping those of entity words unchanged. To evaluate the model we modify the queries in the testing set to replace all entity words (i.e., all country and city names) with those unseen in training. Results obtained using N2N training, reported in Table 2 (column N2N-OOV), show that the model yields performance comparable with non-OOV settings.

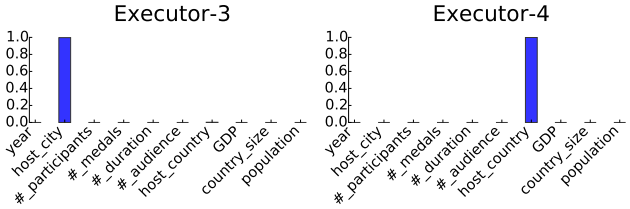


Figure 6: Weights visualization of query Q_3

An interesting question is how the model resolves the types of OOV entity words (i.e., cities *vs.* countries) in ambiguous queries, e.g., Q_3: *"How many people watched the Games in Macau?"*, since the random embeddings of entity words (e.g, *Macau*) cannot link them to their corresponding fields. The model executes Q_3 using the last three executors, with the last executor attending to the `#_audience` field as expected. Interestingly, however, the model attends to the `host_city` field in Executor-3, and then `host_country` in Executor-4 (see Figure 6), indicating the model learns to scan all possible fields to figure out the correct field of an OOV entity.

4.6 Querying Expanded Knowledge Source

We simulate a test case to evaluate the model's ability to generalize to an expanded knowledge source. We train a model on tables whose field sets are either $\mathcal{F}_1, \mathcal{F}_2, \ldots, \mathcal{F}_5$, where $\mathcal{F}_i$ is a subset of the entire field set $\mathcal{F}_\mathcal{T}$ and $|\mathcal{F}_i| = 5$. We then test the model on tables with all fields $\mathcal{F}_\mathcal{T}$ and queries whose fields span multiple

Query Type	SELECT_WHERE	SUPERLATIVE	WHERE_SUPERLATIVE	Overall
Accuracy	68.2%	84.8%	80.2%	77.7%

Table 3: Accuracies for querying expanded knowledge source

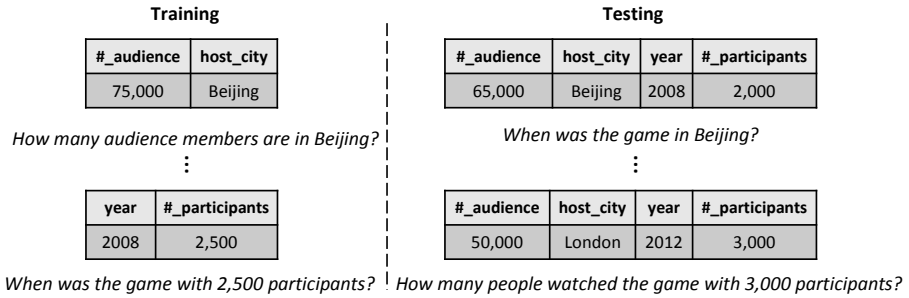


Figure 7: Expanded knowledge source querying simulation

subsets $\mathcal{F}_i$. Figure 7 illustrates the setting. All test queries exhibit field combinations unseen in training. This simulates the difficulty the model often encounters when scaling to large knowledge sources, which usually poses a great challenge on model's generalization ability. We evaluate the N2N model on a dataset of the first three types of relatively simple queries. The sizes of training/testing splits are 75,000 and 30,000, with equal numbers for each query type. Table 3 lists the results. The model maintains a reasonable performance even when the compositionality of test queries is previously unseen, showing the model's generalization ability in tackling unseen query patterns through the composition of familiar ones, and hence the potential to scale to larger and unseen knowledge sources.

5 Related Work

This work is related to semantic parsing, which aims to parse NL queries into logical forms executable on KBs [19, 1]. Recent studies take a semi-supervised learning approach, and adopt the results of query execution as indirect supervision to train a parser [3, 4, 16, 13, 11]. Semantic parsers learned in this way can scale to large open domain KBs, but are inadequate for understanding complex queries because of the intractable search space incurred by the flexibility of parsing algorithms. Our work follows this approach in using query answers as indirect supervision, but jointly performs semantic parsing and query execution in distributional spaces, where the distributed representations of logical forms are implicitly learned in end-to-end QA tasks.

Our work is also related to the recent research of modeling symbolic computation using neural networks, pioneered by the development of Neural Turing Machines (NTMs) [8] and the work of learning to execute (LTE) simple Python programs using LSTM [17]. It is related to both lines of research in using external memories like NTMs and learning by executing like LTE. As a highlight and difference, our work employs multiple layers of deep memories, with the neural network operations highly customized towards querying KB tables.

Perhaps the most related work is the recently proposed NEURAL PROGRAMMER [12], which studies the same task of executing queries on tables using DNNs. While in NEURAL PROGRAMMER, the query planning is modeled using DNNs to determine which operation to execute at each step, the symbolic operations are predefined by users. In contrast our model is fully neuralized: it models both the query planning and query execution using DNNs, which are jointly optimized via end-to-end training. Our model learns symbolic operations using a data-driven approach. We also present results on NL queries and demonstrate that a fully neural system is capable of executing compositional logic operations up to a certain level of complexity.

6 Conclusion

We propose NEURAL ENQUIRER, a fully neural, end-to-end differentiable network that learns to execute compositional natural language queries on knowledge base tables. We present results on a set of synthetic QA tasks to demonstrate the ability of NEURAL ENQUIRER to answer fairly complicated compositional queries across multiple tables. In the future we plan to advance this work in the following directions. First we will apply NEURAL ENQUIRER to natural language questions and natural language answers, where both the input query and the output supervision are noisier and less informative. Second, we are going to scale to real world QA task as in [13], for which we have to deal with a large vocabulary and novel predicates. Third, we are going to work on the computational efficiency issue in query execution by heavily borrowing the symbolic operation.

References

[1] Y. Artzi, K. Lee, and L. Zettlemoyer. Broad-coverage CCG semantic parsing with AMR. In *EMNLP*, 1699–1710, 2015.

[2] D. Bahdanau, K. Cho, and Y. Bengio. Neural machine translation by jointly learning to align and translate. In *ICLR*, 2015.

[3] J. Berant, A. Chou, R. Frostig, and P. Liang. Semantic parsing on freebase from question-answer pairs. In *EMNLP*, 1533–1544, 2013.

[4] J. Berant and P. Liang. Semantic parsing via paraphrasing. In *ACL (1)*, 1415–1425, 2014.

[5] A. Bordes, N. Usunier, A. Garca-Durn, J. Weston, and O. Yakhnenko. Translating embeddings for modeling multi-relational data. In *NIPS*, 2787–2795, 2013.

[6] D. L. Chen and R. J. Mooney. Learning to sportscast: A test of grounded language acquisition. In *ICML*, 128–135, 2008.

[7] J. Clarke, D. Goldwasser, M.-W. Chang, and D. Roth. Driving semantic parsing from the world's response. In *CoNLL*, 18–27, 2010.

[8] A. Graves, G. Wayne, and I. Danihelka. Neural turing machines. *CoRR*, abs/1410.5401, 2014.

[9] J. Kim and R. J. Mooney. Unsupervised pcfg induction for grounded language learning with highly ambiguous supervision. In *EMNLP-CoNLL*, 433–444, 2012.

[10] P. Liang, M. I. Jordan, and D. Klein. Learning dependency-based compositional semantics. In *ACL (1)*, 590–599, 2011.

[11] D. K. Misra, K. Tao, P. Liang, and A. Saxena. Environment-driven lexicon induction for high-level instructions. In *ACL (1)*, 992–1002, 2015.

[12] A. Neelakantan, Q. V. Le, and I. Sutskever. Neural programmer: Inducing latent programs with gradient descent. *CoRR*, abs/1511.04834, 2015.

[13] P. Pasupat and P. Liang. Compositional semantic parsing on semi-structured tables. In *ACL (1)*, 1470–1480, 2015.

[14] T.-H. Wen, M. Gasic, N. Mrksic, P. hao Su, D. Vandyke, and S. J. Young. Semantically conditioned lstm-based natural language generation for spoken dialogue systems. In *EMNLP*, 1711–1721, 2015.

[15] J. Weston, A. Bordes, S. Chopra, and T. Mikolov. Towards ai-complete question answering: A set of prerequisite toy tasks. *CoRR*, abs/1502.05698, 2015.

[16] W. Yih, M. Chang, X. He, and J. Gao. Semantic parsing via staged query graph generation: Question answering with knowledge base. In *ACL (1)*, 1321–1331, 2015.

[17] W. Zaremba and I. Sutskever. Learning to execute. *CoRR*, abs/1410.4615, 2014.

[18] M. D. Zeiler. ADADELTA: an adaptive learning rate method. *CoRR*, abs/1212.5701, 2012.

[19] L. S. Zettlemoyer and M. Collins. Learning to map sentences to logical form: Structured classification with probabilistic categorial grammars. In *UAI*, 658–666, 2005.

Multi-Dimensional, Phrase-Based Summarization in Text Cubes

Fangbo Tao[†], Honglei Zhuang[†], Chi Wang Yu[‡], Qi Wang[†], Taylor Cassidy[§],
Lance Kaplan[§], Clare Voss[§], Jiawei Han[†]
[†] *Department of Computer Science, UIUC*
[‡]*Microsoft Research*
[§] *US Army Research Laboratory*

Abstract

To systematically analyze large numbers of textual documents, it is often desirable to manage documents (and their metadata) in a multi-dimensional text database (Text Cube). Such structure provides flexibility of understanding local information with different granularities. Moreover, the contextualized analysis derived from cube structure often yields comparative insights. To quickly digest the content of subsets of documents in the multi-dimensional context, we study the problem of phrase-based summarization *of a subset of documents of interest. We propose a new phrase ranking measure to leverage the relation between document subsets induced by multi-dimensional context and identify phrases that truly distinguish the queried subset of documents from neighboring subsets (i.e.,* background*). Our quality evaluation suggests the new measure involving dynamic, query-dependent background generation is more effective than previous measures using the whole corpus as a static background for finding representative phrases. Computing this measure is more expensive due to the need of access to many subsets of documents to answer one query. We develop a cube-based analytical platform that implements an efficient solution by materializing a deliberately selected part of statistics, and using these statistics to perform online query processing within a constant latency constraint. Our experiments in a large news dataset demonstrate the efficiency in both query processing time and storage cost.*

1 Introduction

With ever more massive datasets accumulating in text repositories (*e.g.*, news articles, business reports, customer reviews, *etc.*), it is highly desirable to conduct multi-dimensional analysis on text data, where the dimensions correspond to multiple meta attributes (*e.g.*, category, date/time, location, author, *etc.*) associated with the documents. The dimensions provide rich context to partition the documents and relate them, and users can use these dimensions to navigate to a subset of documents of interest from a huge corpus. Typically, structured/relational data has been handled by relational database systems, and such systems also provide some text indexing and search capabilities to assist text data stored in such (extended) relational database systems. However, such kind of systems often suffer from the following limitations.

