Anatomy of a Symbol Crawler (DRAFT 0.2)

Stefan Gower

Created 9/23/2015

Revised 9/28/2015

Purpose of this Document

There is much interest these days in the construction of knowledge graphs, and the biggest companies (e.g. Google) and energetic start-ups (DiffBot) - and probably many others - are working on this problem.

With considerable humility, here is my own take on the construction of a knowledge graph. The inspiration behind it is obvious: can you take the general ideas of a web crawler, where the things being crawled are URLs, and create a very different kind of crawler - a crawler that is crawling *symbols*.

A symbol is not a URL. A symbol may be a high-level concept, such as a tiger, in the sense of an animal, or tiger in the sense of a Major League Baseball team that plays in Detroit; alternatively, a symbol may be little more than a string, with uncertain meaning.

The output of a symbol crawler is a knowledge graph composed of a graph of symbols. Like a web crawler, the goal here is to have a symbol crawler that is both automated and scalable. That is, given access to appropriate data sources, such as the world wide web, a symbol crawler will just crawl away, incrementally creating (or maintaining) a knowledge graph.

A Super-Quick Review of a Web Crawler

Now, the operations of a web crawler are known well enough, but let's briefly review these operations. The description below is very basic. [1]

A web crawler starts by having some set of seed URLs inserted into a queue. This queue holds the URLs that need to be crawled. The web crawler then selects some URL, and fetches its content. The web crawler adds this content to its index, and also extracts any hyperlinks from the web page. The URL of the crawled web page is placed in a set of visited URLs, so this URL won't be needlessly visited again if that URL is encountered again later in the crawl. The URLs of these extracted hyperlinks are then examined. If a URL has not already been crawled, it can be placed in that queue of URLs - the queue of URLs that have not yet been visited . Processing continues while there are URLs in the queue of unvisited urls, or until some other limit has been reached, such as some duration of time (e.g. a week) is reached. The result of this web crawling

1 Take a look at http://nlp.stanford.edu/IR-book/pdf/irbookonlinereading.pdf.

is - amongst other things - a graph of some portion of web, with URLs linked to other URLs through hyperlinks. Of course, other processing, such as citation link analysis can then make it easier to search this graph.

That completes this basic explanation of a web crawler.

Symbol Crawler Operations

Now a symbol crawler has some things in common with a web crawler, but there are also some very sizable differences.

A symbol crawler does not have a queue of URLs. Instead, it has a queue of symbols that have not been visited, and it has a set of symbols that have been visited.

What is a symbol? This will take a little while to explain. I will begin simply and then gradually elaborate the meaning of a symbol.

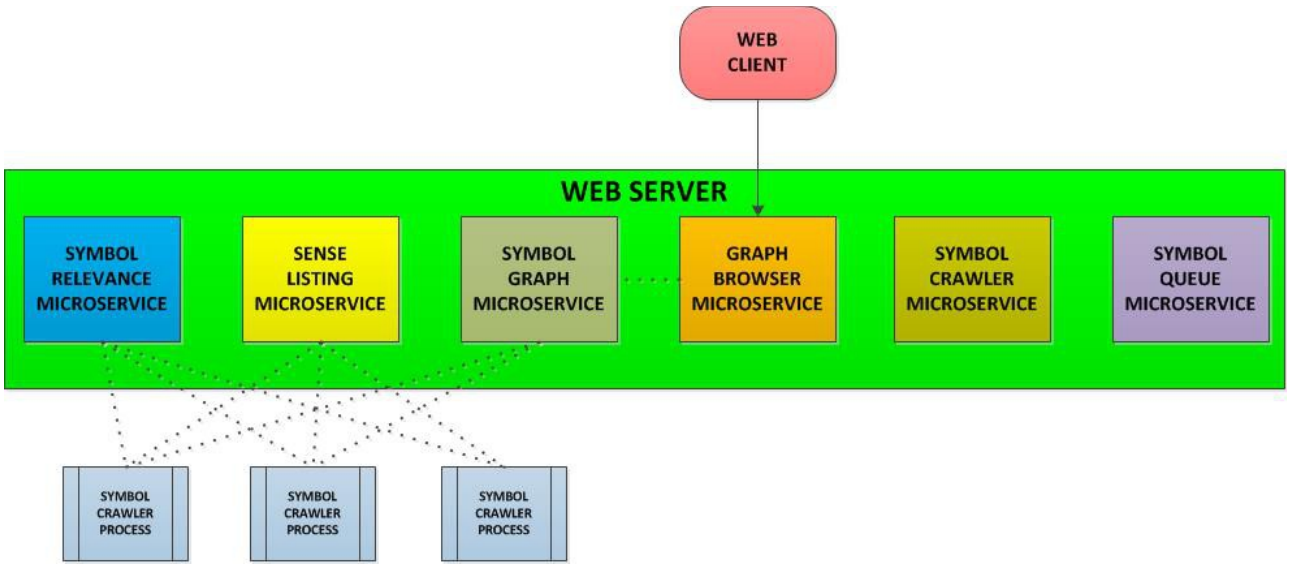
In the context of a system where stemming is employed, a symbol is a stemmed sequence of words. That is, "good information" becomes, after stemming, "good_inform". Similarly, "managed networks" becomes, after stemming, "managed_network". To make symbols easily recognizable, they are preceded by the pound sign. So we have #good_inform and #manage_network and so on. In Topic Scout, each string in the set of strings that reduced to the same symbol is called a *provenance string*.

