Building, Maintaining, and Using Knowledge Bases: A Report from the Trenches

Omkar Deshpande[1], Digvijay S. Lamba[1], Michel Tourn[2],
Sanjib Das[3], Sri Subramaniam[1], Anand Rajaraman, Venky Harinarayan, AnHai Doan[1,3]
[1]@WalmartLabs, [2]Google, [3]University of Wisconsin-Madison

ABSTRACT

A knowledge base (KB) contains a set of concepts, instances, and relationships. Over the past decade, numerous KBs have been built, and used to power a growing array of applications. Despite this flurry of activities, however, surprisingly little has been published about the end-to-end process of building, maintaining, and using such KBs in industry. In this paper we describe such a process. In particular, we describe how we build, update, and curate a large KB at Kosmix, a Bay Area startup, and later at WalmartLabs, a development and research lab of Walmart. We discuss how we use this KB to power a range of applications, including query understanding, Deep Web search, in-context advertising, event monitoring in social media, product search, social gifting, and social mining. Finally, we discuss how the KB team is organized, and the lessons learned. Our goal with this paper is to provide a real-world case study, and to contribute to the emerging direction of building, maintaining, and using knowledge bases for data management applications.

Categories and Subject Descriptors

H.2.4 [**Information Systems**]: Database Management - Systems

Keywords

Knowledge base; taxonomy; Wikipedia; information extraction; data integration; social media; human curation

1. INTRODUCTION

A knowledge base (KB) typically contains a set of concepts, instances, and relations. Well-known examples of KBs include DBLP, Google Scholar, Internet Movie Database, YAGO, DBpedia, Wolfram Alpha, and Freebase. In recent years, numerous KBs have been built, and the topic has received significant and growing attention, in both industry and academia (see the related work). This attention comes from the fact that KBs are increasingly found to be critical to a wide variety of applications.

For example, search engines such as Google and Bing use global KBs to understand and answer user queries (the Google knowledge graph [20] is in fact such a KB). So do e-commerce Web sites, such as amazon.com and walmart.com, using product KBs. As another example, the iPhone voice assistant Siri uses KBs to parse and answer user queries. As yet another example, echonest.com builds a large KB about music, then uses it to power a range of applications, such as recommendation, playlisting, fingerprinting, and audio analysis. Other examples include using KBs to find domain experts in biomedicine, to analyze social media, to search the Deep Web, and to mine social data.

Despite this flurry of activities, however, surprisingly very little has been published about how KBs are built, maintained, and used *in industry*. Current publications have mostly addressed isolated aspects of KBs, such as the initial construction, and data representation and storage format (see the related work section). Interesting questions remain unanswered, such as "how do we maintain a KB over time?", "how do we handle human feedback?", "how are schema and data matching done and used?", "the KB will not be perfectly accurate, what kinds of application is it good for?", and "how big of a team do we need to build such a KB, and what the team should do?". As far as we can tell, no publication has addressed such questions and described the end-to-end process of building, maintaining, and using a KB in industry.

In this paper we describe such an end-to-end process. In particular, we describe how we build, maintain, curate, and use a global KB at Kosmix and later at WalmartLabs. Kosmix was a startup in the Bay Area from 2005 to 2011. It started with Deep Web search, then moved on to in-context advertising and semantic analysis of social media. It was acquired by Walmart in 2011 and converted into WalmartLabs, which is working on product search, customer targeting, social mining, and social commerce, among others. Throughout, our global KB lies at the heart of, and powers the above applications.

We begin with some preliminaries, in which we define the notion of KBs, then distinguish two common types of KBs: global and domain-specific. We also distinguish between ontology-like KBs, which attempt to capture all relevant concepts, instances, and relationships in a domain, and source-specific KBs, which integrate a set of given data sources. We discuss the implications of each of these types. Our KB is a large global, ontology-like KB, which attempts to capture all important and popular concepts, instances, and relationships in the world. It is similar to Freebase and Google's knowledge graph in this aspect.

Building the KB: We then discuss building the KB. Briefly, we convert Wikipedia into a KB, then integrate it with additional data sources, such as Chrome (an automobile source), Adam (health), MusicBrainz, City DB, and Yahoo Stocks. Here we highlight several interesting aspects that have not commonly been discussed in the KB construction literature. First, we show that converting Wikipedia into a taxonomy is highly non-trivial, because each node in the Wikipedia graph can have multiple paths (i.e., lineages) to the root. We describe an efficient solution to this problem. Interestingly, it turned out that different applications may benefit from different lineages of the same node. So we convert Wikipedia into a taxonomy but do preserve all lineages of all Wikipedia nodes.

Second, extracting precise relationships from Wikipedia (and indeed from any non-trivial text) is notoriously difficult. We show how we sidestep this problem and extract "fuzzy relationships" instead, in the form of a relationship graph, then use this fuzzy relationship graph in a variety of real-world applications.

Third, we discuss extracting meta information for the nodes in the KB, focusing in particular on social information such as Wikipedia traffic statistics and social contexts. For example, given the instance "Mel Gibson", we store the number of times people click on the Wikipedia page associated with it, the most important keywords associated with it in social media in the past 1 hour (e.g., "mel", "crash", "maserati"), and so on. Such meta information turns out to be critical for many of our applications.

Finally, we discuss adding new data sources to the KB constructed out of Wikipedia. We focus in particular on matching external instances into those in the KB, and briefly discuss how taxonomy matching and entity instance matching are interwoven in our algorithm.

Maintaining and Curating the KB: Building the initial KB is difficult, but is just the very first step. In the long run, maintaining and curating the KB pose the most challenges and incur most of the workload. We discuss how to refresh the KB every day by rerunning most of it from the scratch (and the reason for doing so). We then discuss a major technical challenge: how to curate the KB and preserve the curation after refreshing the KB. Our solution is to capture most of the human curation in terms of commands, and then apply these commands again when we refresh the KB.

Using the KB: In the last major part of the paper, we discuss how we have used the above KB for a variety of applications, including parsing and understanding user queries, Deep Web search, in-context advertising, semantic analysis of social media, social gifting, and social mining. We discuss the insights gleaned from using an imperfect KB in real-world applications.

Finally, we describe the organization of the team that works on the KB, statistics regarding the KB, lessons learned, future work, and comparison to related work. A technical report version of this paper, with more details, can be found at `pages.cs.wisc.edu/~anhai/papers/kcs-tr.pdf`.

2. PRELIMINARIES

Knowledge Bases: A knowledge base typically consists of a set of concepts $C_1, \ldots, C_n$, a set of instances I_i for each concept C_i, and a set of relationships $R_1, \ldots, R_m$ among the concepts.

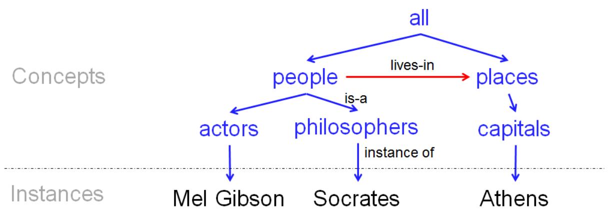


Figure 1: A tiny example of a KB

We distinguish a special relationship called "is-a", which specifies that a concept A is a kind of a concept B (e.g., Professors is a kind of People). The "is-a" relationships impose a taxonomy over the concepts C_i. This taxonomy is a tree, where nodes denote the concepts and edges the "is-a" relationships, such that an edge $A \rightarrow B$ means that concept B is a kind of concept A. Figure 1 shows a tiny KB, which illustrates the above notions.

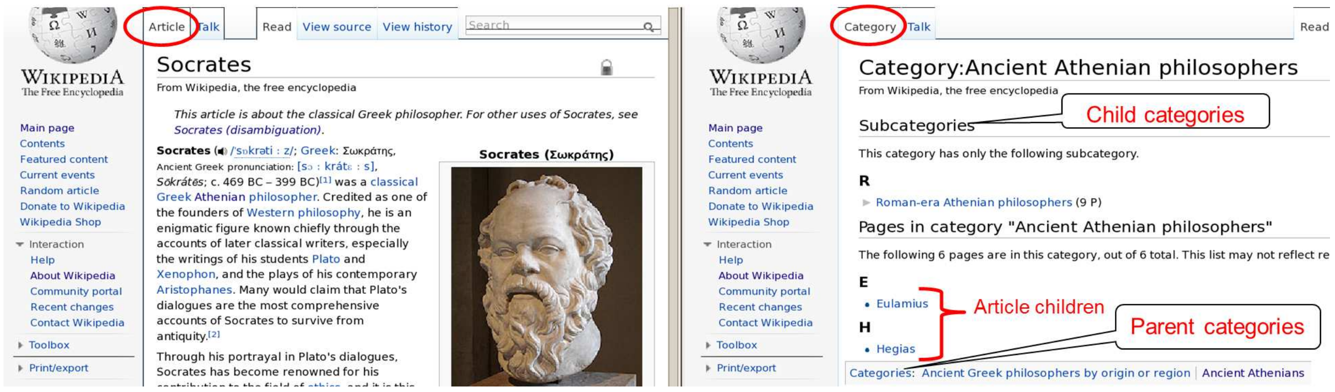
In many KBs, the set of instances of a parent node (in the taxonomy) is the union of the instances of the child nodes. In our context, we do not impose this restriction. So a node A may have instances that do not belong to any of A's children. Furthermore, KBs typically also contain many domain integrity constraints. In our context, these constraints will appear later, being specified by our developers as a part of the human curation process (see Section 4.2).