- It can hardly support systematic analysis of large collections of free text in multi-dimensional way, although such text data is ubiquitous in real-world;

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

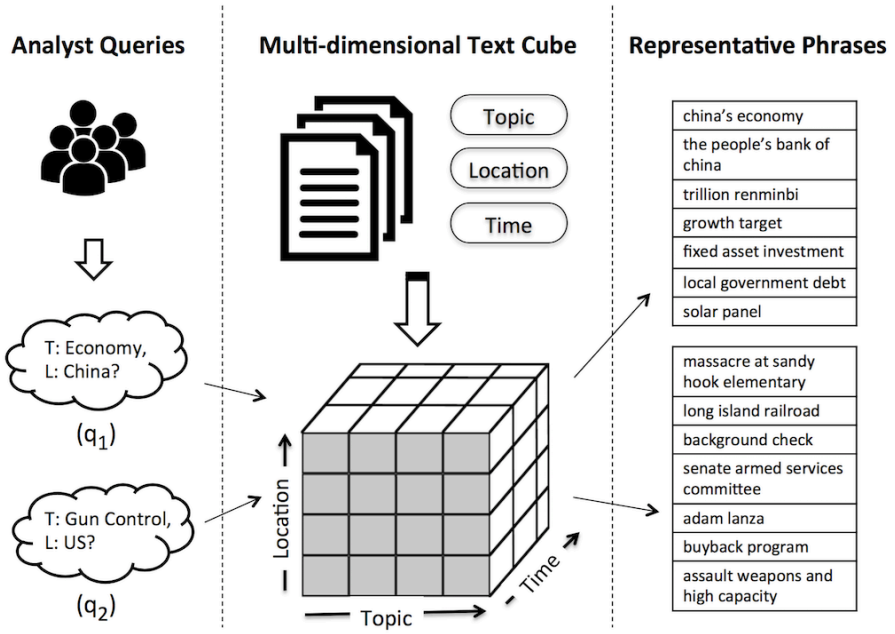


Figure 1: Illustration of phrase-based summarization in text cubes

- It usually does not support data cube technologies on text data and multidimensional text mining, although it is obvious that text mining and data cube technologies can mutually enhance each other; and
- There is a lack of a general platform that can support integrated multi-dimensional analysis of structured and text data, on top of which many powerful analysis methods and tools can be developed, experimented and refined, such as viewing such data sets as interconnected information networks and further applying information network analysis technology.

In this paper, we propose a multi-dimensional perspective of large-scale text corpora. In particular, we introduce the framework of *Text Cube* [8] and its analytical platform [14]. To help users efficiently explore the text cubes, we study the problem of *phrase-based summarization in multi-dimensional context: given user-specified dimensions and their values, return top-k phrases that characterize the corresponding set of documents*. The resulting phrases carry rich semantics and may benefit various downstream applications, *e.g.*, *text summarization*.

Example 1. Suppose a multi-dimensional text database is constructed from *New York Times* news repository with three meta attributes: *Location*, *Topic*, and *Time*, as shown in Figure 1. An analyst may pose multi-dimensional queries such as: (q_1): ⟨China, Economy⟩ and (q_2): ⟨US, Gun Control⟩. Each query asks for summary of a cell defined by two dimensions *Location* and *Topic*. What kind of cell summary does she like to see? Frequent unigrams such as *debt* or *senate* are not as informative as multi-word phrases, such as *local government debt* and *senate armed service committee*. The phrases preserve better semantics as integral units rather than as separate words.

Generally, three criteria should be considered when ranking representative phrases in a selected multi-dimensional cell: (i) integrity: a phrase that provides integral semantic unit should be preferable over non-integral unigrams, (ii) popularity: popular in the selected cell (i.e., selected subset of documents), and (iii) distinctiveness: distinguish the selected cell from other cells.

The remainder of the paper proceeds as follows. Section 2 introduces the framework of *Text Cube* and explains the power of converting text corpora to text cubes. The phrase-based summarization is proposed within the framework. Its effectiveness is evaluated by various experiments. Section 3 introduces the computational

platform for multi-dimensional text analysis, including the computational optimization for phrase-based summarization. Section 4 concludes the paper.

2 Multi-dimensional Text Analysis

In this section, we formally define the concept of *Text Cube*, the *Phrase-based Summarization* problem, the three phrase ranking criteria, and multiple experimental results to elaborate the effectiveness of phrase summarization.

Several pieces of related work have been proposed along this research line. Text Cube [8] takes a multi-dimensional view of textual collections and proposed OLAP-style *tf* and *idf* measures. Besides that, [7, 11] also proposed OLAP-style measures on term level using only local frequency, which cannot serve as effective semantic representations. [16, 4] focused on interactive exploration framework in text cubes given keyword queries, without considering the semantics in raw text. Similarly, R-Cube [10] is proposed where user specify an analysis portion by supplying some keywords and a set of cells are extracted based on relevance. Another related topic is Faceted Search [6, 15, 2, 3], which dynamically aggregates information for an ad-hoc set of documents. the aggregation is usually conducted on meta data (called *facets*), not document content.

2.1 Text Cube

Similar to traditional multi-dimensional data cubes, a *text cube* [8] is a data model but over text collection $\mathcal{DOC}$ that has metadata for documents. The metadata can be either extrinsic attributes of the documents, such as classification taxonomy, or intrinsic information extracted from the documents, such as named entities mentioned in them. In this paper, we focus on single-valued categorical metadata, and leave other types of metadata to future work. We assume there are n categorical attributes (*i.e.*, *dimensions*) associated with each document in $\mathcal{DOC}$. For example, a news article in *NYT* corpus is represented as (*Jan 2012*, *China*, *Economy*, *'After a sharp economic slowdown through much of last year...'*). It denotes that the *'Time'* of the article is *Jan 2012*, *'Location'* is *China* and *'Topic'* is *Economy*.

The dimensions provide valuable context for each document. Like a traditional data cube, all distinct values of one dimension are organized in a *dimension hierarchy*. For i-th dimension, the dimension hierarchy $\mathcal{A}_i$ is a tree where the root is denoted as '$*$'. Each non-root node is a value in that dimension. The parent node of a dimension value a_i is denoted as $par(a_i)$, and the set of direct descendants of a_i is denoted as $des(a_i)$. For example, Figure 3 illustrates a partial dimension hierarchy about 'topics' in *NYT* corpus. It is a tree of height 4, with a root node '$*$'. $par($*Gun Control*$)$ = *Domestic Issues* and $des($'$*$'$) = \{$*Economy*, *Sports*, *Politics*$\}$.

Formally, we have the following definition.

Definition 1 (Multi-dimensional Text Cube): A text cube is defined as $\mathcal{TC} = (\mathcal{A}_1, \mathcal{A}_2, \ldots, \mathcal{A}_n, \mathcal{DOC})$, where $\mathcal{A}_i$ is a dimension hierarchy. Each document is in the form of $(a_1, a_2, \ldots, a_n, d)$, where $a_i \in \mathcal{A}_i \backslash \{*\}$ is a dimension value for $\mathcal{A}_i$ and d is a string of the content. A cell c in the cube is represented as $(a_1, \ldots, a_n, \mathcal{D}_c)$, where $a_i \in \mathcal{A}_i$, and $\mathcal{D}_c \subseteq \mathcal{DOC}$ is the subset of documents contained in cell c. For notation simplicity, we use $\langle a_{t_1}, \ldots, a_{t_k} \rangle$ to refer to a cell with non-$*$ dimension values $\{a_{t_1}, \ldots, a_{t_k}\}$.

Example 3. Figure 2 illustrates a mini example of news article text cube, with 3 dimensions (Time, Location and Topic) and 9 documents d_1–d_9. The Time dimension is derived from extrinsic attribute but Location and Topic are extracted by information extraction as in [13]. We pick 7 non-empty cells, where the top four are leaf cells without '$*$' dimensions, *e.g.*, (Jan 2012, China, Economy, $\{d_1, d_2\}$). The root cell (entire corpus) is represented as ($*$, $*$, $*$, $\{d_1$–$d_9\}$).

Text cube provides a framework for organizing text documents using meta-information. In particular, the cell space defined above embeds the inter-connection between different subsets of text. To capture those semantically close cells, we define *context* of a cell c as a composition of three parts.

Dimensions			Text Data
Year	**Location**	**Topic**	$\mathcal{DOC}$
2011	China	Economy	$\{d_1, d_2\}$
2012	China	Economy	$\{d_3, d_4, d_5\}$
2012	US	Gun Control	$\{d_6, d_7\}$
2013	US	Economy	$\{d_8, d_9\}$
$*$	China	Economy	$\{d_1, \ldots, d_5\}$
2012	$*$	$*$	$\{d_3, \ldots, d_7\}$
$*$	$*$	$*$	$\{d_1, \ldots, d_9\}$

Figure 2: Mini Example of *NYT* Corpus

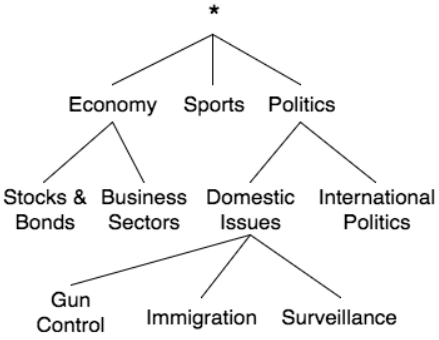


Figure 3: Hierarchy of *Topic*

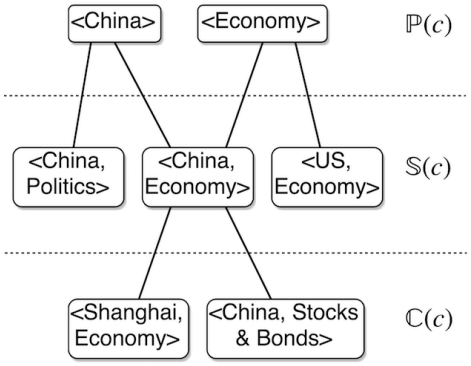


Figure 4: *Context* of cell $\langle$China, Economy$\rangle$

Definition 2 (Cell Context): The context of cell $c = \langle a_{t_1}, \ldots, a_{t_k} \rangle$ is defined as $\mathbb{P}(c) \bigcup \mathbb{S}(c) \bigcup \mathbb{C}(c)$, where:

- Parent set is defined as $\mathbb{P}(c) = \{\langle a_{t_1}, \ldots, par(a_i), \ldots, a_{t_k} \rangle | i \in t_1, \ldots, t_k\}$. Each parent cell is found by changing exactly one non-$*$ dimension value in cell c into its parent value;
- Children set is defined as $\mathbb{C}(c) = \{c' | c \in \mathbb{P}(c')\}$. Each child cell is found by either changing one $*$ value into non-$*$ or by replacing it by one of the child values; and
- Sibling set is defined as $\mathbb{S}(c) = \{c' | \mathbb{P}(c) \bigcap \mathbb{P}(c') \neq \emptyset\}$. Each sibling cell must share one parent with cell c.

Example 4. Figure 4 illustrates the partial context of cell $c = \langle$China, Economy$\rangle$. The parent set $\mathbb{P}(c)$ contains $\langle$China$\rangle$ and $\langle$Economy$\rangle$, sibling set $\mathbb{S}(c)$ has $\langle$China, Politics$\rangle$ and $\langle$US, Economy$\rangle$ and children $\mathbb{C}(c)$ contains $\langle$Shanghai, Economy$\rangle$ and $\langle$China, Stocks & Bonds$\rangle$.

2.2 Cube Perspective of Text Corpora

Organizing a text corpus into a text cube provides various possibilities to substantial improve user experience in browsing, retrieving, and analyzing large scale textual data.

- **Enriched horizon.** Multi-dimensional structure grants analysts with an enriched mine of knowledge to be discovered. For example, without a multi-dimensional structure, an analyst can either perform statistics on the entire corpus, or simply perform statistics on a single documents. However, when the corpus is organized as

a text cube, the analyst is able to study the connection between various statistics of documents and different categories. For example, one may check whether there is a correlation between the frequency of different words and the *publishing time* in a news corpus, to understand how the usage of a word varies. Similarly, one can also examine the key phrases of a certain category of documents (e.g. news articles about "Brazil") to obtain a better picture of the subset of interest. The additional meta-information not only allows analysts to deepen their understanding on different facets of the document data sets, but also provides them with better insights of the dimensions per se.