A symbol may be associated with its *provenance strings*, where the symbol was derived from each of the set of strings. Hence the symbol #good_inform could be associated with the provenance string of "good information". There might even be snippets of text where the symbol was used such as: "The web site on real-time data mining had good information".

Each symbol is either a *surface symbol* or a *sense symbol*. A sense symbol includes the sense of the symbol. For example the symbol #tiger has no sense and is just a surface symbol. On the other hand, the symbol #tiger (animal) denotes a tiger in the sense of an animal; similarly, the symbol #tiger (tank) might denote a tiger in the sense of a German tank used in world war II. The sense symbol #tiger (butterfly) might denote a tiger in the sense of a Tiger butterfly.

Also note that DBPedia refers to a surface form for a word. Our naming is consistent with that of DBPedia, and has the same intent.

Here is an architectural view of a symbol crawler. I think of it as a minimally viable architecture. Other micro-services can be added to this architecture, but the diagram below is minimal.



The symbol queue micro-service is a queue of symbols, and its purpose is functionally identical to that of a queue of URLs for a web crawler; the key difference here is that the items in the symbol queue are symbols instead of URLs.

Like a web crawler, a graph is produced but here, rather than a graph of URLs, there is a symbol graph. The symbol graph is internally represented as a service - the symbol graph service. Such a symbol graph service might be implemented by any number of ways, including Neo4J, Titan and many others.[2]

Before the symbol crawler is started for an initial crawl, one or more symbols are placed in the queue of symbols that have not been visited yet. The symbols are just added via the symbol graph micro-service.

As will be described later, the queue of symbols can be ordered in many different ways. For right now, we will simply assume that it is FIFO queue. Later, I will describe various ways to rank the symbols in symbol queue, and so control the crawl ersin different ways.

The symbol graph is initially empty. While there might be many types of edges, as in DBPedia etc., to keep the exposition simple here, I will focus on a very small number of edge types between the symbols of the symbol graph.

What kind of symbols might be added into the symbol queue to seed the crawl? Here is one example. The surface symbol #tiger could be inserted into the queue of symbols to be visited. That is, this is a symbol for tiger, but with no specific sense. Let us suppose that only this single symbol is added via the symbol queue micro-service. Now a crawl is started.

The crawl itself is initiated via the symbol crawler service, such as an operation of **startCrawl**. This operation may also specify the number of crawlers that should be started, and their distribution in a cluster.

To keep things simple, let us suppose that only a single crawler is used.

When the symbol crawler starts up, the symbol crawler attempts to pop a symbol from the symbol queue service. In this case, the attempt succeeds. The #tiger symbol from the symbol

2 It, however, is probably best to pick a graph store that is not hobbled by ACID transactions. Such reliance on ACID transactions might limit both the performance and scale of the symbol crawler.

queue is returned.
Once the symbol crawler actually has a symbol, it must decide what to do with it. What it decides to do depends greatly on what kind of symbol it.
If a surface symbol has been obtained, the symbol crawler must call on *a sense listing service* to determine the senses of the surface symbol. Why? There is another micro-service - the symbol relevance micro-service - and its job is to take a symbol, and return symbols related to that symbol. There is, however, a requirement. The symbol relevance service needs to have some sense of a symbol. A surface symbol has no sense, and so the symbol crawler will now use the sense listing service to get the sense(s) of a symbol.
What does a sense listing service do?
The input of the sense listing service is a symbol; its output is a set of sense symbols. Note, at this point, it is not important how this sense listing service works, but merely that it does. Later on, we can consider certain implementations of such a sense listing service. Right now, however, it is best to just concentrate on the inputs and outputs of this sense listing service. Also note that if a sense symbol is provided to sense listing service, the sense listing service will also return a set of sense symbols. That set of sense symbols will contain the sense symbol that was provided as an input, as well as possible other sense symbols that related to the inputted sense symbol.
Let's consider both cases.
For example, if the input to the sense listing service is #tiger, such a sense listing service might return, as its outputs, the following set of sense symbols.

- #tiger (animal)
- #tiger (tank)
- #tiger (butterfly)

Alternatively, a specific sense of tiger must be provided. For example, if the input to the sense listing service is #tiger (animal) , such a sense listing service might return, as its outputs, the following set of sense symbols.

- #tiger (animal)
- #tiger (tank)
- #tiger (butterfly)

Note: in practice, the surface symbol #tiger has far more senses than the three senses for #tiger listed above.