Domain-Specific KBs vs. Global KBs: We distinguish two types of KBs. A *domain-specific KB* captures concepts, instances, and relationships of a relatively well-defined domain of interest. Examples of such KBs include DBLP, Google Scholar, DBLife, echonest, and product KBs being built by e-commerce companies. A *global KB* attempts to cover the entire world. Examples of such KBs include Freebase, Google's knowledge graph, YAGO, DBpedia, and the collection of Wikipedia infoboxes.

This distinction is important because depending on the target applications, we may end up building one type or the other. Furthermore, it is interesting to consider whether we need domain-specific KBs at all. To power most of real-world applications, is it sufficient to build just a few large global KBs? If so, then perhaps they can be built with brute force, by big Internet players with deep pocket. In this case, developing efficient methodologies to build KBs presumably become far less important.

We believe, however, that while global KBs are very important (as ours attests), there is also an increasing need to build domain-specific KBs, and in fact, we have seen this need in many domains. Consequently, it is important to develop efficient methodologies to help domain experts build such KBs as fast, accurately, and inexpensively as possible.

Ontology-like KBs vs. Source-Specific KBs: We also distinguish between ontology-like and source-specific KBs. An *ontology-like KB* attempts to capture *all* important concepts, instances, and relationships in the target domain. It functions more like a domain ontology, and is *comprehensive* in certain aspects. For example, DBLP is an ontology-like KB. It does not capture all possible relationships in



Socrates

From Wikipedia, the free encyclopedia

This article is about the classical Greek philosopher. For other uses of Socrates, see Socrates (disambiguation).

Socrates (/ˈsɒkrətiːz/; Greek: Σωκράτης, Ancient Greek pronunciation: [sɔːkrátɛːs], *Sōkrátēs*; c. 469 BC – 399 BC)[1] was a classical Greek Athenian philosopher. Credited as one of the founders of Western philosophy, he is an enigmatic figure known chiefly through the accounts of later classical writers, especially the writings of his students Plato and Xenophon, and the plays of his contemporary Aristophanes. Many would claim that Plato's dialogues are the most comprehensive accounts of Socrates to survive from antiquity.[2]

Through his portrayal in Plato's dialogues, Socrates has become renowned for his

Socrates (Σωκράτης)

Category:Ancient Athenian philosophers

From Wikipedia, the free encyclopedia

Subcategories

This category has only the following subcategory.

R

▶ Roman-era Athenian philosophers (9 P)

Pages in category "Ancient Athenian philosophers"

The following 6 pages are in this category, out of 6 total. This list may not reflect re

E

- Eulamius

H

- Hegias

Categories: Ancient Greek philosophers by origin or region | Ancient Athenians

Figure 2: Two main kinds of Wikipedia pages - article page (left) and category page (right)

the CS publication domains, but is comprehensive in that it contains all publications of the most important publication venues. An ontology-like KB can also be viewed as a kind of "dictionary" for the target domain.

Source-specific KBs, in contrast, are built from a given set of data sources (e.g., RDBMSs, semi-structured Web pages, text), and cover these sources only. For example, an intelligence analyst may want to build a KB that covers all articles in the past 30 days from all major Middle-East newspapers, for querying, mining, and monitoring purposes.

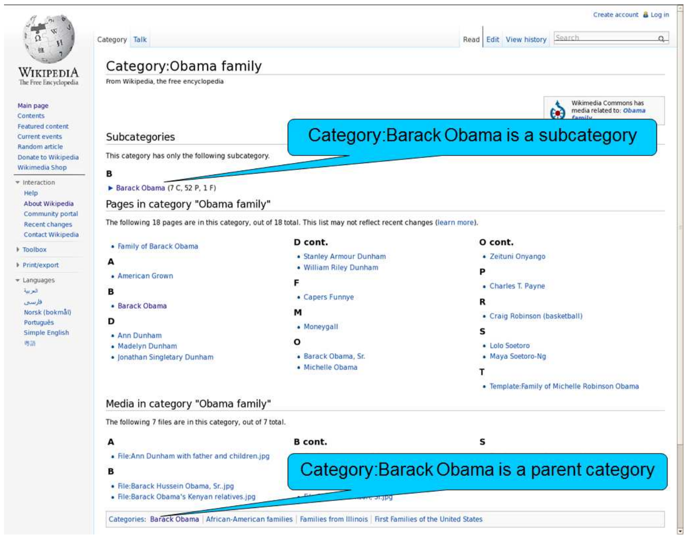
The above distinction is important for two reasons. First, building each type of KB requires a slightly different set of methods. Building a source-specific KB is in essence a data integration problem, where we must integrate data from a given set of data sources. The KB will cover the concepts, instances, and relations found in these sources and these only. In contrast, building an ontology-like KB requires a slightly different mindset. Here we need to think "I want to obtain *all* information about this concept and its instances, where in the world can I obtain this information, in the most clean way, even if I have to buy it?". So here the step of data source acquisition becomes quite prominent, and obtaining the right data source often makes the integration problem much easier.

Second, if we already have an ontology-like KB, building source-specific KBs in the same domain becomes much easier, because we can use the ontology-like KB as a domain dictionary to help locate and extract important information from the given data sources (see Section 6). This underscores the importance of building ontology-like KBs, and efficient methodologies to do so.

Given that our applications are global in scope, our goal is to build a global KB. We also want this KB to be ontology-like, in order to use it to build many source-specific KBs. We now describe how we build, maintain, curate, and use this global, ontology-like KB.

3. BUILDING THE KNOWLEDGE BASE

As we hinted earlier, Wikipedia is not a KB in the traditional sense, and converting Wikipedia into a KB is a nontrivial process. The key steps of this process are: (1) constructing the taxonomy tree from Wikipedia, (2) constructing a DAG on top of the taxonomy, (3) extracting relationships from Wikipedia, (4) adding metadata, and (5) adding other data sources. We now elaborate on these steps.


Create account
Log in
Category
Talk
Read
Edit
View history
Search
Wikimedia Commons has media related to: Obama family
Category:Barack Obama is a subcategory
Category:Barack Obama is a parent category

Category:Obama family

From Wikipedia, the free encyclopedia

Subcategories

This category has only the following subcategory.

B

▶ Barack Obama (7 C, 52 P, 1 F)

Pages in category "Obama family"

The following 18 pages are in this category, out of 18 total. This list may not reflect recent changes (learn more).

- Family of Barack Obama

A

- American Grown

B

- Barack Obama

D

- Ann Dunham
- Madelyn Dunham
- Jonathan Singletary Dunham

D cont.

- Stanley Armour Dunham
- William Riley Dunham

F

- Capers Funnye

M

- Moneygall

O

- Barack Obama, Sr.
- Michelle Obama

O cont.

- Zeituni Onyango

P

- Charles T. Payne

R

- Craig Robinson (basketball)

S

- Lolo Soetoro
- Maya Soetoro-Ng

T

- Template:Family of Michelle Robinson Obama

Media in category "Obama family"

The following 7 files are in this category, out of 7 total.

A

- File:Ann Dunham with father and children.jpg

B

- File:Barack Hussein Obama, Sr..jpg
- File:Barack Obama's Kenyan relatives.jpg

B cont.

S

Categories: Barack Obama | African-American families | Families from Illinois | First Families of the United States

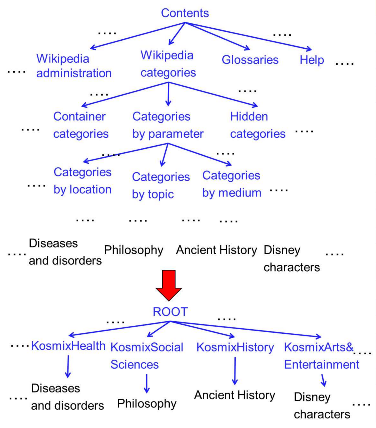
Figure 3: Cyclic references in a Wikipedia category page

3.1 Constructing the Taxonomy Tree

1. Crawling Wikipedia: We maintain an in-house mirror of Wikipedia and keep it continuously updated, by monitoring the Wikipedia change log and pulling in changes as they happen. Note that an XML dump of Wikipedia pages is also available at download.wikimedia.org/enwiki. However, we maintain the Wikipedia mirror because we want to update our KB daily, whereas the XML dump usually gets updated only every fortnight.

2. Constructing the Wikipedia Graph: There are two main kinds of pages in Wikipedia: article pages and category pages (see Figure 2). An *article page* describes an instance. A *category page* describes a concept. In particular the page lists the sub-categories, parent categories, and article children. Other Wikipedia page types include Users, Templates, Helps, Talks, etc., but we do not parse them. Instead, we parse the XML dump to construct a graph where each node refers to an article or a category, and each edge refers to a Wikipedia link from a category X to a subcategory of X or from a category X to an article of X.