- **Contextualized analysis.** Multi-dimensional structure also enables the analysts to conduct analysis with a certain context. A analyst may be interested on a specific subset of documents, for example, news articles about "China Economy". However, documents from other relevant subsets may also be useful in better understanding this concept, like "Japan Economy" or "US Economy". Generally, they help analysts in comparative studies, to better understand the features the document subset shares with other subsets, and the features unique to the subset. As a more specific example, suppose an analyst is interested in summarizing key phrases of "China Economy", it is helpful to remove phrases overlapping with "Japan Economy" or "US Economy", such as "banking" or "currency", as they do not distinguish the subset of interest from others.

2.3 Phrase-based Summarization

This paper deals with the problem of mining representative phrases to serve as summary, in particular within multi-dimensional text cubes. A phrase is a multi-word sequence served as an integral semantic unit. The representative phrases for a cell, are the phrases that characterize the semantics of the selected documents. There is no universally accepted standard of being *representative*. Here we operationalize a definition in terms of three criteria.

- **Integrity:** An integral phrase must satisfy two conditions: (i) the multiple words in a phrase collocate together much more frequently than expected from random chance, and (ii) the phrase is a complete semantic unit, rather than a subsequence of another equally-frequent phrase.
- **Popularity:** A phrase is popular if it has a large number of occurrences. Representative phrases for a cell, in particular, should appear with some frequency within the documents of that cell. Very low frequency phrases within a cell do not contribute substantially to its semantics and so are not considered representative.
- **Distinctiveness:** High-popularity phrases that appear in many different cells constitute background noise, *e.g.*, 'earlier this month' and 'focus on'. Representative phrases should distinguish the target cell from its context, therefore provide more salient information to help users filter the noise. Distinctiveness is particularly critical in text cube scenarios, since analysts often navigate through the whole collection to find subsets of interest. Non-distinctive phrases will appear in many cells and offer redundant information.

However, none of the previous work has followed all three criteria. MCX [12, 1] follows *distinctiveness* (in a rough sense that only compare to the entire corpus) and ignores *popularity* and *integrity*. SegPhrase [5] addresses *integrity* in global quality phrase mining, but the notion of *popularity* and *distinctiveness* with respect to a target cell is not applicable to that problem setting. This paper proposes a new measure to evaluate all three criteria.

Within the whole ranked phrase list, top-k representative phrases normally have higher value for users in text analysis. As a further matter, the top-k query also enjoys computational superiority, so that users can conduct fast analysis. For these reasons, we define the problem as follows.

Definition 3 (Multi-Dimensional, Phrase-Based Summarization in Text Cube): Given a multi-dimensional text cube $\mathcal{TC} = (\mathcal{A}_1, \mathcal{A}_2, \ldots, \mathcal{A}_n, \mathcal{DOC})$, it takes $c = (a_1, \ldots, a_n, \mathcal{D}_c)$ as a query, and outputs top-k representative phrases based on the integrity, popularity and distinctiveness criteria.

Table 1: Top-10 representative phrases for NYT queries

⟨**US, Gun Control**⟩	⟨**US, Immigration**⟩	⟨**US, Domestic Politics**⟩	⟨**US, Law and Crime**⟩	⟨**US, Military**⟩
gun laws	immigration debate	gun laws	district attorney	sexual assault in the military
the national rifle association	border security	insurance plans	shot and killed	military prosecutors
gun rights	guest worker program	background check	federal court	armed services committee
background check	immigration legislation	health coverage	life in prison	armed forces
gun owners	undocumented immigrants	tax increases	death row	defense secretary
assault weapons ban	overhaul of the nation's immigration laws	the national rifle association	grand jury	military personnel
mass shootings	legal status	assault weapons ban	department of justice	sexually assaulted
high capacity magazines	path to citizenship	immigration debate	child abuse	fort meade
gun legislation	immigration status	the federal exchange	plea deal	private manning
gun control advocates	immigration reform	medicaid program	second degree murder	pentagon officials

First, we acknowledge that these three criteria can all be subjective and relative, and it is difficult to find a clear binary judgment whether each phrase satisfies all the criteria. Therefore, we decide to use a score between 0 and 1 to characterize the degree of each phrase in satisfying these criteria. For phrase p in cell c, we use $int(p, c) \in [0, 1]$, $pop(p, c) \in [0, 1]$, and $disti(p, c) \in [0, 1]$ to denote the three criteria, and $r(p, c)$ to denote the overall ranking score that combines these criteria.

To combine the above criteria, we first notice that they reflect conjunctive conditions that should be satisfied, and one cannot replace the other. For example, popular word sequences may have quite low distinctiveness and sometimes ill-formed surface (*i.e.*, low integrity). Rare phrases that only occur once can be well distinctive. Since every criterion is indispensable, any low score (*i.e.*, near 0) in $int(p, c)$, $pop(p, c)$ or $disti(p, c)$ should result in a low rank for phrase p. Therefore, we design $r(p, c)$ as the geometric mean of those three scores.

The three criteria are equally positioned, though one can assign different weights according to user's requirement in different applications. If one of the factors is close to 0, the geometric mean will be close to 0 as well. Alternatively, one can use harmonic mean to have the same property, but the score will then be strongly dominated by the weakest factor, which may be unfavorable because the role of the other two factors will be neglected.

When we design the concrete measures for each criterion, we are aware that the input documents can be any textual word sequences with arbitrary lengths, such as articles, titles, queries, tags, memos, messages and records. A good design of the measures should generalize well to a variety of text data. Therefore, we tend to use more statistical features and fewer linguistic features.

Now we discuss design principles that are more specific to the three criteria.

- Popularity and distinctiveness of a phrase are dependent of the target cell, while integrity is not. Hence, $int(p, c)$ can be simplified as $int(p)$.
- Popularity and distinctiveness can be measured from frequency statistics of a phrase in each cell, while integrity cannot. To measure integrity, one needs to investigate each occurrence of the phrase and other phrases to determine whether that phrase is indeed an integral semantic unit. We leverage *SegPhrase* [9] to compute integrity.
- Popularity relies on statistics from documents only within the cell $\mathcal{D}_c$, while distinctiveness relies on documents both in and out of the cell. We define the documents involved for distinctiveness measure calculation as *contrastive document set*. More precise distinctiveness measure requires appropriate choice of contrastive document set. In our particular algorithm design, sibling set $\mathbb{S}(c)$ is used as contrastive document set.

With the phrase ranking algorithm designed based on the aforementioned principles, it is applied on NYT 2013-2016 dataset and PubMed Cardiac data for quality evaluation. Our algorithm is referred as **RepPhrase** and multiple baselines are referred as **MCX** [12], **SegPhrase** [9] and their combinations.

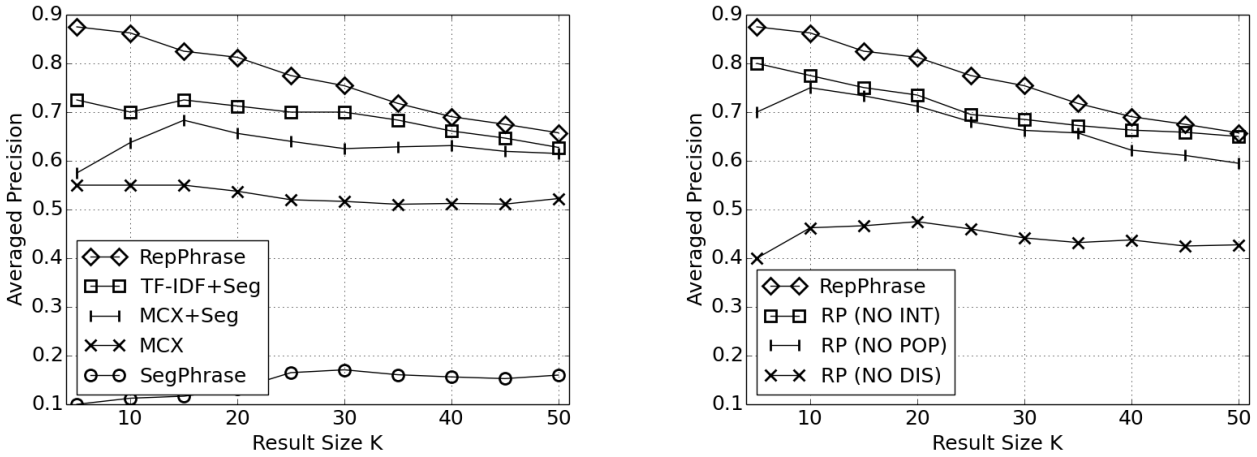
Case study on NYT. We show 5 real queries in NYT dataset and their representative phrase list in Table 1. Query ⟨US, Gun Control⟩ and ⟨US, Immigration⟩ are siblings, ⟨US, Domestic Politics⟩ is their parent cell. ⟨US,

Table 2: Top representative phrases for 5 cardiac diseases

⟨**Cerebrovascular Accident**⟩	⟨**Ischemic Heart Disease**⟩	⟨**Cardiomyopathy**⟩	⟨**Arrhythmia**⟩	⟨**Valve Dysfunction**⟩
alpha-galactosidase a	Cholesteryl ester transfer protein	Interferon gamma	Methionine synthase	Mineralocorticoid receptor
brain neurotrophic factor	apolipoprotein a-I	interleukin-4	ryanodine receptor 2	tropomyosin alpha-1 chain
tissue-type activator	integrin alpha-iib	interleukin-17a	potassium v.g. h member 2	elastin
apolipoprotein e	adiponectin	titin	inward rectifier channel 2	beta-2-glycoprotein 1
neurogenic l.n.h.p. 3	p2y purinoceptor 12	tumor necrosis factor	beta-2-glycoprotein 1	myosin-binding protein c

Domestic Issues⟩, ⟨US, Law and Crime⟩ and ⟨US, Military⟩ are also siblings. For the first two queries, the discovered phrases are specific to gun control and immigration. There are both entity names like *the national rifle association* and *guest worker program* and event-like phrases like *assault weapons ban* and *overhaul of the nation's immigration laws*. In their parent cell ⟨US, Domestic Politics⟩, the top phrases cover various children cell topics, including gun control, immigration, insurance act and federal budget. This list provides very informative phrases that describe the major content. For the two siblings of ⟨US, Domestic Politics⟩ (last two columns), the lists also cover the main entities involved and the major topics, *e.g.*, *second order murder*, *sexual assault in the military*, *etc.*. Also notice that, these top lists of representative phrases keep good balance between short phrases and long phrases. That is mainly credited the consideration of both popularity and distinctiveness without introducing bias to phrase length.

Case study on PubMed Cardiac data. In collaboration with UCLA BD2K team, we apply the phrase-based summarization on PubMed cardiac publications. They provide 5 categories of cardiac diseases and a set of 300+ protein candidates. The goal of our summarization is to discover top contributing proteins for these disease categories. The results are shown in Table 2. These top proteins help medical scientists find more concrete direction to look into and largely reduce the time spent on reading irrelevant publications. Since the *distinctiveness* is a major criterion in our ranking, we note that non-informative proteins that are related to all diseases, like *amyloid beta a4 protein*, are not included in the top list. Another exciting discovery is that protein *titin*, a newly discovered protein, is also listed as top protein for *Cardiomyopathy*.