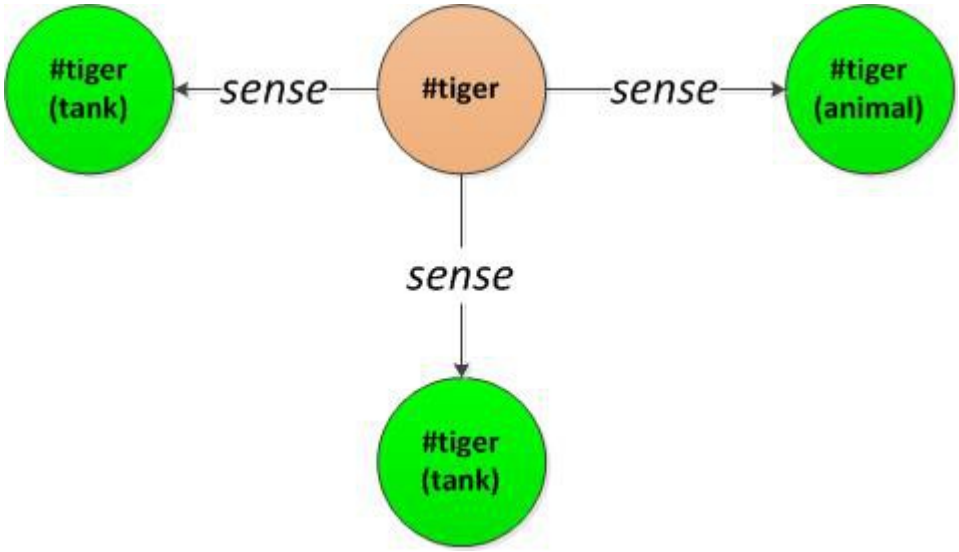
Also note that there may be cases where a sense listing service knows of no sense for a symbol. This case should not be ignored! Consider the wrong way to handle this. If we assume that all symbols relate to things with known senses, we could limit the symbol crawl. On other hand, if a symbol has no clear sense to it, we really need to make it explicit that this is a different kind of symbol. It is a kind of known unknown.

For example, suppose the surface symbol #nox88 is provided as an input into the sense listing

service, but the sense listing service has no idea whatsoever what a "nox88" is.

In such cases, the sense listing service may respond with a * sense, as in #nox88 (*). Such a sense symbol is just a polite way of saying that the sense listing service has no knowledge of this surface symbol and its senses. It is symbol with an unknown sense. But understand just because a surface symbol does not have a sense today, that is not to say it won't in the future. As I will shortly show, a surface symbol will always be related to its set of sense symbols. This set of sense symbols will never be empty. At the very least, if there are no other sense symbols in this set of sense symbols, there will be a * sense of the symbol.

After its sense symbols are obtained from the sense listing service, the surface symbol is placed in a graph, and the sense symbols generated by the sense listing service are placed in the graph as well. The surface symbol is linked to each of the sense symbols through a edge of type *sense*. One end of the edge has the role of *surface*; the other, *sense*.



The sense symbols of #tiger are now added into the symbol queue. This occurs for all sense symbols that have not yet been crawled. In this case, since the symbol has just started up, none of these sense symbols have been crawled, and so all of these sense symbols for tiger will be added using to the symbol queue using the symbol queue micro-service.
Once this is done, the symbol crawler proceeds. Again, it looks for a symbol by requesting a symbol from the symbol queue micro-service.
In the previous case, the symbol removed from the queue was a surface symbol; however, if the symbol removed from the queue is a sense symbol, the symbol crawler handles that sense symbol in a very different way.
A sense symbol is handled as follows. The sense symbol is passed as input to the symbol relevance micro-service. It is the job of the symbol relevance micro-service, when it is given a symbol, to determine the set of symbols most relevant to that symbol, and return this set of symbols as its outputs. The set of symbols returned may include symbols previously encountered by the symbol crawler, or may include previously unencountered symbols. The set of symbol returned may all be sense symbols; the set of symbol returned may all be surface symbols; the set of symbols returned may be a mix of both surface symbols and sense symbols.

Of course the symbol relevance service could be provided by one powered by Topic Scout's relevance engine; however, such a symbol relevance service could be powered by some other symbol relevance service, or even some mashup of symbol relevance services that results in a higher-level symbol relevance micro-service. Or a high school student interested in software programming might construct her own symbol relevance micro-service using deep learning or word2vec. This symbol crawler architecture will accept any symbol relevance service that meets its functional requirements. It is up to the creator of a symbol relevance service to provide the best possible output of relevant symbols, and the criteria that decides what is "best" is flexible as well. It may not always be the case that one symbol relevance service is better than another. Two implementations of a symbol relevance micro-service may simply have different goals and emphasis.

In addition, each symbol produced as the output of the symbol relevance micro-service may be associated with other data, such as a numeric value indicating the ranked importance of the symbol. A symbol with a greater weight is considered to be more semantically important the input symbol than another symbol. For example, the symbol #data_mining (data) might be related to two surface symbols #regress_tree and #optim_data, but the weight of #regress_tree is greater than the weight of #optim_data. The greater weight of #regress_tree means that #optim_data is semantically closer to #data_mining (data) than #optim_data.