Ideally, the articles (i.e., instances) and categories (i.e.,



Name of Vertical	Children	Descendants	Leaves
Products	17	58555	15404
Health	16	229769	78351
Arts&Entertainment	14	1470304	539086
InfoTypeRoot	13	76	62
Sports	12	927296	411975
History	11	533387	155908
Adult	10	12493	3185
Religion	9	273202	78537
Home&Leisure	8	184405	52315
Education	7	286684	107469
Travel	6	1730785	686284
SocialSciences	6	182453	55580
Writing&Language	6	540446	173952
Business&Finance	5	283424	98544
Hidden	5	3252	1073
Technology	4	265023	78775
Politics&Govt.	3	885208	288269
People	3	295672	130958
Science&Nature	3	855792	322689
Vehicles&Transportation	2	553664	174267
Food&Agriculture	2	105396	28584
CreatorTypeRoot	0	4	3
Local	0	4	3

Figure 4: Constructing the top levels of our taxonomy and the list of verticals

concepts) should form a taxonomy, but this is not the case. The graph produced is cyclic. For example, Figure 3 shows the category page "Category:Obama family". This category page lists the page "Category:Barack Obama" as a subcategory. But it also lists (at the bottom of the page) the same page as a parent category. So this page and the page "Category:Barack Obama" form a cycle. As a result, the Wikipedia graph is a directed cyclic graph.

Another problem with this graph is that its top categories, shown in Figure 4 as "Contents", "Wikipedia administration", "Wikipedia categories", etc. are not very desirable from an application point of view. The desirable categories, such as "Philosophy" and "Diseases and Disorders" are buried several levels down in the graph. To address this problem, we manually create a set of very high-level concepts, such as "KosmixHealth", "KosmixSocialSciences", and "KosmixHistory", then place them as the children of a root node. We call these nodes *verticals*. Next, we place the desirable Wikipedia categories in the first few levels of the graph as the children of the appropriate verticals, see Figure 4. This figure also lists all the verticals in our KB.

3. Constructing the Taxonomy Tree: We now describe how to construct a taxonomic tree out of the directed cyclic Wikipedia graph. Several algorithms exist for converting such a graph to a spanning tree. Edmonds' algorithm (a.k.a. Chu-Liu/Edmonds') [12, 9] is a popular algorithm for finding the maximum or minimum number of optimum branchings in a directed graph. We use Tarjan [23], an efficient implementation of this algorithm.

Tarjan prunes edges based on associated weights. So we assign to each edge in our graph a weight vector as follows. First, we tag all graph edges: category-article edges with *warticle* and category-subcategory edges with *wsubcat*. If an article and its parent category happen to have the same name (e.g., "Article:Socrates" and "Category:Socrates"), the edge between them is tagged with *artcat*. We then assign default weights to the tags, with *artcat*, *wsubcat*, and *warticle* receiving weights in decreasing order.

Next, we compute a host of signals on the edges. Examples include:

- *Co-occurrence count of the two concepts forming the edge on the Web:* Given two concepts A and B, we compute a (normalized) count of how frequently they occur together in Web pages, using a large in-house Web corpus. Intuitively, the higher this count, the stronger the relationship between the two concepts, and thus the edge between them.

- *Co-occurrence count of the two concepts forming the edge in lists:* This is similar to the above signal, except that we look for co-occurrences in the same list in Wikipedia.

- *Similarity between the concept names:* We compute the similarity between the concept names, using a set of rules that take into account how Wikipedia categories are named. For example, if we see two concepts with names such as "Actors" and "Actors by Nationality" (such examples are very common in Wikipedia), we know that they form a clean parent-child relationship, so we assign a high signal value to the edge between them.

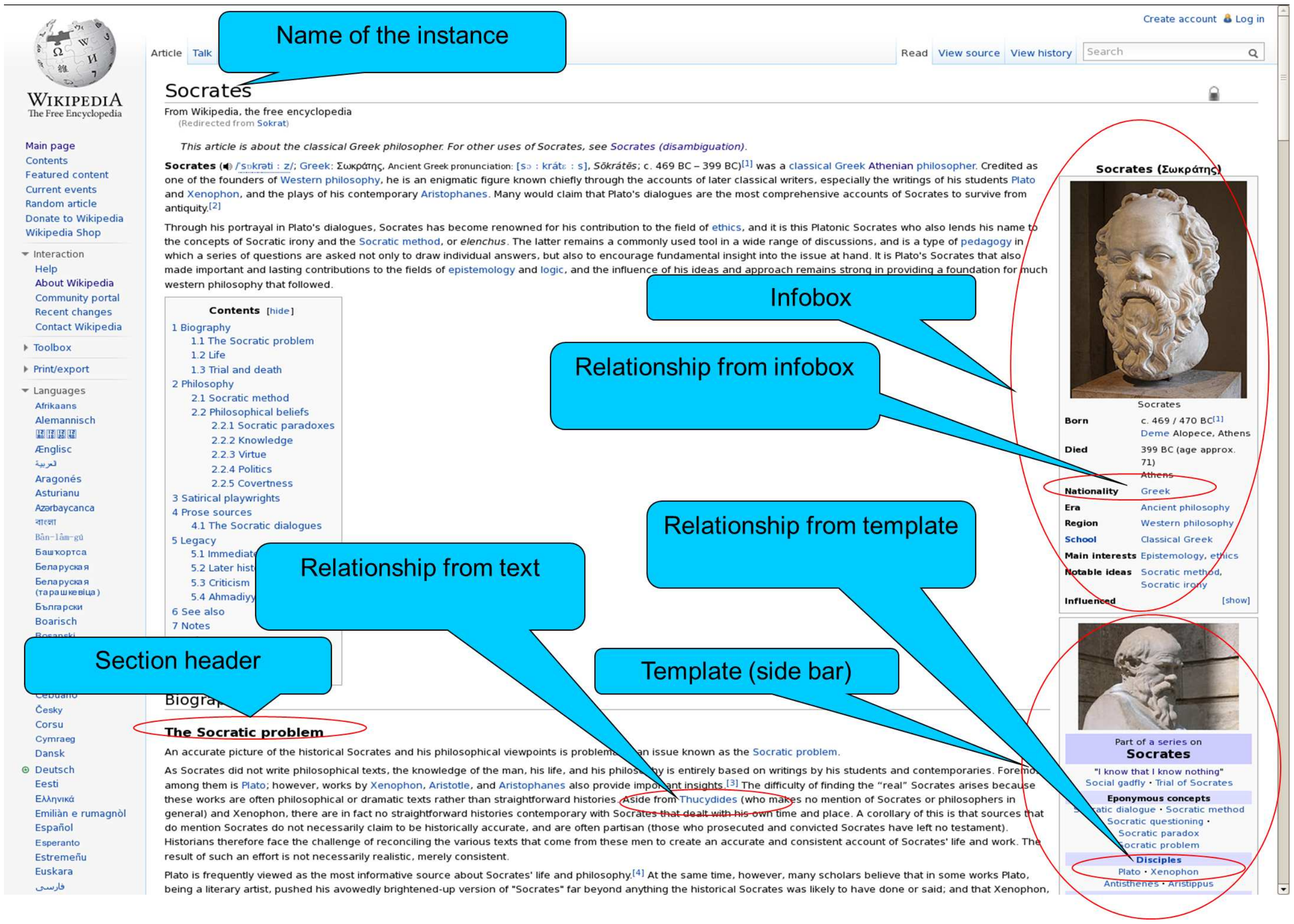
Next, an analyst may (optionally) assign two types of weight to the edges: recommendation weights and subtree preference weights. The analyst can recommend an ancestor A to a node B by assigning a recommendation weight to the edges in the path from A to B. He or she can also suggest that a particular subtree in the graph is highly relevant and should be preserved as far as possible during pruning. To do so, the analyst assigns a high subtree preference weights to all the edges in that subtree, using an efficient command language. For more details see Section 4.2, where we explain the role of an analyst in maintaining and curating the KB.

Now we can assign to each edge in the graph a weight vector, where the weights are listed in decreasing order of importance: <recommendation weight, subtree preference weight, tag weight, signal 1, signal 2, ...>. Comparing two edges means comparing the recommendation weights, then breaking tie by comparing the next weights, and so on. The standard Edmonds' algorithm works with just a single weight per edge. So we modified it slightly to work with the weight vectors.

3.2 Constructing the DAG

To motivate DAG construction, suppose that the article page "Socrates" is an instance of category "Forced Suicide", which in turn is a child of two parent categories: "Ancient Greek Philosophers" and "5th Century BC Philosophers". These two categories in turn are children of "Philosophers", which is a child of "ROOT".