(a) comparison to baselines (b) comparison to ablations

Figure 5: Phrase assignment accuracy

Phrase-to-cell assignment accuracy. The idea of this experiment is to quantify how many phrases among top-k results of a cell indeed represent the semantics of that cell. We test eight queries. Four of them are *1-Dim Queries*, and the other four are *2-Dim Queries*. To generate non-trivial test queries, we first randomly pick two *1-Dim Queries* and two *2-Dim Queries*; then for each picked query, we add the most similar sibling in terms

of both size and content as a paired query. To ease the labeling, for each pair of test queries, we first collect all top-50 phrases generated by all the measures for both queries. For each phrase in the pool, we label it with either one of the two cells which it best represents, or a 'None' label in three circumstances: 1) it is not a valid phrase, 2) it is not relevant to either cell and 3) it is a background phrase that are shared by both cells. We then measure the accuracy of phrase assignment by the average precision from top-5 to top-50 phrases. We show the result in Figure 5(a) for baselines and Figure 5(b) for ablations.

In general, as k grows, the precisions of those measures go down. In Figure 5(a), **RepPhrase** has the best precision and **SegPhrase** has the worst. Also, the difference of precision between **RepPhrase** and others decreases as k grows. That is attributed to the limited number of true representative phrases. **RepPhrase** successfully ranks these good phrases high, others gradually include them as k grows. Amongst all the baselines, **TF-IDF+Seg** outperforms others since it is the only baseline that captures all three criteria. However, it still loses to **RepPhrase**. Both use sibling cells as contrastive group, using classification probability (**RepPhrase**) as *distinctiveness* performs better than using IDF (**TF-IDF+Seg**).

In Figure 5(b), we show the performance drop by removing one of the three criteria respectively. We notice that **RP (NO INT)** has the best precision amongst all ablations and **RP (NO DIS)** has the worst, which indicates the relative importance of the criteria: *distinctiveness* $>$ *popularity* $>$ *integrity*. One interesting comparison is between **MCX+Seg** and **RP (NO POP)**. These two can be viewed as two versions of standalone *distinctiveness* measure with different contrastive document groups. Using dynamic sibling cells as contrastive group (**RP (NO POP)**) performs better than using the static entire collection (**MCX+Seg**), especially on the top phrases. It further justifies the choice of using dynamic background over static background.

3 Platform for multi-dimensional text analysis

As discussed above, converting text corpora into multi-dimensional text cubes provides various benefits, including i) flexibility of user queries that captures insights with different granularities and ii) contextualized analysis that is able to discover comparative insights. To support such general cube-based analytical tasks, we proposed and implemented the generalized infrastructure. Like the phrase-based summarization task, other multi-dimensional analytical tasks share similar computational and operational characteristics.

Computational Characteristic: given a pre-defined dimension structure, similar to traditional *OLAP*-operations, proper pre-computation (called *materialization*) helps to speedup online user queries and therefore supports real-time query responses. Number of possible multi-dimensional queries are often exponential, thus it is necessary to intelligently select partial cells to materialize. Since these text analytical measures are more complicated than traditional *distributive* and *algebraic* measures, we normally need to materialize intermediate result and require reasonable amount of online computation after query is issued. Such hybrid computational scheme is normally shared by various analytical tasks.

Operational Characteristic: different analytical tasks share the same input/output format. Normally, the system takes a multi-dimensional user query as input, i.e., $\langle$China, Economy$\rangle$, and returns structured textual result, i.e., top-k phrases in phrase-based summarization and k-topics in cube-based topic modeling. Therefore, many operations including *indexing*, *retrieval* and etc. can also be shared by different tasks.

Therefore, we created a platform for general multi-dimensional text analysis. It provides a generalized platform that can easily import any collection of free text and structured data, such as news data, aviation reports or academic papers, extract entities, construct the text-rich data cube and support powerful search and mining functions. For structured data, multidimensional data cube can be constructed easily. For text-intensive data with minimally predefined structured information (e.g., news data), natural language and information extraction tools can be used to extract entities of multiple types such as person, location, organization, time, and event. This platform provides a tremendous opportunity to conduct multi-dimensional analysis on text and structured data in powerful and flexible ways.

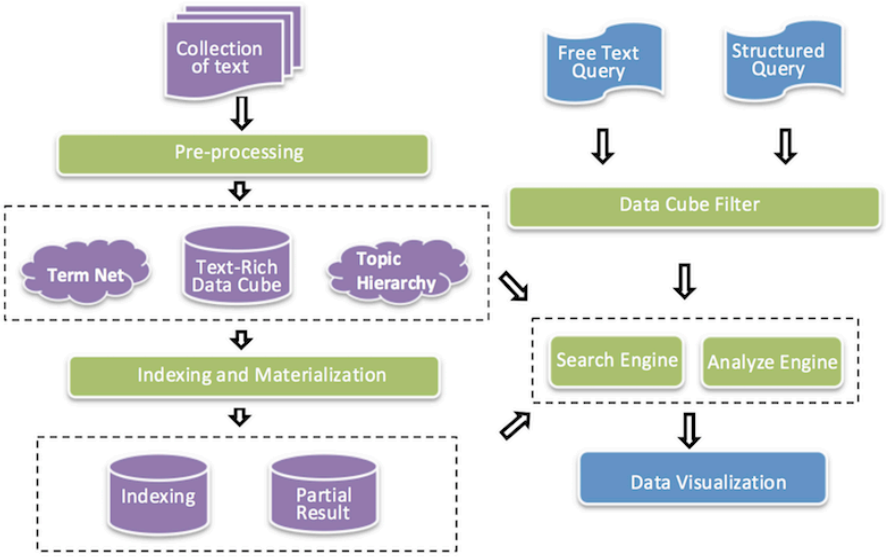


Figure 6: System Architecture of the Platform

System Architecture. The platform is designed as shown in Figure 6. It consists of the following modules: (1) *Data Uploading and Preparation*, which pre-processes the free text corpus from user's uploading and converts it into a text-rich data cube with term network and topic hierarchy extracted; (2) *Indexing and Materialization*, which builds indexing and partial materialization results for keyword search, top cell finding, single dimension distribution and hierarchical topic modeling; (3) *Query-Based Search and Mining Module*, which processes user-queries (both search and analysis queries) by parsing the query, selecting and executing appropriate search or mining module (which searches or mines on the constructed text-rich data cube to derive results); and (4) result presentation by *Visualization and Interpretation* of the search/mining processes and results.

The platform reveals another advantage of converting text corpora into multi-dimensional text cubes, that is the power of real-time text analysis. The rich structure embedded in text cubes empowers smart indexing and materialization that enables real-time processing of any multi-dimensional query. Without such multi-dimensional structure, it is challenging to support real-time text analysis on arbitrary portion of large text corpora.

3.1 Real-time Phrase-based Summary Generation

In this section, we use phrase-based summarization as example task to introduce how the offline/online computation scheme is implemented.

Utility-Guided Materialization. In offline computation, we extend the *GreedySelect* algorithm [8] to our task and develop the utility-guided partial materialization. The algorithm first conducts a *topological sorting* by the *parent-descendant* relationship in the cube space. Then it traverses the cells in the bottom-up order. This order ensures that all cells used for aggregating the current cell must have been examined, so the dynamic programming of cost estimation can proceed. For each cell, we do not simply materializing all the required cells (all siblings). Instead, it repeatedly attempts materialization of one sibling, and reevaluates the cost of querying the target cell, until it falls below threshold. The order of choosing siblings affects how many siblings will be materialized and how much storage cost is needed to meet the constraint. We use a *utility* function for each sibling cell c' to guide this process.

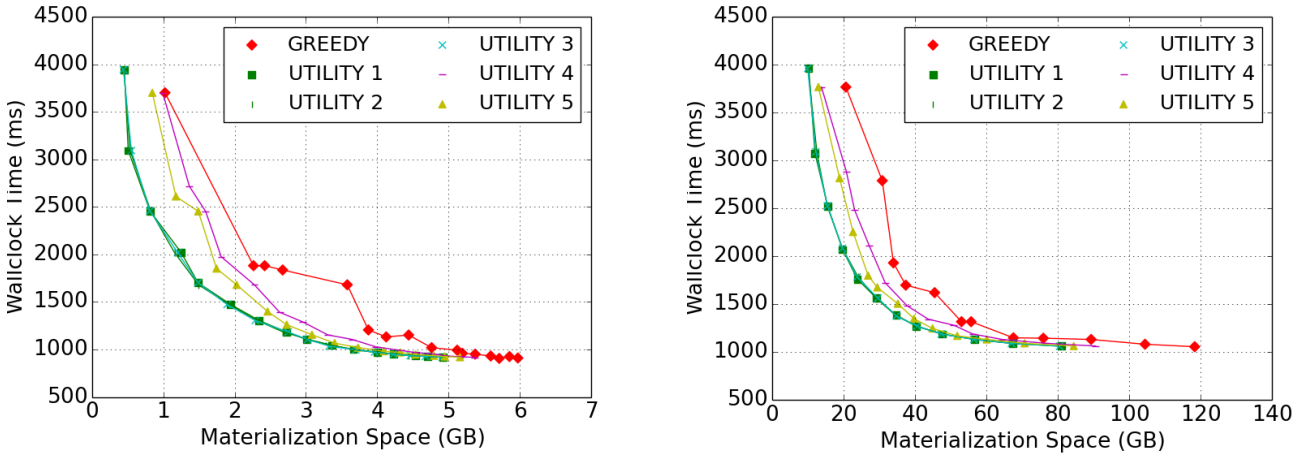
Optimized Online Processing. The vanilla online processing needs to compute the ranking measure for all phrase candidates in a cell in order to sort them. The computation of the distinctiveness score can be expensive,

if the cell is not materialized. We propose an early termination and skipping technique to prune phrase candidates that are impossible to be among top-k.

We evaluate the computational performance using the full NYT dataset **4-Dim Cube** and **6-Dim Cube** (both have 4.7 million articles, 17.04 GB raw size, but different dimension numbers). For the offline computation, we compare the following algorithms for materializing phrase-level statistics: 1) **FULL** (full materialization), 2) **LEAF** (leaf materialization), 3) **GREEDY** (in [8]) and **UTILITY 1-5** (with 5 different utility functions).

Table 3: Space-time trade-off of **LEAF** and **FULL**

	4-Dim Cube		**6-Dim Cube**	
	Space (GB)	Time (s)	Space (GB)	Time (s)
LEAF	0.68	73.2	26.76	3407.5
FULL	20.17	0.86	706.0	0.89



(a) 4-Dim Cube

(b) 6-Dim Cube

Figure 7: Time-space balance

Figure 7(a) and 7(b) shows the space-time trade-off on **4-Dim Cube** and **6-Dim Cube**. Since **LEAF** and **FULL** strategies have quite exceptional worst query time or materialization space, the result is separately shown in Table 3. We first notice that the space cost of **LEAF** is as low as 26.76 GB in **6-Dim Cube**, but the worst query time is more than 3,400 seconds. If we materialize every cell as in **FULL**, it has the minimized worst query time but consumes about 706 GB to materialize. The other 6 strategies make trade-offs between time and space by setting different latency constraint. We notice that all five utility-guided strategies outperform **GREEDY**, *i.e.*, their curves are closer to the origin point. In particular, picking any of **UTILITY 1-3** yields the best trade-off that can take less than 10% of the storage compared to **FULL** and less than 50% of the **GREEDY** strategy with same worst query time.

4 Conclusion

This paper proposes multi-dimensional text analysis in text cubes and an interesting application: multi-dimensional phrase-based summarization. It mines top-k representative phrases based on three criteria: integrity, popularity and distinctiveness. We propose a fine-grained distinctiveness assessment that considers phrase distributions across sibling cells. This is shown to be more effective than previous measures. Given computational

challenges imposed by these textual measures, we develop a generalized platform to support efficient online and offline computational optimization. These can be generally applied to any measure in text cubes.