Again, the symbol relevance service is - architecturally speaking - an abstract service. It is not important how this relevance service works, but merely that it does. Of course, the better it works, and the higher its quality, in respect to the symbols it produces, the better, but even a rather low-quality relevance service might be used in this symbol crawling architecture, and produce some sort of symbol graph.

I want to tie the function of the symbol relevance micro-service to modern web crawlers. There is a connection. The symbols produced by the symbol relevance service are analogous to the URLs abstracted by a web crawler when it extracts hyperlinks from a web page.

Compare the two operations below:

- Web crawler: extractURLs(URL) -> set of URLs
- Symbol crawler: getRelatedSymbols(symbol) -> set of symbols

It is easy to see the similarity between the functions of the two crawlers. Both crawlers have some way to produce information an input. In the case of a web crawler, a web page represented by an URL is used as input, and a set of URLs is produced as output. In the case of a symbol crawler, a sense symbol is provided to the symbol relevance service and, as its output, it produces a set of symbols. Functionally, the two kinds of crawlers are quite similar.

Of course, there is a difference. For a web crawler, the links from a web page are explicit things that may be found in the web page by looking for hyperlinks in a HTML page. A symbol relevance service doesn't have quite such an easy job. It is up to those who implement a symbol relevance service to figure out a way to produce symbols related to the input symbol. How they do this is, again, up to them. They may draw on additional data sources from the world wide web. They might use their own proprietary data sources, or ... whatever.

For each symbol in the set of symbols produced by the symbol relevance symbol, the following occurs: if the symbol has already been crawled, it is discarded. Otherwise, the symbol is added

to the queue of symbols that need to be visited. An edge of type link is created between the sense symbol used to generate these symbols, and each generated symbol. Each link edge is assigned a weight per the symbol generated by the symbol relevance service.

Hence we have a symbol graph with two edge types - *sense* and *link* - and a link edge has a weight. Such a weighted link just indicates the relevance of the object symbol to the subject symbol.

In its minimal architectural form, this is how the symbol crawler works. In this sense - no pun intended - the symbol crawler is quite simple, and one would expect it to be so, as a web crawler really is quite simple in its basic operations too, despite the fact that a production web crawler can turn into something very sophisticated indeed!

Of course, a crawl can involve just a single symbol crawler process, or a large set of symbol crawler processes, and these symbol crawler processes might be on a single machine, in a cluster, or be running on virtual machines in the cloud.

There are a few mechanical nuances that I anticipate.

It should be noted that before a sense symbol is inserted into the graph, its related surface symbol needs to be present in the graph. This is true even when inserting a * sense symbol into the symbol graph.

How does a crawl stop?

Note that there many conditions by which a crawl of a symbol crawler could be halted. It could be halted by a specific action, such as an admin executing a command to stop the crawl; it could be stopped because a certain time limit has been reached; the crawl could be halted because of resource limits. There could be many other ways (and reasons) to halt a crawl. An implementation of this symbol crawler architecture might want to provide a messaging micro-service, or something akin to Zookeeper (or Zookeeper itself) to manage starting, maintaining and ending a crawl.

Resilience is important for web crawlers, and resilience is similarly important for a symbol crawler. A crawl could be interrupted by a power failure, or partially interrupted when a server crashes. There is no problem here. As long as the appropriate data structures are persisted, there is no problem in resuming a symbol crawl. As long as the queue of symbols is managed and persisted properly, resumption from failure is fairly straightforward.

The only nuance in an implementation is to avoid windows of opportunity where some symbol could be processed, but the symbol lost because of some failure, and not returned to the symbol queue. Such lost of symbols should be carefully avoided. Again, similar problems can also face web crawlers, and the solutions there should also apply to a symbol crawler.

Sense Listing Service

A sense listing service may be implemented in many ways, or it can be implemented to combine the results of various sense listing services.

I will describe a very simple - almost trivial - approach. A simple approach is to simply use DBPedia to list all the senses of some word or string. For example, consider this URL.

- lookup.dbpedia.org/api/search.asmx/KeywordSearch?QueryString=tiger

This query to DBPedia will return the senses of the string "tiger".

Note, however, that the results are as good (or bad) depending on the data in Wikipedia. Wikipedia can be very good, and provide quite a complete list of senses. Other times, it can glaringly incomplete. Results are good but also inconsistent.

Another approach is to use WordNet from Priceton university. There are a lot of tools for using WordNet, and they won't be described here.

Another approach is to combine the senses produced by DBPedia and Wordnet. Such a combination is plausible, but may present technical challenges, as both WordNet and DBPedia may produce senses that are really the same, but differ slightly, and so are not merged when they really represent the same sense.

Note that the approaches above rely on curated data sources. Yes, both data sources are very, very rich indeed. That is without question. There will be cases, however, when a sense is missing from these data sources, even when it is a rather important one. These curated data sources are definitely not complete. For example, the meaning of "firestorm", as found in DBPedia in late September 2015 missed some major meanings found easily on the world wide web, related to both guns and arrows.

What to do about things that are found in curated data sources? The answer lies in word sense induction. As its name implies, word sense induction attempts to learn the various meaning of a word (or phrase) by some automated means, such as looking for meanings in various texts, and then inferring the meanings of that word from those examples.