Then "Forced Suicide" has two lineages to the root: L_1 = Force Suicide - Ancient Greek Philosophers - Philosophers - ROOT, and L_2 = Force Suicide - 5th Century BC Philosophers - Philosophers - ROOT. When constructing the taxonomic tree, we can select only one lineage (since each node can have only one path to the root). So we may select lineage L_1 and delete lineage L_2. But if we do so, we lose information. It turns out that keeping other lineages such as L_2 around can be quite beneficial for a range of applications. For example, if a user query refers to "5th century BC", then keeping L_2 will boost the relevance of "Socrates" (since the above phrase is mentioned on a path from "Socrates" to the root). As yet another example, Ronald Reagan has two paths to the root, via "Actors" and "US Presidents", and it is desirable to keep both, since an application may make use of



WIKIPEDIA
The Free Encyclopedia

Main page
Contents
Featured content
Current events
Random article
Donate to Wikipedia
Wikipedia Shop

Interaction
Help
About Wikipedia
Community portal
Recent changes
Contact Wikipedia

Toolbox

Print/export

Languages
Afrikaans
Alemannisch
Ænglisc
العربية
Aragonés
Asturianu
Azərbaycanca
বাংলা
Bân-lâm-gú
Башҡортса
Беларуская
Беларуская (тарашкевіца)
Български
Boarisch
Česky
Corsu
Cymraeg
Dansk
Deutsch
Eesti
Ελληνικά
Emiliàn e rumagnòl
Español
Esperanto
Estremeñu
Euskara
فارسی

Create account Log in

Article Talk Read View source View history Search

Socrates

From Wikipedia, the free encyclopedia
(Redirected from Sokrat)

This article is about the classical Greek philosopher. For other uses of Socrates, see Socrates (disambiguation).

Socrates (/ˈsɒkrəti ː z/; Greek: Σωκράτης, Ancient Greek pronunciation: [sɔ ː krátɛ ː s], *Sōkrátēs*; c. 469 BC – 399 BC)[1] was a classical Greek Athenian philosopher. Credited as one of the founders of Western philosophy, he is an enigmatic figure known chiefly through the accounts of later classical writers, especially the writings of his students Plato and Xenophon, and the plays of his contemporary Aristophanes. Many would claim that Plato's dialogues are the most comprehensive accounts of Socrates to survive from antiquity.[2]

Through his portrayal in Plato's dialogues, Socrates has become renowned for his contribution to the field of ethics, and it is this Platonic Socrates who also lends his name to the concepts of Socratic irony and the Socratic method, or *elenchus*. The latter remains a commonly used tool in a wide range of discussions, and is a type of pedagogy in which a series of questions are asked not only to draw individual answers, but also to encourage fundamental insight into the issue at hand. It is Plato's Socrates that also made important and lasting contributions to the fields of epistemology and logic, and the influence of his ideas and approach remains strong in providing a foundation for much western philosophy that followed.

Contents [hide]

Biogra

The Socratic problem

An accurate picture of the historical Socrates and his philosophical viewpoints is problem an issue known as the Socratic problem.

As Socrates did not write philosophical texts, the knowledge of the man, his life, and his philos by is entirely based on writings by his students and contemporaries. Foremo among them is Plato; however, works by Xenophon, Aristotle, and Aristophanes also provide impo ant insights.[3] The difficulty of finding the "real" Socrates arises because these works are often philosophical or dramatic texts rather than straightforward histories. Aside from Thucydides (who makes no mention of Socrates or philosophers in general) and Xenophon, there are in fact no straightforward histories contemporary with Socrates that dealt with his own time and place. A corollary of this is that sources that do mention Socrates do not necessarily claim to be historically accurate, and are often partisan (those who prosecuted and convicted Socrates have left no testament). Historians therefore face the challenge of reconciling the various texts that come from these men to create an accurate and consistent account of Socrates' life and work. The result of such an effort is not necessarily realistic, merely consistent.

Plato is frequently viewed as the most informative source about Socrates' life and philosophy.[4] At the same time, however, many scholars believe that in some works Plato, being a literary artist, pushed his avowedly brightened-up version of "Socrates" far beyond anything the historical Socrates was likely to have done or said; and that Xenophon,

Socrates (Σωκράτης)

Socrates

Born	c. 469 / 470 BC[1] Deme Alopece, Athens
Died	399 BC (age approx. 71) Athens
Nationality	Greek
Era	Ancient philosophy
Region	Western philosophy
School	Classical Greek
Main interests	Epistemology, ethics
Notable ideas	Socratic method, Socratic irony
Influenced	[show]

Part of a series on
Socrates
"I know that I know nothing"
Social gadfly · Trial of Socrates
Eponymous concepts
Socratic dialogue · Socratic method
Socratic questioning ·
Socratic paradox
Socratic problem
Disciples
Plato · Xenophon
Antisthenes · Aristippus

Figure 5: Extraction of relationships from a Wikipedia page

either. We would want to designate a lineage as the primary one (e.g., "US Presidents" in the case of Ronald Reagan since he is more well known for that), by making that lineage as a part of the taxonomy, while keeping the remaining lineages (e.g., "Actors").

Consequently, we want to construct a primary taxonomic tree from the Wikipedia graph, but we also want to keep all other lineage information, in the form of a DAG that subsumes the taxonomic tree. Recall that the Wikipedia graph has cycles. We however do not want such cycles because it does not make sense to have "category - sub category" edges go in cycle. Because of these, we construct the desired DAG as follows. After obtaining the taxonomic tree T, we go back to the original Wikipedia graph G and assign high weights to the edges of T. Next, we remove cycles in G by running multiple iterations of DFS. In each DFS iteration, if we detect a back edge then there is a cycle. We break it by deleting the edge with the lowest weight, as given by the weight vector. We stop when DFS does not detect any more cycle.

3.3 Extracting Relationships from Wikipedia

Recall from Section 2 that a KB has a finite set of *predefined* relationships, such as lives-in(people, location) and write(author, book). An instance of such a relationship involves concept instances. For example, lives-in(Socrates, Athens) is an instance of relationship lives-in(people, location).

In principle, we can try to define a set of such relationships and then extract their instances. But this raises two problems. First, Wikipedia contains hundreds of thousands, if not millions, of potentially interesting relationships, and this set changes daily. So defining more than just a handful of relationships quickly becomes impractical. Second, and more seriously, accurately extracting relationship instances from any non-trivial text is well known to be difficult, and computationally expensive.

For these reasons, we take a pragmatic approach in which we do not pre-define relationships nor attempt to extract their instances. Instead, we extract *free-form* relationship instances between concept instances. For example, suppose the article page "Barack Obama" has a section titled "Family", which mentions the article page "Bo (Dog)". Then we create a relationship instance <Barack Obama, Bo (Dog), Family>, indicating that "Barack Obama" and "Bo (Dog)" have a relationship "Family" between them.

In general, extracted relationship instances have the form <name of concept instance 1, name of concept instance 2, some text indicating a relationship between them>. We extract these relationship instances as follows:

- *Extraction from infoboxes:* An infobox at the top right-hand corner of an article page summarizes the page and provides important statistics. We write a set of rules to extract relationship instances from such infoboxes. For example, from the page in Figure 5, we can extract <Socrates, Greek, Nationality>.

- *Extraction from templates:* Templates describe materials that may need to be displayed on multiple pages. We extract relationship instances from two common templates: hat notes (i.e., short notes at the top of an article or section body, usually referring to related articles) and side bars. Figure 5 shows a sample side bar. From it we can extract for example <Socrates, Plato, Disciples>.

- *Extraction from article text:* We use a set of rules to extract potentially interesting relationship instances from the text of articles. For example, the page "Socrates" in Figure 5 has a section titled "The Socratic Problem", which mentions "Thucydides". From this we can extract <Socrates, Thucydides, Socratic Problem> as a relation instance. Other rules concern extracting from lists, tables, and so on.

Thus, using a relatively small set of rules, we can extract tens of millions of free-form relationship instances from Wikipedia. We encode these relationships in a *relationship graph*, where the nodes denote the concept instances, and the edges denote the relation instances among them.

When using the relationship graph, we found that some applications prefer to work with a smaller graph. So on occasions we may need to prune certain relation instances from the relationship graph. To do this, we assign priorities to relation instances, using a set of rules, then prune low-priority relationships if necessary. In decreasing order of priority, the rules rank the relation instances extracted from infoboxes first, followed by those from templates, then those from "See Also" sections (in article text), then reciprocated relation instances, then unreciprocated ones.

A *reciprocated* relation instance is one where there is also a reverse relationship from the target instance to the source instance. Intuitively, such relationships are stronger than unreciprocated ones. For example, Michelle Obama and Barack Obama have a reciprocated relationship because they mention each other in their pages. On the other hand, the Barack Obama page mentions Jakarta, but the reverse is not true. So the relationship between these two is unreciprocated and is not as strong.

3.4 Adding Metadata

At this point we have created a taxonomy of concepts, a DAG over the concepts, and a relationship graph over the instances. In the next step we enrich these artifacts with metadata. This step has rarely been discussed in the literature. We found that it is critical in allowing us to use our KB effectively for a variety of applications.

Adding Synonyms and Homonyms: Wikipedia contain many synonyms and homonyms for its pages. Synonyms are captured in Redirect pages. For example, the page for Sokrat redirects to the page for Socrates, indicating that Sokrat is a synonym for Socrates. Homonyms are captured in Disambiguation pages. For example, there is a Disambiguation page for Socrates, pointing to Socrates the philosopher, Socrates a Brazilian football player, Socrates a play, Socrates a movie, etc.