There are several possible extensions of the current problem to explore in future work. (1) Instead of outputting top-k phrases, one can design measures for generating top-k semantic clusters, which improve coverage of the content and reduce semantic redundancy. (2) Users may make a sequence of OLAP queries before navigating to the target cell. One can study the patterns of such query sequence and develop semantic representations accordingly. (3) One can further investigate the context-aware materialization problem. It will be useful to develop algorithms with stronger theoretical guarantee in optimizing the time-space trade-off.

Beside phrase-based summarization, other useful text analytical problems can be studied within data cube scenarios, including outlier detection, sentence-based summarization, and sentiment analysis. We believe the multi-dimensional framework can help us achieve real-time, flexible and contextualized analysis for such tasks.

References

[1] S. Bedathur, K. Berberich, J. Dittrich, N. Mamoulis, and G. Weikum. Interesting-phrase mining for ad-hoc text analytics. *PVLDB*, 2010.

[2] O. Ben-Yitzhak, N. Golbandi, N. Har'El, R. Lempel, A. Neumann, S. Ofek-Koifman, D. Sheinwald, E. Shekita, B. Sznajder, and S. Yogev. Beyond basic faceted search. In *WSDM*, 2008.

[3] D. Dash, J. Rao, N. Megiddo, A. Ailamaki, and G. Lohman. Dynamic faceted search for discovery-driven analysis. In *CIKM*, 2008.

[4] B. Ding, B. Zhao, C. X. Lin, J. Han, and C. X. Zhai. Topcells: Keyword-based search of top-k aggregated documents in text cube. In *ICDE*, 2010.

[5] A. El-Kishky, Y. Song, C. Wang, C. R. Voss, and J. Han. Scalable topical phrase mining from text corpora. *PVLDB*, (3), 2014.

[6] M. A. Hearst. Clustering versus faceted categories for information exploration. *CACM*, (4), 2006.

[7] A. Inokuchi and K. Takeda. A method for online analytical processing of text data. In *CIKM*, 2007.

[8] C. X. Lin and e. a. Ding, Bolin. Text cube: Computing ir measures for multidimensional text database analysis. In *ICDM*, 2008.

[9] J. Liu, J. Shang, C. Wang, X. Ren, and J. Han. Mining quality phrases from massive text corpora. In *SIGMOD*, 2015.

[10] J. M. Pérez-Martínez, R. Berlanga-Llavori, M. J. Aramburu-Cabo, and T. B. Pedersen. Contextualizing data warehouses with documents. *Decision Support Systems*, 45(1):77–94, 2008.

[11] F. Ravat, O. Teste, R. Tournier, and G. Zurfluh. Top_keyword: an aggregation function for textual document olap. In *Data Warehousing and Knowledge Discovery*. Springer, 2008.

[12] A. Simitsis, A. Baid, Y. Sismanis, and B. Reinwald. Multidimensional content exploration. *PVLDB*, (1), 2008.

[13] F. Tao, G. Brova, J. Han, H. Ji, C. Wang, B. Norick, A. El-Kishky, J. Liu, X. Ren, and Y. Sun. Newsnetexplorer: automatic construction and exploration of news information networks. In *SIGMOD*, 2014.

[14] F. Tao, J. Han, et al. Eventcube: multi-dimensional search and mining of structured and text data. In *KDD*, 2013.

[15] D. Tunkelang. Faceted search. *Synthesis lectures on information concepts, retrieval, and services*, (1), 2009.

[16] B. Zhao, X. Lin, et al. Texplorer: keyword-based object search and exploration in multidimensional text databases. In *CIKM*, 2011.

Answering End-User Questions, Queries and Searches on Wikipedia and its History

Maurizio Atzori
University of Cagliari
atzori@unica.it

Shi Gao
UCLA
gaoshi@cs.ucla.edu

Giuseppe M. Mazzeo
UCLA
mazzeo@cs.ucla.edu

Carlo Zaniolo
UCLA
zaniolo@cs.ucla.edu

Abstract

Knowledge bases (KBs) encoded using RDF triples deliver many benefits to applications and programmers that access the KBs on the web via SPARQL endpoints. In this paper, we describe and compare two user-friendly systems that seek to make the universal knowledge of Web KBs available to users who neither know SPARQL, nor the internals of the KBs. We first describe CANaLI, that lets people enter Natural Language (NL) questions and translates them into SPARQL queries executed on DBpedia. CANaLI removes the ambiguities that are often present in NL communication by requiring the use of a Controlled NL and providing on-line knowledge-driven question-completion that shows alternate correct interpretations. While CANaLI is a very powerful NL system, which placed first in the 2016 competition on Question Answering over Linked Data QALD-6, even more powerful user-friendly interfaces are available to users who enter questions and queries on web-browsers. In particular, the SWiPE system provides a wysiwyg interface that lets users specify powerful queries on the Infoboxes of Wikipedia pages in a query-by-example fashion. Thus, in addition to those supported in CANaLI, we now have queries with (i) complex aggregates, (ii) structured conditions combined with keyword-based searches, and (iii) temporal conditions on Cliopedia, a historical knowledge base that captures the evolution of Wikipedia entities and properties. These systems demonstrate that semi-curated web document corpora and their KBs are making possible the seamless integration through user-friendly interfaces of (i) NL question answering, (ii) structured DB queries, and (iii) information retrieval. These were once viewed as distinct functions supported by different enabling technologies.

1 Introduction

Knowledge bases (KBs) are playing a crucial role in many applications, such as text summarization and classification, opinion mining, semantic search, and question answering systems. In recent years, several projects have been devoted to creating such KBs of general nature or specialized focus: [23, 34, 35, 36, 37, 39, 41, 42]. Of particular interest are KBs that are closely connected with curated or semicurated document corpora, such as Wikipedia. In fact, most of Wikipedia pages use an Infobox to summarize key properties and values of the entities (subjects) described in the pages. Now, DBpedia [43] has harvested and organized the information contained in those Infoboxes into a KB using RDF [22] and provides a SPARQL endpoint for accessing the information. The success of the encyclopedic Wikipedia and its associates KBs have motivated the start of thousands

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

of related projects that cover more specialized domains [38], and provide RDF KBs accessible via SPARQL or other structured query languages.

However, the great majority of web users are neither familiar with SPARQL nor with the internals of the KBs and are thus denied access to these KBs and the many benefits they offer. Therefore, the design of user-friendly interfaces to deliver the riches of web KBs to all users has emerged as a challenging research priority of great technical interest and practical significance. The importance of the topic has inspired a large body of previous research and the launching of an annual competitions on Question Answering over Linked Data (QALD). In this paper, we describe the Natural Language (NL) QA system CANaLI [25], that placed first in the 2016 competition and also supports a KB-driven query completion function that assists users in formulating their questions.

While NL QA interfaces often provide the most comfortable communication medium for casual users, other friendly interfaces are available through web browsers. In particular, the SWiPE system provides a wysiwyg interface [2] where *by-example structured queries* (BEStQ) are entered as follows: (i) the user selects an example page activating its Infobox, (ii) the user can now insert conditions into the relevant fields of the Infobox, and (iii) these conditions are translated into a SPARQL query that is executed on the DBpedia KB. We will discuss this system in some depth since it allows users to ask powerful queries and questions that are currently not available in NL QA systems, including (a) queries requiring complex conditions and aggregates, (b) questions combining structured queries with keyword searches, and (c) questions and queries on the history of the KB. We will underscore the significance of these new types of questions and queries which suggest important new goals for research and CANaLI's extensions.

2 The Design of CANaLI

The importance and the difficulty of NL QA is witnessed by the large amount of research efforts devoted to this problem [12, 13, 28]. Answering questions posed in NL requires to tackle several non-trivial sub-problems, such as deriving the syntactic structure of the question, associating the phrases of the question with the resources of the KB, and resolving the ambiguities that are quite common in these two tasks (inasmuch as different concepts can be represented using the same phrase and several syntactical relationships are possible between the constituents of the sentence). Ambiguity resolution is indeed a very hard problem, sometimes even for humans[1], and the domain knowledge, usually available to the speakers but not to the system, is fundamental in order to disambiguate the query intention.

We designed our NL system, CANaLI [25], with the objective of avoiding ambiguities. This goal was achieved by combining the use of a *controlled natural language* (CNL) interface with an interactive autocompleter that guides the user in typing questions that are consistent with the underlying KB.

The use of a CNL [20] reduces the possibility of ambiguities in interpreting the syntactical relationships between the constituents of the sentence, by limiting the syntactical forms that can be accepted. The key challenge of these systems is to be able to accept a language that is formal enough to be interpreted by machines with high accuracy, and, at the same time, natural enough to be readily acquired by people as an idiomatic version of their NL. The CNL used by CANaLI, besides being flexible enough to support the typical questions that users may want to pose over a KB, enables a very desirable question autocompletion function: although it is very popular in Web search engines, autocompletion is a novelty for CNL systems. Moreover, while autocompletion in search engines is based on the popularity of searches, the CANaLI autocompletion strategy is based on the underlying KB. This feature allows people to feel more comfortable while entering the questions, since they are (i) guided in following the grammar accepted by CANaLI whereby users will only need minimal knowledge of the CNL accepted by the system before they can start using it, and (ii) made aware that alternative interpretations are possible for a question. Thus, while the main benefit of (i) is improving interaction with the CNL interface, (ii) would be desirable in any NL system to forewarn the users of potential ambiguities in their questions.

[1] This problem can be exemplified by the sentence: "I saw the man on the hill with a telescope".

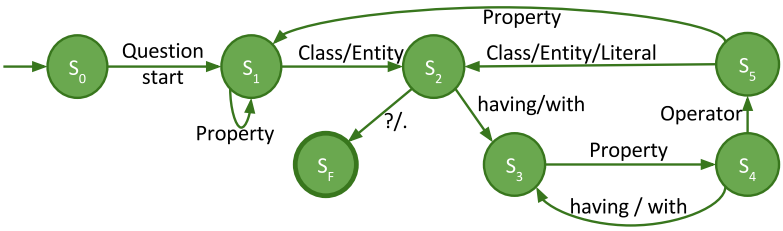


Figure 1: The main states and transitions of the finite-state automaton used by CANaLI

2.1 Answering Simple Questions

Figure 1 depicts the main states and transitions of the finite-state automaton used by CANaLI[2]. We provide an intuition of the operation of CANaLI by means of some simple examples. Let us consider the question: "Who is the spouse of Barack Obama?". The automaton is initially in the state S_0, ready to accept tokens representing the *question start*. The token "Who is the" represents a valid way to start a question; thus the automaton accepts it, and moves to the state S_1, where it is ready to accept a token representing either a *property*, or an *entity*, or a *class*. In our example, the user enters "spouse", that is a *property* recognized by CANaLI. Thus, the system loops back to S_1, ready to accept another *property*, *entity*, or *class*. In our example, the user enters "Barack Obama" that, being an *entity* with the *property* "spouse," is accepted by the system. In general, in order to be consistent with the underlying KB (*DBpedia*, in our example), the user must enter entities that have a spouse, otherwise the system will stop the user from progressing any further. However, to reach this 'no progress' point the user must have ignored the set of previously generated suggestions, shown as valid completions of the user's input just under the input window: if the user selects and clicks on any such completion its text is added to the input window. Going back to our example, the input of the token "Barack Obama" triggers the transition to S_2, where a range of new input tokens can be accepted, including the *question mark*, which marks the end of the question, and launches the translation of the sequence of accepted tokens to a SPARQL query followed by its execution.