Unfortunately, word sense induction is rather an open problem these days. Fortunately, a sense listing service can abstract exactly how senses are obtained, and may use only curated data sources, or word sense induction, or a combination of the two.

Hence some implementations of a symbol crawler may look for ways to discover senses that lie outside these curated data sources, using some form of word sense induction.

For example, an open source service such as carrot2.org might be used to discover clusters, and then an attempt could be made to find significant clusters that do not match up with any known sense from a curated data source.[3] Again, as pointed out previously, it may be technically

3 Or even from senses that were discovered from previous data mining.

challenging to determine when two senses, from a different source, are actually the same sense, or vice-versa; however, an implementation that can handle this problem will be rewarded with a more dynamic production of senses, rather than one that is strictly reliant on curated data sources.

Crowd sourcing could be used to uncover missing senses. For example, humans could help, through their collective feedback, in discovering new senses, deleting unneeded senses, or combining existing senses from different sources. Think mechanical Turk or something like this.

And there are, of course, other ways to search for senses. This is a good place to innovate.

But there is some good news here, at least architecturally, This architecture allow innovation in this important area to proceed at its own pace. Within a symbol crawler architecture such as this, a poorer sense listing service can easily be replaced by a better one, and the symbol crawler system will happily continue working, but just work better.

Topics versus Symbols

One of the problems in constructing a graph of symbols is to know when a symbol relates to something in such a way that a normal human being would recognize it as a topic. I am not going to get sucked into the rabbit hole of what is or is not a topic. I will, rightly or wrongly, keep things very simple.

Any sense symbol that is not a * sense is a *topic*. Symbols that are not topics will likely outnumber symbols that are topics, and by a very wide margin.

For example, a symbol relevance service might produce large numbers of symbols, where only a small fraction of these symbols will be associated with any real sense symbol (not a * symbol).

Consider the output of Topic Scout. Today, it produces 1- or 2-word combinations. It might also produce larger combinations, such 3-word and 4-word combinations. Now suppose that DBPedia was used as the source of senses. Well, symbols such as #swivelling_hips (from the Elvis Presley lexicon) probably doesn't correspond to a topic in DBPedia, and hence that surface symbol would probably end up just being associated with a * symbol, and nothing more.

On the other hand, there are many ways that symbol senses might be found, including but not limited to looking at queries posed to web search engines, or looking at twitter data.

In any case, these * symbols could outnumber other sense symbols very significantly. Based on my experience with Topic Scout, this seems a pretty safe assumption.

Ordering the Symbols in the Symbol Queue

As before, a lot of the inspiration for a symbol crawler comes from web crawlers. Now let's consider how a web crawl can be controlled.

For a web crawler, there is a set of URLs that need to be crawled, and there may be limited resources to do the crawling. So a lot of attention can go into ordering the URLs in the queue of URLs to be crawled so that the crawl is effective as possible.

In short, some URLs can be more interesting than others. Perhaps a URL is more interesting if it comes from a popular web site. Or perhaps it is more important because the topic of the web page is of interest (e.g. finance or sports).

Similarly, a symbol crawler needs to order its queue of URLs. Here are some of the ways that a symbol queue - the queue of symbols awaiting crawling - might be ordered. This is not an exhaustive list, and a software engineer could devise other ways of ordering these symbols, in addition to the methods described here.

These ways of ordering the elements of a queue are not just limited to ordering, but also include filtering out symbols from the symbol queue altogether.

- Strict FIFO. First in, first out.
- * Symbols are ignored.
- Symbols that relate to entities, such as by using entity recognition.
- A symbol that has many incoming links to is ranked before one with less links.
- Symbols with smaller number of component strings are ranked before those with more; that is #building comes before either #building_site or #site.
- A symbol, if used as a query to a web search service, such as Yahoo BOSS, will produce so many hits. Symbols with more hits can be ranked before symbols with fewer hits.
- A symbol that is trending on Twitter, or some other data source, can be ranked above a symbol that is not trending on Twitter.
- Symbols can be classified by domain. Here I do not mean domain as in an internet domain. Rather, I mean domain in the sense of a broad subject, such as finance or security or shopping. That is, if one wants to do a crawl based on the domain of finance, symbols strongly related to the domain of finance could be ranked above those that are not related to the domain of finance, or have a weaker relationship.
- Some combination of the above.

The methods above are not exhaustive, and an engineer could add to these methods, or modify the methods above, or further elaborate the methods above, or combine them in different ways. The point here is that ideas applied to web crawlers can also be applied to symbol crawler. The actual application of these ideas may differ strongly in their particulars, but functionally the ideas are much the same.

These various methods can be combined, and an overall ranking score can result. A symbol with a ranking score higher than the ranking score of another symbol is placed before that symbol in the symbol.

Of course, the computational costs of such ordering must also be considered. Nothing is without a cost.

Pretty Good is Good Enough to Start

There will be lots of little and big challenges relating to implementing a symbol crawler. This is true even for a non-production symbol crawler - a crawler implemented to show feasibility and little more. But even a simple version of such a symbol crawler could produce enough data that, after review and feedback, could help suggest improvements that would move a 0.1 version onwards to something better.