Table 1: Examples of non-Wikipedia sources that we have added

Name	Domain	No. of instances
Chrome	Automobile	100K
Adam	Health	100K
Music-Brainz	Music	17M
City DB	Cities	500K
Yahoo! Stocks	Stocks and companies	50K
Yahoo! Travel	Travel destinations	50K

We added all such synonyms and homonyms to our KB in a uniform way: for each synonym (e.g., Sokrat), we create a node in our graph then link it to the main node (e.g., Socrates) via an edge labeled "alias". For each disambiguation page (e.g., the one for Socrates), we create a node in the graph then link it to all possible interpretation nodes via edges labeled "homonym". Wikipedia typically designates one homonym interpretation as the default one. For example, the default meaning of Socrates is Socrates the philosopher. We capture this as well in one of the edges, as we found this very useful for our applications.

Adding Metadata per Node: For each node in our KB we assign an ID and a name, which is the title of the corresponding Wikipedia page (after some simple processing). Then we add multiple types of metadata to the node. These include

- *Web URLs:* A set of home pages obtained from Wikipedia and web search results. For a celebrity, for example, the corresponding Wikipedia page may list his or her homepages. We also perform a simple Web search to find additional homepages, if any.

- *Twitter ID:* For people, we obtain their Twitter ID from the corresponding Wikipedia page, from their home pages, or from a list of verified accounts (maintained by Twitter for a variety of people).

- *Co-occurring concepts and instances:* This is a set of other concepts and instances that frequently co-occur. This set is obtained by searching the large in-house Web corpus that we are maintaining.

- *Web signatures:* From the corresponding Wikipedia page and other related Web pages (e.g., those referred to by the Wikipedia page), we extract a vector of terms that are indicative of the current node. For instance, for Mel Gibson, we may extract terms such as "actor", "Oscar", and "Hollywood".

- *Social signatures:* This is a vector of terms that are mentioned in the Twittersphere in the past one hour and are indicative of the current node. For instance, for Mel Gibson, these terms may be "car", "crash", and "Maserati" (within a few hours after he crashed his car).

- *Wikipedia page traffic:* This tells us how many times the Wikipedia page for this node was visited in the last day, last week, last month, and so on.

- *Web DF:* This is a DF score (between 0 and 1) that indicates the frequency of the concept represented by this node being mentioned in Web pages.

- *Social media DF:* This score is similar to the Web DF, except it counts the frequency of being mentioned in social media in the near past.

3.5 Adding Other Data Sources

In the next step we add a set of other data sources to our KB, effectively integrating them with the data extracted from Wikipedia. Table 1 lists examples of data sources that we have added (by mid-2011). To add a source S, we proceed as follows.

First, we extract data from S, by extracting the data instances and the taxonomy (if any) over the instances. For example, from the "Chrome" data source we extract each car description to be an instance, and then extract the taxonomy T over these instances if such a taxonomy exists. For each instance, we extract a variety of attributes, including name, category (e.g., travel book, movie, etc.), URLs, keywords (i.e., a set of keywords that co-occur with this instance in S), relationships (i.e., set of relationships that this instance is involved in with other instances in S), and synonyms.

If the taxonomy T exists, then we match its categories (i.e., concepts) to those in our KB, using a state-of-the-art matcher, then clean and add such matches (e.g., Car = Automobile) to a concordance table. This table will be used later in our algorithm.

We now try to match and move the instances of S over to our KB. We handle the simplest cases first, then move on to more difficult ones. Specifically, for each instance (x, c) of S, where x refers to the instance itself, and c refers to the category of the instance:

- If there exists an instance (x', c') in our KB such that $x.name = x'.name$ and $c.name = c'.name$ (e.g., given a node (salt, movie) in S there may already exist a node (salt, movie) with identical instance name and category name in our KB), we say that (x, c) matches (x', c'). We simply add the metadata of (x, c) to that of (x', c'), then return.
- Otherwise, we find all instances (x', c') in our KB such that $x.name = x'.name$, but $c.name \neq c'.name$ (e.g., (salt, movie) vs. (salt, cinema) or (salt, movie) vs. (salt, condiment)). For each found (x', c'), we match it with the instance (x, c) from S, using all available attributes and meta data. If we find a positive match, then we have a situation such as (salt, movie) vs. (salt, cinema). In this case we simply add the metadata of (x, c) to that of (x', c'), then return.
- Otherwise, we have a difficult case. Here, we first use the concordance table created earlier to find a match $c = c'$, meaning that category c of S maps into category c' of our KB. If such a match exists, then we try to match the instance x with all the instances of c', again utilizing all available attributes and metadata. If x matches an instance x' of c', then we add the metadata of x to that of x', then return; otherwise, we create a new instance x' of c', and copy over into x' the metadata of x. If we cannot find a mapping for category c, then we alert the analyst. He or she will create a category for c in our KB, add x, and then update the concordance table accordingly.

4. MAINTAINING THE KNOWLEDGE BASE

After building the initial KB from Wikipedia and other data sources, we need to maintain it over time. Maintaining the KB means (1) updating it on a regular basis, as Wikipedia and data sources change, and (2) manually curating its content, to improve the accuracy and add even more content. We now elaborate on these two aspects.

4.1 Updating the Knowledge Base

There are two main ways to update a KB: (1) rerunning the KB construction pipeline from the scratch, or (2) performing incremental updates. Option 1 is conceptually simpler, but takes time, and raises the difficult problem of recycling human curations, as we discuss below.

Option 2, performing incremental updates, typically takes far less time (and thus can allow us to keep the KB more up to date). It is also easier to preserve human curations. For example, if an analyst has deleted an edge in the graph, then when performing incremental updates, we may simply decide not to override this analyst action.

But to execute Option 2, we may need to write many rules that specify how to change the KB given an update at a source. For example, if a new page or a new subtree is added to Wikipedia, or a new instance is deleted from a data source, what changes should we make to the KB? Writing such rules has proven to be difficult in our setting, for two reasons.

First, during the construction pipeline we execute several algorithms that examine the *entire* Wikipedia graph to make decisions. In particular, an algorithm examines the Wikipedia graph to create a taxonomic tree, and another algorithm examines the graph, together with the tree produced by the previous algorithm, to detect and break cycles, to create a DAG. It is unclear how to modify these "global" algorithms into making incremental updates, given a change in their input graph.

Second, we allow our analysts to write "integrity constraints" over the taxonomic tree and the DAG (see the next subsection). In general, when a construction pipeline utilizes a set of integrity constraints, it becomes much more difficult to figure out how to modify the pipeline into making incremental updates, given a change in the input.

For the above reasons, currently we use Option 1, rerunning the construction pipeline from the scratch, to update our KB. However, we do propagate certain simple updates in real time, those for which we are confident we can write reasonable update rules. Examples of these updates are adding a new instance having just one parent and adding a new synonym instance. Examining how to propagate more complex updates is an ongoing work.

As mentioned earlier, Option 1 raises two main challenges. First, we need to minimize the time it takes to execute the entire pipeline. We use parallel processing for this purpose. For example, to parse Wikipedia, we split it into NUMSPLITS equal-sized chunks and launch RUNSPLITS parallel threads, each thread working on NUMSPLITS/RUNSPLITS chunks on an average (we set RUNSPLITS=# of CPUs and NUMSPLITS=2*RUNSPLITS). Similarly, to create the relationship graph, we partition the relationship edges (where each edge is of the form <Source, Target, Relationship>) on the source nodes then process them in parallel threads. Using a single machine with 256G RAM, 0.8GHz processor, and 32 processors, it takes roughly 12.5 hours to complete the construction pipeline (see Section 6 for more statistics). So far this time has been acceptable for us. Should we need to process the entire pipeline faster, we will consider using a MapReduce infrastructure.

The second challenge of Option 1, rerunning from the scratch, is to preserve human curations of the previous iterations. We discuss this challenge in the next subsection.

4.2 Curating the Knowledge Base

Curating the KB is a critical part of our ongoing maintenance. For this purpose we employ a data analyst. The job of the analyst is to evaluate the quality of the KB and then curate it as much as possible.

Evaluating the Quality: Every week the analyst performs one or multiple manual evaluations. In each evaluation, the analyst performs two main steps. First, he or she takes a sample of the taxonomy, by randomly sampling a set of paths, each path goes all the way from the root to a leaf. The analyst then manually examines the path, to see if the nodes and edges on the path are correct. For example, if a path says that Socrates is an instance of concept "Ancient Greek Philosopher", which is a child of "Philosopher", which is a child of the root, then this path is judged to be correct. On the other hand, if the path says Socrates is an instance of "Ancient Greek Philosopher", which is a child of "Famous Suicide", which is a child of the root, then this path is judged not correct.

In the second step of the evaluation, the analyst also checks all nodes that have at least 200 children. For each such node, he or she checks to see if the node is assigned to the correct parent. Finally, if developers working on applications that use the KB find any quality problems, they alert the analyst to those as well.

Curating by Writing Commands: Based on the quality evaluation, the analyst can perform the following curating actions:

- *Adding/deleting nodes and edges:* We have seen that the root node and the verticals were added editorially. The analyst can add and delete other nodes and edges too by writing a set of commands and storing these commands in files. During the construction process, the files will be parsed and the add/delete actions will be executed on the Wikipedia graph (before the algorithm to create the taxonomic tree is run).

- *Changing edge weights:* The analyst can change any component in the weight vector associated with an edge if he or she deems that necessary. This will have a direct impact on the construction of taxonomy tree and DAG.