Let us now consider another example: "What is the alma mater of the spouse of Barack Obama?". In this case, at S_1, the user would input the *property* "alma mater", whereby the system loops back to S_1, where it accepts the second *property*: "spouse". CANaLI accepts this 'chain' of properties because it is consistent with the underlying KB, which contains cases of "spouses" with an "alma mater". Now, the system is still in S_1, where the question is completed like in the previous example. These two simple examples show how the four basic states S_0, S_1, S_2, and S_F, support a large set of very common 'factual' questions asked by everyday users[3].

More complicated but nevertheless common questions are those adding constraints, i.e., query conditions. For instance, assume that the user wants to ask[4]: "Who are the spouses of politicians having birthplace equal to United States?". After the input of the fragment "Who are the spouses of" has taken the automaton to S_1, the token "politicians" is accepted as a *class* with "spouse" as a valid *property*, and this moves the automaton to S_2. In S_2, CANaLI can accept "having", and other uninterpreted connectives used as syntactic sugar, to move to S_3, where it will accept only a *property* related to a previously accepted token. In this case, "birthplace" can be accepted since "spouses" have this *property*. This example illustrates the ambiguities that beset all NL interfaces, no matter how sophisticated their parser is. In fact, also "politicians" have "birthplace" as a valid *property*. CANaLI tackles this problem by means of its interactive autocompletion system, that displays in real time all alternative interpretations that are compatible with the underlying KB. For instance, in the case at hand, CANaLI will display the two alternatives shown in Figure 2.

[2]The system response, indeed, is based on the context provided by the question typed so far and the underlying ontology, rather than just the current state and last token as a finite-state automaton would.

[3]Indeed, the factual questions asked most frequently on the web are definition questions (e.g., What is the EU?), that are even simpler.

[4]This provides a good example of the broken but effective English now supported by CANaLI.

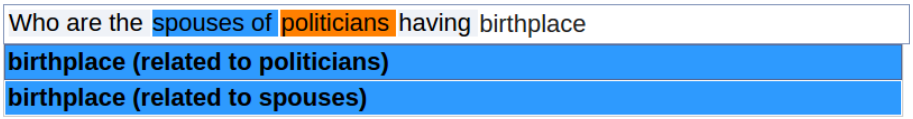


Figure 2: The autocompleter of CANaLI suggesting properties that can be related to spouses/politicians and having thelast word typed by the user, i.e., "birthplace", in their label

Once the user has explicitly made a choice by clicking on the result corresponding to her intention, the automaton moves to the state S_4, that expects an operator of some kind. Thus, the user can input "equal to", and the automaton moves to state S_5, that accepts the right-hand side of the constraint. In general, the right hand side of a constraint can be an element of the KB or a *literal*, as long as it is a valid value for the previously accepted *property*. In our example, the *entity* "United States" can be accepted, since it is a valid value for the *property* "birthplace". Now that the automaton is in S_2 again, the user can specify more constraints, or input the question mark, ending the question.

2.2 Experimental evaluation

A popular set of QA benchmarks has evolved as result of the annual QALD (Question Answering over Linked Data) challenge [40]. The benchmarks consist of sets of NL questions, each associated with its gold standard formulation in SPARQL, that produces answers against which the accuracy of the answers returned by the system under test are evaluated. The evaluation uses precision, recall and the F-measure that combines the two as: $F = (2 \times Precision \times Recall) \div (Precision + Recall)$. CANaLI entered the 2016 contest, QALD-6. On DBpedia questions CANaLI ranked first with an F-measure of 88%—i.e., 13% above the second-placed system, and with wider difference over the other competitors, including those that will be discussed in Section 5. CANaLI also performed well on previous QALD testbeds, including those based on MusicBrainz [39] KB and the three biomedical KBs: DrugBank [35], Diseasome [42], and SIDER [41]. In fact on these testbeds CANaLI performed as well or better than other systems in terms of precision, recall and F-measure.

2.3 Interacting Using a Controlled Natural Language

Given that CANaLI proved so accurate, it is natural that one should wonder about the extent in which restrictions imposed by the CNL makes it less user friendly than a full natural language interface. To answer this question, we show in Table 1 a sample of questions taken from previous QALD challenges, and their rephrased forms that were accepted by CANaLI.

Clearly, in all these questions, a reasonable rephrasing effort is demanded to the user. However, we found that such a rephrasing becomes quite natural once users understands that CANaLI expect questions on concepts or properties of concepts, expressed as nouns—plus optional specifications on properties that the concepts have. For instance, "Who is the creator of Captain America" is a question about a noun and thus accepted by CANaLI, whereas "who created Captain America" is not (see Q1). Likewise, "What is the height of Michael Jordan" is proper rephrasing for "how tall is Michael Jordan" (Q2), and "What is the date of Halloween" is the one for "when is Halloween" (Q3); likewise "What is the number of students of a University" is the rephrasing needed for "how many students does a University have" (Q4). Q4 presents an apparent difficulty, given by the actual name (used in DBpedia) of the Free University of Amsterdam. The autocompleter, in this case, helps the user by suggesting the correct denomination of the university. Q5 shows how constraints must be specified by using the properties. In this case, the "German lakes" are those lakes "having country Germany". The same limitation holds on questions that require counting. Thus, in Q6, in order to know "how often Nicole Kidman married", we need to count the number of values taken by this talented actress' property "spouse". Q7 shows an example of questions with two constraints. The preposition "by", quite common in natural language, is indeed ambiguous,

	Original question	Question rephrased for CANaLI
Q1	Who created the comic Captain America?	Who is the creator of Captain America?
Q2	How tall is Michael Jordan?	What is the height of Michael Jordan?
Q3	When is Halloween?	What is the date of Halloween?
Q4	How many students does the Free University in Amsterdam have?	What is the number of students of VU University Amsterdam?
Q5	Which rivers flow into a German lake?	What are the rivers flowing into lakes having country Germany?
Q6	How often did Nicole Kidman marry?	What is the count of spouse of Nicole Kidman?
Q7	Give me all books by William Goldman with more than 300 pages.	Give me the books having author William Goldman and number of pages greater than 300.
Q8	Does Abraham Lincoln's death place have a website?	Is there a website of death place of Abraham Lincoln?
Q9	Who is the youngest Darts player?	Who is the darts player with the greatest birth date?
Q10	Give me all actors who were born in Paris after 1950.	Give me the actors born in Paris having birth date greater than 1950-12-31.

Table 1: Questions taken from QALD-3 and QALD-4 challenges and their rephrasing for CANaLI

and only the background knowledge of the fact that "William Goldman" is an author (and not and editor, for instance) can disambiguate its corresponding meaning. The same principle holds for Yes/No questions, like Q8. The user must ask if a concept there exists, in this case the property "website" of the property "death place" of "Abraham Lincoln". Q9 gives an idea of how superlatives must be rephrased. In this case, the "youngest" player is a player with the "greatest birth date". Q10, instead, shows how constraints on dates must be rephrased (the autocompleter guides in typing dates in the standard format `yyyy-MM-dd`).

This analysis allows us to conclude that most questions require some reformulation in order to be accepted by CANaLI. However, we observe that (i) users quickly gain the needed skills once they start working with the system, and (ii) the introduction synonyms for concepts in the KB, and the addition of few simple rules to extend the grammar recognized by the system can go a long way toward making the interface of the system more natural, while still avoiding ambiguities. While these improvements are likely to confirm CANaLI as a leading CNL for DBpedia and, RDF KBs in general, the SWiPE system discussed next, can supports advanced queries requiring (i) complex aggregates, (ii) a mixture of structured and keyword conditions, and (iii) historical queries, i.e., queries that pose difficult research challenges for NL systems.

3 SWiPE: Query by Example on Wikipedia

The very Infoboxes that have been the source of the knowledge stored in DBpedia can be turned into active forms on which powerful queries can be specified using the By-Example Structured Query (BEStQ) approach of SWiPE [2, 3].

As a running example, let us suppose a user from Minneapolis is looking for a Law School with some desired requirements. She can start her search from the Wikipedia page of any Law school containing the Infobox: for instance the *University of Minnesota Law School* page. Then, by clicking on the SWiPE *bookmarklet*[5] the original Wikipedia page will be refreshed, showing the same page with the Infobox turned into an active form that allows the user to enter the desired conditions (as shown in Fig. 3(a)), plus a personalized toolbar showing at the bottom of the page (called SWiPE *bottom bar*) with additional features. In our case, the user wants to find Law schools located in New York with a high bar pass rate. She will then enter `>85%` as a first constraint for the *Bar pass rate* and `New York` in the *Location* field (replacing the original values in the Infobox). It is

[5] A special browser bookmark that actively interacts with the current web page

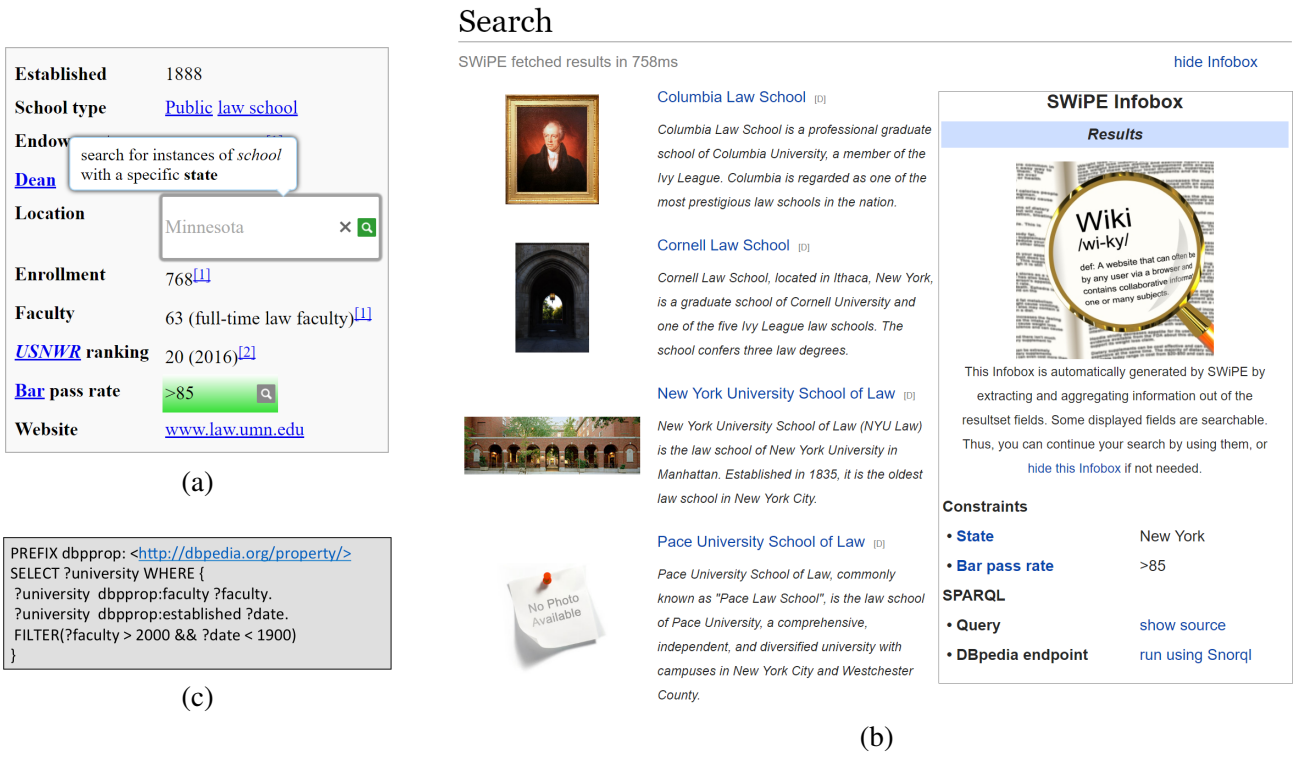


Figure 3: A QBE query on a university Infobox (a), its results (b) and a SPARQL query generated by SWiPE (c)

worth noting that the Location field in Wikipedia is recognized to correspond to the *state* property in DBpedia, as shown in the popup tooltip in Fig. 3(a) just before overwriting the original value "Minnesota" with the user constraint "New York". SWiPE also features an autocompletion-like functionality to help the user with the proper spelling for values[6]. The user may also choose to sort results based on some fields, for instance the *USNWR ranking*. Upon clicking on the green *lens* button available in each field and also in the Bottom Bar, SWiPE converts this two-line BEStQ specification into the equivalent (22-line long) SPARQL query. This is made possible by our on-the-fly algorithm that matches Infobox labels and values to the appropriate DBpedia RDF properties (`http://dbpedia.org/property/state` and `http//dbpedia.org/ontology/barPassRate` in the case at hand), greatly simplifying the user search experience by removing the burden of manually finding the correct properties and property names in the underlying KB. Finally, the SPARQL query is executed against our instance of DBpedia, on a Virtuoso server; results are then reformatted by SWiPE in a Wikipedia-like layout and presented to the user as shown in Fig. 3(b).