Resuming a Crawl

An implementation should produce a crawler that can be halted and then resumed. If the crawler is distributed, this also should apply to any server in a cluster. That is, a production symbol crawler with 30 servers should continue to work properly if one server crashes. The other servers should simply pick the work left undone by that crashed server.

A Scalable Symbol Crawler

Like a scalable web crawler, a symbol crawler can also be made to scale. Some of the details may vary. That is, a topic crawler is not a web crawler. But the principles are basically the same.

Such a symbol crawler should also be monitorable, and have appropriate means to administer concurrent crawls.

Advanced Feature: Projecting Supertopics and Subtopics on the symbol graph

A symbol crawler will produce a symbol graph. Now today Topic Scout is very hierarchical in nature, and going to a symbol graph throws away these hierarchies. Is there any way to discover hierarchies in the graph, so that hierarchies, or even lattices, form naturally, based on data mining?

One would think so, and such relationships might once again be obtained through both use of curated data sources as well as dynamic data mining.

Topic Scout, for example, knows how to measure how likely one topic is a subtopic of another, and it does this in a novel spin on information theory. So it is conceivable that Topic Scout could flexibly impose a topic hierarchy on a graph, and then adjust this topic hierarchy as the underlying symbol graph changes. That is, the topic hierarchy (or lattice) would "float" on top the underlying symbol graph, and this topic hierarchy (or lattice) would not be fixed. It would adapt itself to the symbol graph as that symbol graph changed.

Crowd sourcing might be another way to flexibly impose a topic hierarchy on a very large symbol graph. But given the number of topics involved, such topic hierarchies might not fully cover the number of topics within the symbol graph. Still, crowd sourcing could be a focused solution.

Data Mining and the Symbol Graph

There are lots of interesting ways to mine patterns from a symbol graph. I have built some related graphs from Topic Scout data, and it is quite clear, as a symbol graph grows large, there should be interesting ways to mine information from it using data mining.

There could be lots of pattern matching opportunities in a symbol graph.

Location Symbols

Symbols related to location are rather special and might require special treatment in some implementations of a symbol crawler. Location might even warrant its own distinct edge type.

Time Dimension

Symbols can have a temporal dimension. Certainly, the symbols related to a symbol can change over time, as the content concerning a symbol can change. Just consider the difference in symbols for #Barack_Obama (person) during a re-election campaign, versus governing during other times. There will be major differences in the symbols over time.

While it is more ambitious, an ideal symbol graph will have a temporal dimension, and so a sense symbol such as #Patriots (NFL) might have distinct temporal versions, and the symbols surrounding each temporal instance might vary considerably, from week to week, during the Super bowl, or when footballs have been mysteriously under-inflated.

A temporal graph would have some sizable advantages over one without a temporal dimension. Besides the ability to query across time, there is another important reason: *topic smoothing*. Some topics, such as sports or business or politics, have symbols that are changes quite a lot depending on the news item of the day. One way to discover the stable symbols of such topics is take a set of time-specific topics and then integrate these symbols into a more "timeless" topic. Of course, no topic is truly timeless, but clearly if certain symbols of a topic appear throughout many time-specific manifestations of a topic, that symbol is *perennial*. A *perennial topic* can then be formed from such a set of *perennial symbols*.

A symbol graph as a massive web-scale size classifier

Topic Scout has already demonstrated a strong ability to take ordinary web pages from the world wide web and determine the topic of that web page. Topic Scout achieved nearly 90% accuracy on hundreds of thousands of web pages with over 4,000 topics to choose from.

The symbol graph described is also creating topics and lexicons, but doing it through symbol

crawling. Consider the fact that a sense symbol has links to other symbols, and these links are weighted. These links form a topic lexicon for a sense symbol. That is, a sense symbol is a topic, and each such topic has a set of weighted symbols that form the topic's lexicon.

Given such a set of topics and lexicons, logically extracted from the symbol graph, these topics and their lexicons can be tuned and the tuned lexicons can be placed in a runtime classifier. Topic Scout already does this today. This could also be done for a large symbol graph where there could be millions of topic lexicons within that symbol graph.

Let's be honest here.

The goal here is ambitious: a classifier that is not exactly universal, but has so many topics in it, that it wouldn't be shameful to call it a universal classifier or, at least, a near-universal classifier.

Of course, approaches to classification different from Topic Scout could be used. Alternatively, a different kind of classifier might be built, based on activation spreading, and achieve similar goals. This is speculation, but reasonable speculation. That is, one could light up the symbol graph by consuming a stream of symbols produced from a document, or other source, such as a conversation. The activated symbol graph could then highlight the most important topics.

Confidence

Most of the big-time work on knowledge graphs has focused extensively on confidence levels. That is, how confidence is the system in the assertions being made? Of course, this is extremely important.

I have done zero work in this area, and can do little than acknowledge the importance of assigning confidence levels to portions of the symbol graph.