- *Changing the assignment of an instance-of or an is-a relationship:* Recall that we run an algorithm to create a taxonomic tree. Ideally, all edges of this tree should be describing is-a relationships, that is, true concept - subconcept relationships. But since the algorithm is automatic, it can make mistakes. For example, it may say that concept X is a kind of concept Y, whereas in practice, X is actually a kind of Z. In such cases, the analyst may write a command that specifies that X is actually a kind of Z. When the tree creation algorithm is run again (in the next update cycle), it will take this command into account.

  Similarly, we may have incorrect instance-of relationships in the graph, such as saying that a is an instance of X, whereas it is actually an instance of Y. In such cases, the analyst can write similar commands to correct the mistakes.

- *Recommending an ancestor to a node:* For example, generally all politicians go under the concept "Politics", but an analyst may write a command to recommend that the dead politicians should go under the concept "History".

  In general, a command that recommends that X should be an ancestor of Y will be processed as follows. If the recommended X is an immediate parent of Y, then we just assign a high recommendation weight to the edge $X \rightarrow Y$. Otherwise, we perform a path selection process to determine which path should be chosen from Y to X. We assign a high recommendation weight to all the edges in the chosen path. If there is no path from Y to X, then we find their least common ancestor (LCA). This LCA now becomes the recommended ancestor and a high recommendation weight is assigned to all the paths from Y to the LCA node.

- *Assigning preference to a subtree in the graph:* Suppose there is a node X with two paths leading to the root. Clearly, in the taxonomy we can select only one path. Suppose the analyst prefers one path to the other, because that path belongs to a subtree judged to be more appropriate, then the analyst may want to assign a higher preference to that subtree. For example, the analyst might want to give a higher preference to the subtree rooted at "Music by Artist" and lower preference to the subtree rooted at "Music by Genre". He or she does this by writing a command that adjusts the subtreepref weight in the weight vectors associated with the edges of the subtrees.

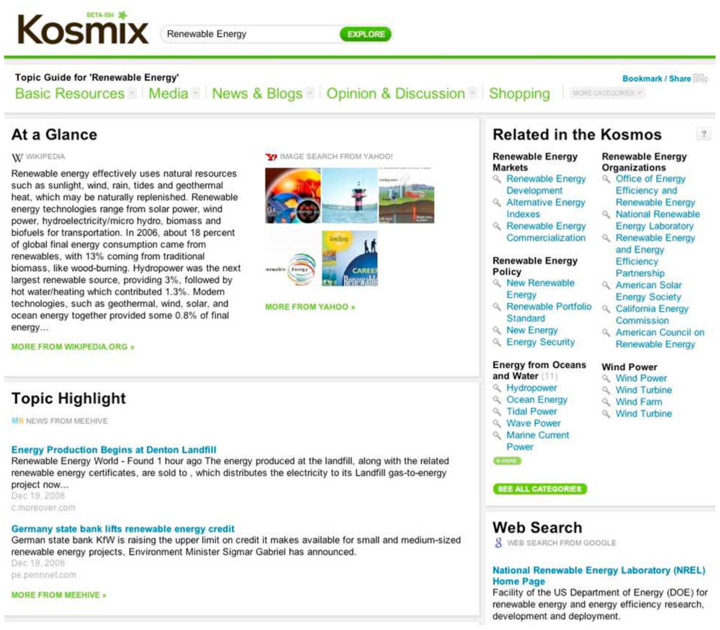
Managing Commands: As described, the analyst can perform a variety of curating actions. He or she encodes these actions as commands, using a special command language that we have developed. Whenever the KB construction pipeline is run (or rerun), it processes and takes into account these commands at various points during the construction process.

As described, the use of commands brings two benefits. First, using the command language, the analyst can write a simple command that affects hundreds or thousands of nodes in one shot. This is much better than if he or she modifies these nodes one by one. Second, using the commands allows us to recycle the human curation performed by the analyst across reruns.

At the moment our KB contains several thousands such human-curation commands, written by our analysts over 3-4 years. Thus, the analyst in charge is also responsible for maintaining these commands. Every single day, the commands will be evaluated and if any one of them is violated, the analyst will receive an alert, and can then decide how best to handle the violation. (For more details, please see the technical report at `pages.cs.wisc.edu/~anhai/papers/kcs-tr.pdf`.)

5. USING THE KNOWLEDGE BASE

We now briefly describe how we have used the KB for a variety of real-world applications at Kosmix and later at



Topic Guide for 'Renewable Energy'

Basic Resources | Media | News & Blogs | Opinion & Discussion | Shopping

At a Glance

WIKIPEDIA

Renewable energy effectively uses natural resources such as sunlight, wind, rain, tides and geothermal heat, which may be naturally replenished. Renewable energy technologies range from solar power, wind power, hydroelectricity/micro hydro, biomass and biofuels for transportation. In 2006, about 18 percent of global final energy consumption came from renewables, with 13% coming from traditional biomass, like wood-burning. Hydropower was the next largest renewable source, providing 3%, followed by hot water/heating which contributed 1.3%. Modern technologies, such as geothermal, wind, solar, and ocean energy together provided some 0.8% of final energy...

MORE FROM WIKIPEDIA.ORG »

IMAGE SEARCH FROM YAHOO!

MORE FROM YAHOO »

Related in the Kosmos

Renewable Energy Markets
- Renewable Energy Development
- Alternative Energy Indexes
- Renewable Energy Commercialization

Renewable Energy Policy
- New Renewable Energy
- Renewable Portfolio Standard
- New Energy
- Energy Security

Energy from Oceans and Water (11)
- Hydropower
- Ocean Energy
- Tidal Power
- Wave Power
- Marine Current Power

Renewable Energy Organizations
- Office of Energy Efficiency and Renewable Energy
- National Renewable Energy Laboratory
- Renewable Energy and Energy Efficiency Partnership
- American Solar Energy Society
- California Energy Commission
- American Council on Renewable Energy

Wind Power
- Wind Power
- Wind Turbine
- Wind Farm
- Wind Turbine

Topic Highlight

NEWS FROM MEEHIVE

Energy Production Begins at Denton Landfill
Renewable Energy World - Found 1 hour ago The energy produced at the landfill, along with the related renewable energy certificates, are sold to , which distributes the electricity to its Landfill gas-to-energy project now...
Dec 19, 2008
c.moreover.com

Germany state bank lifts renewable energy credit
German state bank KfW is raising the upper limit on credit it makes available for small and medium-sized renewable energy projects, Environment Minister Sigmar Gabriel has announced.
Dec 19, 2008
pe.pennnet.com

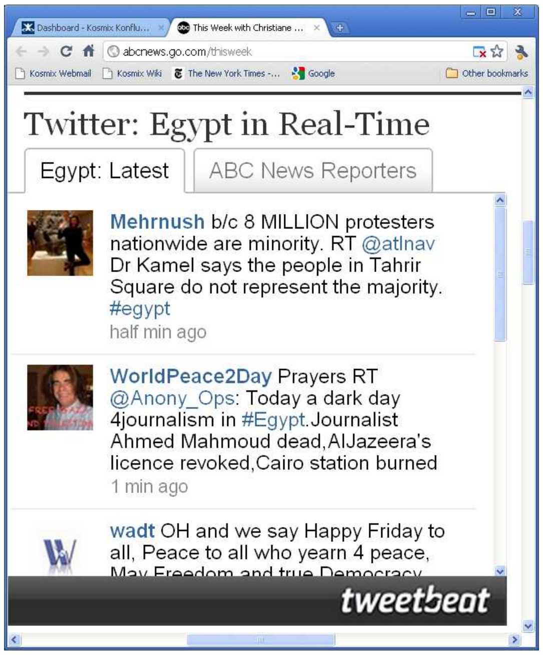
MORE FROM MEEHIVE »

Web Search

WEB SEARCH FROM GOOGLE

National Renewable Energy Laboratory (NREL) Home Page
Facility of the US Department of Energy (DOE) for renewable energy and energy efficiency research, development and deployment.

Figure 6: A result page for querying Deep Web using the keyword "Renewable Energy"



Twitter: Egypt in Real-Time

Mehrnush b/c 8 MILLION protesters nationwide are minority. RT @atlnav Dr Kamel says the people in Tahrir Square do not represent the majority. #egypt
half min ago

WorldPeace2Day Prayers RT @Anony_Ops: Today a dark day 4journalism in #Egypt.Journalist Ahmed Mahmoud dead,AlJazeera's licence revoked,Cairo station burned
1 min ago

wadt OH and we say Happy Friday to all, Peace to all who yearn 4 peace, May Freedom and true Democracy

Figure 7: Event monitoring in social media using Tweetbeat

WalmartLabs. These are query understanding, Deep Web search, in-context advertising, event monitoring in social media, product search, social gifting, and social mining. In Section 6 we discuss lessons learned from using the KB for these applications.

Understanding User Queries: Kosmix started out as a Deep Web search engine. A user poses a search query such as "Las Vegas vacation" on kosmix.com, we try to understand the query, go to an appropriate set of Deep Web data sources (i.e., those behind form interfaces), query the sources, combine then return the answers to the user.