Since the original SWiPE prototype presented in [2], a number of improvements and new features have been added, including joins, aggregates, and the keyword-based retrieval capabilities of free-text search engines. Joins allow to input constraints on a different page w.r.t. the example pages used to start the search. For instance, in the previous example, the user may want to find only Law Schools belonging to universities established before 1900 and with a faculty size above 2000. These constraints will be entered on another example page, related to a University, and results will be filtered accordingly. The first page of the returned results also shows an Infobox that summarizes the query, and provides links to meta-information, such as the SPARQL query used, which for the example at hand is shown in Fig. 3(c).

Another example that we have frequently used [5] is that of finding cities with certain properties. A mobile

[6]This is an important feature to avoid unexpected empty results for constraints such as "USA" not matching DBpedia values stored as "United States"

version has been also developed [5], simplifying geolocation searches by featuring a touch interface for with location-based ranges queries. For that, our user only needs to find in Wikipedia the page of a familiar city, and then replace in its Infobox the existing values of the desired properties with conditions that specify the query. This is the well-known Query-By-Example (QBE) approach that is credited with bringing ease-of-use to relational databases and is even more desirable here, since it shields the users from having to discover the internal names and organization of DBpedia. It is also quite powerful, since it supports the specification of queries involving joins and aggregates.

Moreover, the user can still enter text conditions in the standard search box of Wikipedia to find the pages that satisfy both the structured query and the standard keyword-based retrieval made popular by web search engines. Our experiments show that this combination yields very powerful queries producing selective high-precision answers [3]. In order to detail the combination of keyword-based and structure, in addition to the two conditions previously entered in the Infobox, our prospective law student may also enter the words "`ivy league`" in the search box shown in the SWiPE bottom bar. As a result, SWiPE will provide the list of all Wikipedia pages satisfying both the structured conditions in the Infobox and those in the search box. Therefore, the last two entries in Figure 3(b) will no longer be in the answer, since 'New York University' and 'Pace University' are not in the Ivy League. This simple example clarifies the dramatic improvements in precision and recall delivered by BEStQ searches with respect to traditional keyword-only searches. In fact we claim that the combination of structured queries and keyword based searches routinely delivers accuracy levels (i.e., F-measure values) that are very hard to achieve with keywords alone, even when long-tail keywords and related optimization techniques are used. For instance, by using the following keyword combination "*Ivy League Law Schools New York bar pass rate,*" major search engines return hundreds of thousands of results, while Wikipedia gets a total of 53 answers, including some very surprising ones such as '*Christmas*', '*List of the Cosby Show episodes*' and '*List of Batman: The Brave and the Bold Character*', each of which contains all the specified keywords but is totally unrelated to the topic of interest.

In addition to supporting structured and keyword searches combined, the BEStQ approach supports well sorting, aggregate queries and historical queries that are not easily supported in CANaLI. While aggregate queries are simply implemented using an approach similar to that used in QBE, support for historical queries required nontrivial temporal extensions, that are described in [27] and summarized in the next section.

4 Cliopedia and SPARQL^{T}: Managing the History of KBs

As the real world evolves and the KBs are updated, the history of entities and their properties becomes of great interest. Table 2 shows the edit history of some attributes in Wikipedia Infoboxes. Such updates are quite common in many properties: e.g., on the average each value in the population property of the city pages is updated more than 7 times. This is not specific to Wikipedia, but also happens in other knowledge repositories such as Yago [23] and GovTrack [37]. Thus, users need query tools of comparable power and usability to explore such histories and flash-back to the past.

There is in fact a growing recognition of the importance of managing and querying the evolution history of knowledge bases in the technical literature, and several approaches [14, 21, 29, 30] have been proposed to support the queries on temporal RDF datasets.

In [8, 9] we proposed a vertically integrated system, RDF-TX (RDF Temporal eXpress), that efficiently supports the data management and query evaluation of large temporal RDF datasets while simplifying the temporal queries for SPARQL programmers and consequently, for end-user interfaces facilitating the expression of the same queries. To support the queries over the evolution history of KBs, we proposed efficient storage and index schemes for temporal RDF triples using multiversion B+ tree [4] and implemented a query engine which achieves fast query evaluation by taking advantage of comprehensive indices. We also built a query optimizer that generates efficient join orders using a cost-based model that exploit statistics collected on temporal RDF graphs. Thus,

Category	Property	Average Number of Updates
Software	Release	7.27
Player	Club	5.85
Country	GDP(PPP)	11.78
City	Population	7.16

Table 2: Statistics of Wikipedia Infobox Edit History

in RDF-TX, we provide a general and scalable solution for the problem of managing and querying massive temporal RDF by providing:

- A user-friendly BEStQ interface to view and query the history of knowledge base. Users can specify queries by entering simple conditions using a point-based temporal model. The system then translates these queries into equivalent SPARQL^T queries described next.
- SPARQL^T, a temporal extension of the structured query language SPARQL which simplifies the expression of temporal joins and eliminates the need for temporal coalescing.
- An efficient main memory system RDF-TX for managing temporal RDF data and evaluating SPARQL^T queries using a multiversion B+ tree (MVBT). This stores indexed temporal RDF triples in combination with an effective delta encoding scheme to reduce the storage overhead. RDF-TX also features a query optimizer that generates efficient join orders using the statistics of temporal RDF graphs.

4.1 Temporal RDF and By-Example Temporal Queries

Temporal RDF Model. RDF KBs can be presented as *(*subject, predicate, object) triples, which work well for static information, but not for the evolution history. For that we instead use the Temporal RDF model [14] that extends the RDF Graph with temporal elements. In the temporal RDF model, each *(*s, p, o) triple is annotated with a time element, producing the quadruplet *(*s, p, o, t). Table 3 shows the temporal RDF triples for Wikipedia Infobox of *San Diego*.

Consider the fact that Bob Filner served as the mayor of San Diego from December 4, 2012 to August 30, 2013. This fact can be represented as: ⟨*San Diego*, *mayor*, *Bob Filner*⟩:[*12/04/2012 . . . 08/30/2013*], while [*12/04/2012 . . . 08/30/2013*] represents all the days between 12/04/2012 and 08/30/2013. DAY provides the basic granularity in out temporal model.

Note that we adopt the point-based temporal model that dovetails with our BEStQ interface and simplifies the expression of temporal queries at the logical level; however at the physical level we retain the interval representation for efficiency reasons. Queries on the point-based model can be easily mapped into equivalent ones on the interval-based model for execution.

By-Example Temporal Queries. The Wikipedia Infobox of San Diego, clearly shows the current mayor, but not the past mayors, nor it allows us to find who was the mayor at certain date, nor to find the total population of the city when Bob Filner was mayor. To provide answers to these and similar questions, we have developed a system that extends Wikipedia Infobox with historical information and extend SPARQL and SWiPE with the ability of asking such temporal queries.

Our system support historical Infoboxes, where once a field is selected a pull down menu opens showing the history of that field. Thus, by selecting the mayor field in the Infobox the user actually sees its history with the last four mayors of San Diego. But in addition to viewing former values in the history of the entity properties, our user might enter query conditions in the fields of the Infobox that are identified as active fields by their highlighted backgrounds. Then, say that our user who saw the previous mayor in the pull-down menu of San Diego, now wants to find its population at the time when Bob Filner served as the mayor. For that, the user

Predicate	Object	Timestamp
Mayor	Bob Filner	12/04/2012 . . . 08/30/2013
	Todd Gloria	08/31/2013 . . . 03/02/2014
	Kevin Faulconer	03/03/2014 . . . *now*
Population	1322553	12/19/2012 . . . 10/01/2013
	1307402	10/02/2013 . . . 04/29/2014
	1345895	04/30/2014 . . . 05/21/2015
	1381069	05/22/2015 . . . *now*

Table 3: Temporal RDF Triples for *San Diego*

Figure 4: What was the population of San Diego when Bob Filner was the mayor?

can use the page of any city, including San Diego which is shown in Figure 4. Since all the fields are editable, the user enters "Bob Filner" in the *Government Mayor* and variable "?pop" in *City Population* and these two Infoboxes are set with the same temporal variable "?t" to indicate the temporal join.

Then our system generates and executes a SPARQL^T query, as discussed later. Again, it is important to remember that this query could have been entered in the Infobox of any city, and indeed since in our query conditions "San Diego" is not specified, our query will return the population of every city where Bib Filner served as the mayor.

4.2 SPARQL^T Query Language

To query the temporal RDF graphs, we propose a temporal extension of SPARQL called SPARQL^T. A SPARQL query consists of triple query patterns $\{s\ p\ o\}$, which corresponds to the RDF model $\{subject\ predicate\ object\}$. Similarly, SPARQL^T query is a set of SPARQL^T query patterns $\{s\ p\ o\ t\}$ where t refers to the temporal element in temporal RDF model. SPARQL^T supports all kinds of temporal queries and Allens operations [1]. Temporal selection queries that retrieve information about a previous snapshot of the KB can be easily expressed in SPARQL^T using one query pattern, as shown in Example 1.

EXAMPLE 1. When did Bob Filner served as the mayor of San Diego.
SELECT [?t]
$\{$**San_Diego mayor Bob_Filner ?t**$\}$

More complex queries often use temporal joins; in SPARQL^T joins are expressed by query patterns that share the same temporal element. For finding the population of San Diego when Bob Filner was mayor we write:

EXAMPLE 2. Find the population of San Diego when Bob Filner served as the mayor of San Diego.
SELECT ?pop [?t]
{**San_Diego mayor Bob_Filner ?t .**
San_Diego population ?pop ?t . }

SPARQL^T also introduces a set of built-in functions to faciliate the expression of complex temporal conditions, such as TSTART and TEND. Because of lack of space, we refer our reader to [9] for more query examples and complete semantics.