Sentiment Analysis

It is known that the words and phrases related to sentiment analysis often vary based on the topic. That is, the adjective "big" may be considered positive in combination to a house, when one is looking to rent or buy a house, but "big" might be negative in respect to having a "big ego".

Hence it may be good to allow any sense symbol to be related to a sentiment symbol where this sentiment symbol will capture what is relevant to sentiment analysis for that sense symbol.

Linking to Images

Symbols can be linked to images. If the symbol is related to a resource in DBPedia, the symbol might linked back to an image on Wikipedia. Similarly, the symbol might be linked to a thumbnail image.

For symbols that are not linked to images through a curated data source, there are other

alternatives. A search engine, such as Yahoo BOSS, can be used to find an image related to the symbol. Now a user interface may allow users to navigate a semantic graph where the semantic graph is largely composed (from a UI point of view) of images, and/or videos. Think Instagram® meets knowledge graph meetings web browsing meets Youtube®. Legal disclaimer: both Instagram and Youtube are registered trademarks of their respective owners.

In this way, it is possible to associate many symbols in the symbol graph with an image. Such associations open up the possibility of searching the graph, where the visual presentation of the graph is by images and by other media.

Porn Filters

Port filters should be applied to the symbol graph, particularly if access based on media (see above) is provided, so that underage users have a safe browsing experience; however, other users might also want to impose porn filters when accessing the symbol graph.

Citation Link Analysis Revisited

A symbol graph, as described here, can be further modified in ways that echo ideas in citation link analysis, but with some substantial differences.

In citation link analysis, which is a kind of authority-based popularity content, an URL is more important when it is the target of hyperlinks from web pages that are themselves authoritative. Yes, this is a circular definition, but implementations of citation link analysis deal with the circularity.

Now the importance of links may be turned on its head in a symbol graph. After all, there may be some questionable symbols that crop up in some graphs, such as symbols like #click_link or #privacy_policy or #add_cart etc. I have seen a *lot* of such symbols based on my experiences on implementing Topic Scout. I call these *noise symbols*.[4]

One could argue that a good symbol relevance service would not produce such noise symbols, but the web is a noisy place, and symbol relevance services may not be perfect in distinguishing relevant symbols from noise symbols.

So a symbol like #add_cart might have huge numbers of incoming links to it. Does this make #add_cart a super important symbol. Probably the opposite. That is, in a symbol graph, noise symbols may be identified simply because they have so many incoming links. and so if one is trying to rank the symbols of the symbol graph, the principles that drive the ranking here might be very different. I probably shouldn't risk speculating as to what these guiding heuristics will be. More data and experience is needed, but it seems to safe to say that the measures that would drive ranking symbols in a symbol graph might be markedly different than those of citation link analysis as applied to a graph of URLs produced by a web crawler.

4 A more colorful name is *crap symbols*.

It seems, however, a reasonable assumption that a symbol with huge numbers of incoming links is probably not the most symbol. Time will tell if this assumption is true or not.

Inside the Enterprise

This document has described a micro-service called the relevant symbol service. It is worth noting that some implementations of this micro-service might depend on web resources. That is, an implementation of a symbol relevance service might have dependencies on web sources.

On the other hand, other implementations of a symbol relevance micro-service might not have this dependency. An implementation by Topic Scout does not have this dependency.

This means that Topic Scout can find the relevant symbols of a topic even within an enterprise. This enterprise might be commercial, governmental or non-profit, but whatever the status of the enterprise, Topic Scout can find, for a sense symbol, the relevant symbols within that enterprise.

This means that it is possible to create private graphs - within the enterprise - that are connected with the more public symbol graph.

That is, there might not be one symbol graph, but a set of connected symbol graphs, some graphs being private.

Specialization

The previous section introduced the idea of connected symbol graphs, some being private. It is also possible that there could be specialized symbol graphs run by certain organizations, producing symbol graphs for their specialized domain, such as medicine, software, military, finance, etc.

It might then be possible for less specialized symbol graphs to link to such specialized symbol graphs.

Such specialization might be especially helpful in respect to symbol graphs associated with text classifiers. That is, a general-purpose text classifier, when it finds documents about a very specific subject, such as toxoplasmosis, might delegate this text to a specialized text classifier build upon a symbol graph targeting that domain (e.g. medicine).

Symbol Graph Maintenance

Once a symbol graph is in place, there is the question of maintaining the symbol graph. Like a web graph produced by a web crawler, there are lot of ways to schedule maintenance. The ways to schedule maintenance include, but are not limited to, the following:

- Strict time basis. That is, a symbol is re-examined every so often, such as every 8 weeks.

- Trending. Certain symbols, such as symbols related to popular Twitter hashtags are re-examined on a much more frequent basis, perhaps even hourly, or by the minute.
- Symbols related to popular subject domains, such as finance or celebrities, are re-examined more frequently.

Richer Symbols

This section relates more to Topic Scout than some of the other sections, but the digression may still be useful.