Clearly, understanding the user query is a critical component in the above process. To understand a query, we attempt to detect if it mentions any concept or instances in the KB. For example, the query "Las Vegas vacation" mentions the instance "Las Vegas" and concept "vacation". We do this by performing information extraction from the user query. Specifically, we use the KB (as a dictionary) to identify strings such as "Las Vegas" and "vacation" as candidate concepts/instances . Next, we ascertain that these indeed refer to the concepts/instances in our KB (i.e., to resolve any potential homonym problems). Finally, we return the set of concepts/instances mentioned in the query as the understanding of that query.

Deep Web Search: As discussed above, Deep Web search means querying a set of appropriate Deep Web data sources. To do this, we first assemble a set of a few thousand Deep Web data sources, covering a broad range of topics. Next, we assign these sources to the appropriate nodes in our taxonomy. For example, TripAdvisor, a source that reviews vacation destinations, is assigned to the node "vacation".

Now given a user query Q, suppose in the query understanding step we have found that Q mentions concept nodes X and Y in our taxonomy. Then we query the Deep Web data sources at these nodes, combine the answers, then return those to the user. Figure 6 shows a sample answer page, in response to the query "renewable energy". Note that the answers to this query come from a variety of sources, including Wikipedia, meehive, etc. Note also that we show a set of related concepts on the upper righthand corner of the answer page. These concepts come from the neighborhood of concept "renewable energy" in our taxonomy. In practice the Deep Web search process is far more complex. But the above description should give an idea on how our KB is used in that process.

In-context Advertising: The basic idea here is that when a user is reading a Web page (e.g., a newsarticle or a discussion forum page), we parse the page, identify the most important concepts/instances on the page, and then highlight those concepts/instances. When the user hovers over a highlight, he or she will see a pop-up ad that is relevant to the highlighted concepts/instances. This is known as in-context advertising. We address this problem in a way similar to the way we try to understand a user query. That is, given a Web page, we perform information extraction and then entity disambiguation to determine if the page mentions any concepts and instances in our KB, and how relevant those concepts/instances are for the page.

Event Monitoring in Social Media: In late 2010 Kosmix turned its focus to social media. It built a flagship application that monitors the Twittersphere to detect interesting emerging events (e.g., Egyptian uprising, stock crash, Japanese earthquake), then displays all tweets of these events in real time. Figure 7 shows an example. This was a little widget embedded on the ABC news homepage, and powered by Kosmix. For the event "Egyptian uprising" (which was probably the hottest political event at that time), the widget shows interesting tweets related to that event, scrolling in real time.

To do this, for each incoming tweet we must decide whether it belongs to an event E. Again, we use our KB to perform information extraction and entity disambiguation, to detect all concepts/instances mentioned in the tweet. We then use this information to decide if the tweet belongs to the event.

Product Search: In mid 2011 Kosmix was acquired by Walmart and since then we have used our KB to assist a range of e-commerce applications. For example, when a user queries "Sony TV" on walmart.com, we may want to know all categories that are related to this query, such as "DVD", "Bluray players", etc. We use the KB to find such related categories. We also use a version of the KB that is greatly expanded with product data sources to interpret and understand user queries.

Social Gifting: In Spring 2012 we introduced a Facebook application called ShopyCat [18]. After a Facebook user installs ShopyCat and gives it access to his/her Facebook account, ShopyCat will crawl his posts as well as those of his friends, then infer the interests of each person. Next, ShopyCat uses these interests to recommend gifts that the user can buy for his or her friends. For example, ShopyCat may infer that a particular friend is very interested in football and Superbowl, and hence may recommend a Superbowl monopoly game from deluxegame.com as a gift.

ShopyCat infers the interests of a person by processing his or her posts in social media, to see what concepts/instances in our KB are frequently mentioned in these posts. For example, if a person often mentions coffee related products in his or her posts, then ShopyCat infers that the person is likely to be interested in coffee. Thus, our KB can be used to process a person's social-media activities to infer his or her interests (see [18] for more details).

Social Mining: In one of the latest applications, we use our KB to process all tweets that come from a specific location, to infer the overall interests of people in that location, then use this information to decide how to stock the local Walmart store. For example, from mining all tweets coming from Mountain View, California, we may infer that many people in Mountain View are interested in outdoor activities, and so the outdoor section at the local Walmart is expanded accordingly. Such social mining appears to be quite useful on a seasonal basis.

Fast and Scalable Access: To serve a broad range of applications, we have to store and serve our KB in a way that ensures fast and scalable access. We now briefly discuss these issues.

We store the taxonomy tree, the DAG, and the relationship graph on disk as several files of vertices and edges in a proprietary data format. We keep the metadata in separate files. The whole KB occupies around 30G. We load the tree and the DAG in memory as a Boost(C++) adjacency list. The metadata is also loaded in-memory in data structures that are essentially maps and hash tables. The in-memory data structures are around 25G in size.

Our KB APIs support only a fixed number of pre-defined functions. These are basically graph traversal functions such as getLineage, getParents, getChildren, getNumChildren, etc. We pre-compute JSON outputs of these functions for all the nodes and store these outputs in Cassandra, a distributed key-value store, configured over a cluster of machines. This helps us scale better against multiple concurrent clients as opposed to having one central taxonomy serve all requests. However, not all application queries can be answered from the pre-computed outputs stored in Cassandra, in which case the functions are evaluated on the fly.

6. TEAM ORGANIZATION, STATISTICS, AND LESSONS LEARNED

Team Organization: During the period 2010-2011 Kosmix had about 25-30 developers. Out of these a core team of 4 persons was in charge of the KB. A data analyst performed quality evaluation and curated the KB, as discussed earlier. A developer wrote code, developed new features, added new signals on the edges, and so on. A system person worked 50% of the time on crawling the data sources, and maintaining the in-house Wikipedia mirror and the Web corpus. An UI specialist worked 50% of the time on the look and feel of the various tools. Finally, a team lead designed, supervised, and coordinated the work. Occasionally we hired interns to help with quality evaluation of the KB.

Clearly, developing a different KB may require a different team size and composition. But the above team demonstrated that even a relatively small team can already build and maintain large KBs that are useful for a broad range of real-world applications.

At WalmartLabs we have significantly expanded the KB in several directions, and are building a variety of other KBs (e.g., a large product KB). So at the moment there are multiple teams working on a variety of KB projects, but most teams have no more than 4 person whose long-term job is to be in charge of the KB.

Statistics: Figure 8 shows several statistics about the KB. Overall, the KB has 13.2M concepts and instances, with a relationship graph of 165M edges. It is interesting, but not surprising, that mining the article texts produce 100M, the most of such relationships.

The size of the entire KB on disk and in memory is relatively small. This is important because we have found that our applications often want to replicate the KB and have their own KB copy, so that they can minimize access time and modify the KB to suit their own need. Having a relatively small footprint serves this purpose well. The construction time is still relatively high (12.5 hours), but that is partly because we have not fully optimized this time. Finally, the pipeline was run on just a single (albeit powerful) machine. All of these demonstrate the feasibility of building and using relatively large KBs with a relatively modest hardware and team requirements. This is important for anyone considering building such KBs for a particular domain.

Lessons Learned: We now discuss several lessons learned from our experience.

- *It is possible to build relatively large KBs with modest hardware and team requirement:* We have discussed this point above.

- *Human curation is important:* Using automatic algorithms, our taxonomy was only 70% accurate. Human curation in the form of the commands increased this rate to well above 90%. More importantly, our experience demonstrates that human curation that makes a difference is indeed possible even with a relatively small team size.

- *An imperfect KB is still useful for a variety of real-world applications:* If an application must show the

COMPOSITION	No. of concepts	6.5M
	No. of instances	6.7M
	No. of verticals	23
	No. of relationships	165M
	No. of relationships from infoboxes	15M
	No. of relationships from templates	50M
	No. of relationships from "See Also"	0.2M
	No. of relationships from text	100M
STORAGE	Size of DAG on disk	5.2G
	Size of taxonomic tree on disk	500M
	Size of metadata files on disk	10G
	Size of relationship graph on disk	14G
	Size of in-memory caches (data structures)	25G
PIPELINE	System configuration	256G RAM 0.8 GHz processor 32 processors
	Time to completion	12.5 hours
	Time to create Wikipedia graph	4.5 hours (NUMSPLITS = 64 RUNSPLITS = 32)
	Time to construct taxonomic tree and DAG	5 hours
	Time to add other data sources	1 hour
	Time to generate build reports	2 hours
SOME FACTS	Instance with maximum no. of descendants	Habeas corpus petitions of Guantanamo Bay detainees (1090)
	Instance with maximum number of homonyms	Workers' Party (581)
	Concept with maximum no. of leaf descendants	Media (498141)
	Concept with maximum no. of children in Wikipedia graph	Writers (98206)
	Concept with maximum no. of children in DAG	Writers (98189)

Figure 8: Statistics on the construction and composition of our KB

data from the KB to the end user, then it is important that the data should be near perfect, or the user should be aware of and does not mind the fact that the data may not be perfect. For example, if the user is an intelligence analyst, then he or she may be okay with imperfect data as long as he or she can still mine useful intelligence out of it. However, if the user is a Web search user, posing a query such as "France vacation", then he or she is less likely to be happy with imperfect structured data being returned about France.