4.3 RDF-TX Query Engine

Previous works [14, 21, 29, 30] rely on relational databases/RDF engines to store and query temporal RDF triples, which results in complex and inefficient evaluation for temporal queries. Moreover, the indices for accelerating the processing of temporal queries proposed in the past only support a limited set of temporal queries. These limitations have been overcome by RDF-TX, which integrates indexing and query evaluation as follows:

Storage and Index. We choose in-memory MVBT indices to store temporal RDF and support fast SPARQL^T query evaluation. MVBT is a general temporal index that supports fast data update and range search. However, it comes with the large storage overhead, which is resolved in RDF-TX by applying delta compression between index entries.

Query Evaluation. Given a SPARQL^T query, we first parse the query and generate an execution plan in which every SPARQL^T query pattern is converted into a query pattern (k, i) on MVBT to retrieve all the temporal RDF triples with keys in range k and intervals overlapping interval i.

Optimization. In complex queries with multiple query patterns, inefficient execution plans may lead to large intermediate results and significantly slow down execution. Thus we introduce a query optimizer which use statistical estimations to find efficient join orders for complex queries. We introduced a temporal aggregate index, CMVSBT, which provides fast statistics estimation with a small storage overhead. Then, using CMVSBT, the optimizer employs the bottom-up dynamic programming strategy [26] to find the cost-optimal query plans.

5 Related Work

The problem of supporting user-friendly interfaces to DBpedia and other KBs has been widely recognized as important and attracted many research approaches, which because of space limitations we can only mention in a very succinct and incomplete fashion.

Approaches such as *exploratory browsing* [17] and *faceted search* [15, 16] allow users to formulate complex selection conditions, whereas SWiPE can also support joins and aggregates [2]. The NL approach is the one that, so far, has received most attention [12, 13, 20, 28]. These systems are remarkable, not only because of their number, but also because they use techniques that are quite diverse and tested at very different levels of F-measure on QALD testbeds. Among such systems we find **Xser** [32], **gAnswer** [33], **CASIA** [19], **Aqqu** [18], **Intui3** [6], **RTV** [10], besides **Squall2sparql** [7] where the sentences must be annotated by users, and **GFMed** [24] which is a CNL system specialized for the biomedical domain.

There has also been a significant amount of previous work on historical KBs. For instance, Gutierrez et al. [14] extended the RDF model with time elements and several approaches [11, 21, 29, 30] have been proposed to support the queries on temporal RDF datasets. Most previous works employ relational databases and RDF engines to store temporal RDF triples and rewrite temporal queries into SQL/SPARQL for evaluation. The languages proposed in those works use an interval-based temporal model which leads to complex expressions for temporal

queries, e.g., those requiring joins and coalescing [31]. At the physical level, previous approaches exploit indexes such as tGrin [29] to accelerate the processing of simple temporal queries, but they do not explore the use of general temporal indices and query optimization techniques.

6 Conclusions and Further Work

In this paper, we have described systems that let non-programmers access RDF KBs using either NL or friendly by-example interfaces. These systems integrates into simple and powerful framework the functions of question answering, query computation, and keyword-based document searching, which were traditionally viewed as separate functions supported by different technologies. Our current work focuses on improving the naturalness of CANaLI by careful addition of synonyms, while making sure that no ambiguity is introduced. Extensions to include keywords in NL questions, and to support historical questions are also under investigation. Combining structured Infobox conditions with CNL queries represents a topic for longer term research.

7 Acknowledgements

This research was supported in part by a 2015 Google Faculty Research Award.

References

[1] J. F. Allen. Maintaining Knowledge About Temporal Intervals. *Commun. ACM*, Vol. 26(11):832–843, 1983

[2] M. Atzori and C. Zaniolo. SWiPE: Searching WikiPedia by Example. *WWW (Companion Volume)*, 309–312, 2012.

[3] M. Atzori and C. Zaniolo. Expressivity and Accuracy of By-Example Structured Queries on Wikipedia. *WETICE*, 239–244, 2015.

[4] B. Becker, S. Gschwind, T. Ohler, B. Seeger, and P. Widmayer. An Asymptotically Optimal Multiversion B-tree *VLDB*, 264–275, 1996

[5] A. Dessi, A. Maxia, M. Atzori and C. Zaniolo. Supporting semantic web search and structured queries on mobile devices. *Semantic Search over the Web (SSW 2013), VLDB 2013 Workshops*, 5:1–5:4, 2013.

[6] C. Dima. Answering Natural Language Questions with Intui3. *Working Notes for CLEF 2014 Conference*, 1201–1211, 2014

[7] S. Ferré. SQUALL: The expressiveness of SPARQL 1.1 made available as a controlled natural language. *Data Knowledge Engineering*, Vol. 94(B): 163–188, 2014

[8] S. Gao, M. Chen, M. Atzori, J. Gu, and C. Zaniolo. SPARQLT and its User-Friendly Interface for Managing and Querying the history of RDF knowledge base. *ISWC*, 2015

[9] S. Gao, J. Gu, and C. Zaniolo. RDF-TX: A Fast, User-Friendly System for Querying the History of RDF Knowledge Bases. *EDBT*, 269–280, 2016

[10] C. Giannone, V. Bellomaria, and R. Basili. A HMM-based Approach to Question Answering against Linked Data. *Working Notes for CLEF 2013 Conference*, 2013

[11] F. Grandi. T-SPARQL: a TSQL2-like Temporal Query Language for RDF. *GraphQ*, 21–30, 2010

[12] B. F. Green Jr., A. K. Wolf, C. Chomsky, and K. Laughery. Baseball: An Automatic Question-answerer. *Western Joint IRE-AIEE-ACM Computer Conference*, 1961.

[13] P. Gupta and V. Gupta. A Survey of Text Question Answering Techniques. *International Journal of Computer Applications*, Vol. 53(4):1–8, 2012

[14] C. Gutierrez, C. A. Hurtado, and A. A. Vaisman. Introducing Time into RDF. *TKDE*, Vol 19(2):207–218, 2007

[15] J. Guyonvarch and S. Ferré. Scalewelis: a Scalable Query-based Faceted Search System on Top of SPARQL Endpoints. *Working Notes for CLEF Conference* , 2013

[16] R. Hahn, C. Bizer, C. Sahnwaldt, C. Herta, S. Robinson, M. Bürgle, H. Düwiger, and U. Scheel. Faceted Wikipedia Search. *Int. Conf. on Business Information Systems*, 2010

[17] L. Han, T. Finin, and A. Joshi. Schema-free structured querying of DBpedia data. *CIKM*, 2012

[18] B. Hannah and H. Elmar. More Accurate Question Answering on Freebase. *CIKM*, 1431–1440, 2015

[19] S. He, Y. Zhang, K. Liu, and J. Zhao. CASIA@V2: A MLN-based Question Answering System over Linked Data. *Working Notes for CLEF 2014 Conference*, 1249–1259, 2014.

[20] T. Kuhn. A Survey and Classification of Controlled Natural Languages. *CoRR*, abs/1507.01701, 2015

[21] E. Kuzey and G. Weikum. Extraction of Temporal Facts and Events from Wikipedia. *TempWeb*, 25–32, 2012

[22] J. Lehmann, R. Isele, M. Jakob, A. Jentzsch, D. Kontokostas, P. N. Mendes, S. Hellmann, M. Morsey, P. van Kleef, S. Auer, and C. Bizer. DBpedia - A Large-scale, Multilingual Knowledge Base Extracted from Wikipedia. *Semantic Web Journal*, Vol. 6(2):167–195, 2015

[23] J .Hoffart, F. M. Suchanek, K. Berberich, and G. Weikum. YAGO2: A Spatially and Temporally Enhanced Knowledge Base from Wikipedia *Artif. Intell.*, 28–61, 2013

[24] A. Marginean. GFMed: Question Answering over BioMedical Linked Data with Grammatical Framework. *Working Notes for CLEF 2014 Conference*, 1224–1235, 2014

[25] G. M. Mazzeo and C. Zaniolo. Answering Controlled Natural Language Questions on RDF Knowledge Bases. *EDBT*, 608-611, 2016

[26] G. Moerkotte and T. Neumann. Analysis of Two Existing and One New Dynamic Programming Algorithm for the Generation of Optimal Bushy Join Trees without Cross Products. *VLDB*, 930–941, 2006

[27] H. Mousavi, M. Atzori, S. Gao, and C. Zaniolo. Text-Mining, Structured Queries, and Knowledge Management on Web Document Corpora. *SIGMOD Record 43(3)*, 48–54, 2014

[28] S. R. Petrick. Natural Language Based Computer Systems. *IBM J. Res. Dev.*, Vol. 20(4):314–325, 1976

[29] A. Pugliese, O. Udrea, and V. S. Subrahmanian. Scaling RDF with Time. *WWW*, 605–614, 2008

[30] J. Tappolet and A. Bernstein. Applied Temporal RDF: Efficient Temporal Querying of RDF Data with SPARQL. *ESWC*, 308–322, 2009

[31] D. Toman. Point vs. Interval-based Query Languages for Temporal Databases. *PODS*, 58–67, 1996

[32] K. Xu, Y. Feng, and D. Zhao. Answering Natural Language Questions via Phrasal Semantic Parsing. *Working Notes for CLEF 2014 Conference*, 1260–1274, 2014

[33] L. Zou, R. Huang, H. Wang, J. X. Yu, W. He, and D. Zhao. Natural language question answering over RDF: a graph data driven approach. *SIGMOD*, 313–324, 2014

[34] `http://www.cyc.com/platform/opencyc`

[35] `http://www.drugbank.ca/`

[36] `http://www.geonames.org/`

[37] `https://www.govtrack.us/`

[38] `http://linkeddata.org/`

[39] `http://musicbrainz.org/`

[40] `http://www.sc.cit-ec.uni-bielefeld.de/qald`

[41] `http://sideeffects.embl.de/`

[42] `http://wifo5-03.informatik.uni-mannheim.de/diseasome/`

[43] `http://wiki.dbpedia.org/`

Data Engineering

It's FREE to join!

TCDE

tab.computer.org/tcde/

The Technical Committee on Data Engineering (TCDE) of the IEEE Computer Society is concerned with the role of data in the design, development, management and utilization of information systems.

- Data Management Systems and Modern Hardware/Software Platforms
- Data Models, Data Integration, Semantics and Data Quality
- Spatial, Temporal, Graph, Scientific, Statistical and Multimedia Databases
- Data Mining, Data Warehousing, and OLAP
- Big Data, Streams and Clouds
- Information Management, Distribution, Mobility, and the WWW
- Data Security, Privacy and Trust
- Performance, Experiments, and Analysis of Data Systems

The TCDE sponsors the International Conference on Data Engineering (ICDE). It publishes a quarterly newsletter, the Data Engineering Bulletin. If you are a member of the IEEE Computer Society, you may join the TCDE and receive copies of the Data Engineering Bulletin without cost. There are approximately 1000 members of the TCDE.

Join TCDE via Online or Fax

ONLINE: Follow the instructions on this page:

www.computer.org/portal/web/tandc/joinatc

FAX: Complete your details and fax this form to **+61-7-3365 3248**

Name _____

IEEE Member # _____

Mailing Address _____

_____

Country _____

Email _____

Phone _____

TCDE Mailing List

TCDE will occasionally email announcements, and other opportunities available for members. This mailing list will be used only for this purpose.

Membership Questions?

Xiaoyong Du
Key Laboratory of Data Engineering and Knowledge Engineering
Renmin University of China
Beijing 100872, China
duyong@ruc.edu.cn

TCDE Chair

Xiaofang Zhou
School of Information Technology and Electrical Engineering
The University of Queensland
Brisbane, QLD 4072, Australia
zxf@uq.edu.au

IEEE Computer Society
1730 Massachusetts Ave, NW
Washington, D.C. 20036-1903

Non-profit Org.
U.S. Postage
PAID
Silver Spring, MD
Permit 1398