In our case, most of our applications do not have to show the KB data to the end user. Rather, they use the KB to guide the process of finding such data (e.g., using the KB to find out which Deep Web data sources they should query). As such, even somewhat imperfect KBs can already serve quite useful guiding heuristics.

- *An imperfect graph of relationships is still useful for a variety of real-world applications:* This is a related point to the point above. While imperfect, the relationship graph still provides more contexts for the concepts/instances and show how they are related to one other. Our applications can exploit such information to help entity disambiguation and to find related items (e.g., in product search).

- *It is important to build ontology-like KBs:* We recognized the importance of this when we tried to understand a body of tweets, effectively wanting to build a source-specific KB from this body of tweets. Having an ontology-like KB proved immensely useful for two reasons. First, it allowed us to ground the notion of understanding. We can say that understanding the body of tweets means being able to interpret it with respect to the concepts, instances, and relationships we have in the KB. Second, it allowed us to use dictionary-based approaches to information extraction from the body of tweets. We found this approach to be reasonably accurate and fast. In contrast, many traditional information extraction approaches that rely on well-formed and grammatical text break down when applied to tweets.

- *Capturing contexts is critical for processing social media:* Recall from Section 3.4 that for each node in our KB we maintain as much context information as we can. Examples include Web signatures, social signatures, and Wikipedia traffic (see Section 3.4). It turned out that such context information is critical for processing social media. For example, given a tweet, "mel crashed his car", without knowing that in the past two hours, when people tweet about Mel Gibson, the most common words being mentioned are "crash" and "car", there is virtually no way for us to identify that "mel" in this tweet refers to Mel Gibson. As another example, given "go Giant!", without context we cannot tell if this is the New York Giant sport team or the San Francisco Giant team. Capturing such time-sensitive contexts however is a major challenge in terms of scaling and accuracy.

- *It is important to have clear and proven methodologies for building and maintaining KBs:* This point is further underscored now that at WalmartLabs we have multiple teams building multiple KBs. Turnover at the teams also means that it is important to have a methodology in place to help guide new members.

7. RELATED WORK

Cyc [14] and WordNet [16] are well-known early works on building global ontology-like KBs. As we mentioned in Section 3, Wikipedia is not a KB in the traditional sense, because its graph structure does not correspond to a taxonomy and its pages do not correspond neatly to concepts and instances.

Many recent works however have utilized Wikipedia (and other data sources) to semi-automatically build global ontology-like KBs. Well-known examples include Freebase [5, 6], DBpedia [1, 4] and YAGO [21, 22], and WolframAlpha. As far as we can tell, however, the end-to-end process of building these KBs has not been described in detail in the literature. In particular, the problem of converting the Wikipedia graph to a taxonomy has not been discussed, as we do in this paper. Further, little or no work has discussed maintaining, updating, and using KBs, even though these topics are becoming increasingly important (and often incur the largest costs of ownership in practice).

While relatively few global ontology-like KBs have been built (as discussed above), many domain-specific KBs have been developed and described, often together with applications that use the KBs. For example, IBM's Watson [13] uses a KB to answer questions. Other examples include Microsoft's EntityCube [25], which powers an academic search engine, Google Scholar [7], Rexa (rexa.info), DeepDive [17], and DBLife [11]. Again, here relatively little has been reported on the end-to-end process of building these KBs, and even less has been reported on maintaining and updating them over time. Further, even though it is important to have clear and proven methodologies for building and maintaining KBs, so far we have seen very little work in this direction (with [11] being an early attempt).

On other related topics, several works have described how to leverage user feedback in building KBs [5, 8, 10]. In particular, the work [10] describes how to enlist both automatic methods and a community of users to build KBs. Finally, any discussion of works on KBs would be incomplete without mentioning the Semantic Web [2]. The Semantic Web community has helped in coming up with W3C standards in formats (URIs [19], RDF [15], etc.) and protocols that need to be adhered to if one wants his or her data (such as the recently released Facebook graph data [24]) to link to a global KB called the linked data [3].

8. CONCLUSIONS

Over the past decades numerous KBs have been built and the KB topic has also received significant and growing attention. Despite this, however, relatively little has been published about how KBs are built, maintained, and used in the industry. Our paper hopes to contribute to filling this gap. In the paper we have described how we build, update, curate, and use a relatively large KB (combined from Wikipedia and other data sources) for a variety of real-world applications at Kosmix and WalmartLabs. We have also touched on topics that have received relatively little attention, such as team composition. Finally, we have discussed a set of lessons learned from our experience.

In the near future we plan to significantly expand our KB activities in many directions. We hope to expand our current KB with more traditional data sources. We have been building and will be expanding a large KB called Social Genome that cover interesting entities, relationships, and instances in social media. We have also been building a large KB of products and associated information. Finally, we are examining how to employ crowdsourcing techniques in our KB construction and maintenance process.

9. REFERENCES

[1] S. Auer, C. Bizer, G. Kobilarov, J. Lehmann, R. Cyganiak, and Z. Ives. DBpedia: A nucleus for a web of open data. In *The Semantic Web*, 2007.

[2] T. Berners-Lee, J. Hendler, O. Lassila, et al. The semantic web. *Scientific American*, 284(5):28–37, 2001.

[3] C. Bizer, T. Heath, and T. Berners-Lee. Linked data-the story so far. *IJSWIS*, 5(3):1–22, 2009.

[4] C. Bizer, J. Lehmann, G. Kobilarov, S. Auer, C. Becker, R. Cyganiak, and S. Hellmann. DBpedia- a crystallization point for the web of data. *Web Semantics: Science, Services and Agents on the World Wide Web*, 7(3):154–165, 2009.

[5] K. Bollacker, R. Cook, and P. Tufts. A platform for scalable, collaborative, structured information integration. In *IIWeb*, 2007.

[6] K. Bollacker, C. Evans, P. Paritosh, T. Sturge, and J. Taylor. Freebase: a collaboratively created graph database for structuring human knowledge. In *SIGMOD*, 2008.

[7] D. Butler. Science searches shift up a gear as google starts scholar engine. *Nature*, 432(7016):423–423, 2004.

[8] X. Chai, B. Vuong, A. Doan, and J. F. Naughton. Efficiently incorporating user feedback into information extraction and integration programs. In *SIGMOD*, 2009.

[9] Y. J. Chu and T. H. Liu. On the shortest arborescence of a directed graph. *Science Sinica*, 14(270):1396–1400, 1965.

[10] P. DeRose, X. Chai, B. Gao, W. Shen, A. Doan, P. Bohannon, and X. Zhu. Building community wikipedias: A machine-human partnership approach. In *ICDE*, 2008.

[11] P. DeRose, W. Shen, F. Chen, A. Doan, and R. Ramakrishnan. Building structured web community portals: A top-down, compositional, and incremental approach. In *VLDB*, 2007.

[12] J. Edmonds. Optimum branchings. *Journal of Research of the National Bureau of Standards B*, 71:233–240, 1967.

[13] D. Ferrucci, E. Brown, J. Chu-Carroll, J. Fan, D. Gondek, A. A. Kalyanpur, A. Lally, J. W. Murdock, E. Nyberg, J. Prager, et al. Building Watson: An overview of the DeepQA project. *AI magazine*, 31(3):59–79, 2010.

[14] D. B. Lenat. CYC: A large-scale investment in knowledge infrastructure. *Communications of the ACM*, 38(11):33–38, 1995.

[15] F. Manola, E. Miller, and B. McBride. RDF primer. *W3C recommendation*, 10:1–107, 2004.

[16] G. A. Miller. WordNet: a lexical database for English. *Communications of the ACM*, 38(11):39–41, 1995.

[17] F. Niu, C. Zhang, C. Ré, and J. Shavlik. DeepDive: Web-scale knowledge-base construction using statistical learning and inference. In *VLDS*, 2012.

[18] Y. Pavlidis, M. Mathihalli, I. Chakravarty, A. Batra, R. Benson, R. Raj, R. Yau, M. McKiernan, V. Harinarayan, and A. Rajaraman. Anatomy of a gift recommendation engine powered by social media. In *SIGMOD*, 2012.

[19] L. Sauermann, R. Cyganiak, and M. Völkel. Cool URIs for the semantic web. *W3 Interest Group Note. http://www.w3.org/TR/cooluris/*.

[20] A. Singhal. Introducing the Knowledge Graph: things, not strings. *Official Google Blog, May*, 2012.

[21] F. M. Suchanek, G. Kasneci, and G. Weikum. Yago: a core of semantic knowledge. In *WWW*, 2007.

[22] F. M. Suchanek, G. Kasneci, and G. Weikum. Yago: A large ontology from Wikipedia and WordNet. *Web Semantics: Science, Services and Agents on the World Wide Web*, 6(3):203–217, 2008.

[23] R. E. Tarjan. Finding optimum branchings. *Networks*, 7(1):25–35, 1977.

[24] J. Weaver and P. Tarjan. Facebook Linked Data via the Graph API. *Semantic Web*, 2012.

[25] J. Zhu, Z. Nie, X. Liu, B. Zhang, and J. R. Wen. StatSnowball: a statistical approach to extracting entity relationships. In *WWW*, 2009.