For reasons specific to the origins of Topic Scout, Topic Scout focuses on finding symbols related to a topic that will be particularly useful for text classification, and to be able to do this in very tight semantic spaces. As two or more topics, perhaps highly specialized topics, get closer and closer together, Topic Scout still works. That is, Topic Scout can find differences when things semantically close together. On the other hand, Topic Scout has entirely ignored early identification of the meaning of words, of entity recognition, of finding relations etc. That wasn't its emphasis, and this decision also simply reflected the poor nature of many open source tools for these purposes at that time.

Instead, Topic Scout's approach to find things that are relevant, and then understand *later* what they are. As long as sufficient context is kept - that is, where did symbols come from - it is possible to mine greater meaning from symbols later. That is, find what is relevant first, then find the predicates, in the sense of subject-predicate-object triples.

This approach by Topic Scout is not, however, at all the only approach. It is entirely possible to envision symbols that can be annotated and promoted (in the sense of greater meaning) in many different ways. That is, a relevant symbol service could produce symbols that include, but are not limited to, information related to entity recognition, grammar, etc. Hence the meaning of symbols might be at a higher level much earlier in the life of symbol graph construction.

Graph Enrichment Service

Once the graph of symbols is in place, another important service comes into place. I call this the *graph enrichment micro-service*. This micro-service was not shown in the earlier architectural diagram because it is not a basic micro-service. It is an advance micro-service.

The symbol enrichment service has a mission. Its mission is to examine the symbol graph, or portions of the symbol graph, and finding further meaning in the graph and, if warranted, rewrite or extend the graph in ways to add this meaning to the graph. This graph enrichment service may also find patterns in the data, and then create patterns in the graph that match with these patterns. It may perform activities such as entity recognition as well.

Note: in some cases, especially when text mining is involved, there may need to be linkages to the text from which symbols were derived, so that the meaning of a symbol can be traced to its

many occurrences in text. Hence some implementations of a symbol crawler may take steps to save such text, as to provide appropriate context for later analysis in the context of the graph.

Knowledge Graph?

Is the symbol graph a knowledge graph? In its most basic form, probably not. As described here, symbols are pretty simple. The symbol graph needs more annotations, more entity recognition, more relations, etc. Until there is more meaning attached to symbols, maybe this is **not** a knowledge graph. That is a perfectly reasonable point of view.

On the other hand, such a symbol graph, as described here, can be annotated further, and enriched, thus increasing its value and making it more and more like a knowledge graph as opposed to a symbol graph.

In reality, symbol graphs and knowledge graphs are on a continuum.

Perhaps a symbol graph, in its basic form, is a kind of lower-tier precursor to a knowledge graph. But since the graph here is on a continuum, portions of such a graph could be elevated in meaning, and move towards a higher tier. The mission of the graph enrichment service could, in fact, do just this. Over time, implementations of the architecture might advance to the point that the data inserted into the symbol graph looks a lot more like knowledge.

Frankly, this is a bit too philosophical for me. Probably, best to get a large symbol graph up and running, and then things will sort themselves out.

Language

While this symbol crawler has implicitly assumed a single human language. Certainly, a symbol crawler that had graphs that incorporated different languages is possible by a skilled engineer. Consider DBPedia and its inclusion of many different languages. Certainly a symbol graph, such as described here, could handle multiple languages.

It might also allow for edges that connect symbols in different languages, as is also done in DBPedia.

At the very least, an implementation should incorporate a language dimension early on, so as to avoid expensive rewrites later.

Other Technologies

Lots of exciting work is happening with Word2Vec and Deep Learning. Certainly such exciting technologies might be applied here. For example, a symbol relevance micro-service could be built with some of the services described here, such as the symbol relevance service.

It would be possible to add new edge types to this symbol graph, such as relations, and add many other properties to symbols, including properties including, but not limited to, entity

recognition.

Seeding a Crawl

The URLs initially inserted into a web crawler queue may also be carefully chosen, and so the same applies to a symbol crawler.

A crawl of a symbol crawler could be seeded in many ways.

Here is just one example. For a broad-based crawl, symbols could be extracted from DBPedia and then inserted into the symbol queue. This might be done at random, perhaps selecting 5,000 symbols out of the millions of items in Wikipedia, and then letting the crawl take care of finding the connections, and finding new symbols unknown to DBPedia.

But the above paragraph is just an example. Skilled folks will come up with lots of ways to seed a crawl, and to control a symbol crawl.

Crawling Policies

Web servers have ways to control a crawl. Such controls might be manifested through crawling policies.

Again web crawlers can serve as a source of inspiration. A web server crawl might be associated with one or more of such crawling policies, or custom policies might be configured. Such crawling policies might, in some instances, even be able to be changed as the crawl is taking place.

Similar ideas can be adapted and applied to a symbol crawler.

Will It Work?

It is hard to know without trying. Certainly, one of the chief technical problems is creating a world-class symbol relevance micro-service.

For me, I have a big head start because of Topic Scout. But Topic Scout is not ready out of the box either. Analysis of the requirements here has certainly inspired some long hours, including careful re-examination of the topic lexicons produced by Topic Scout.

Some of the assumptions behind Topic Scout today don't match up with the requirements for such a symbol relevance micro-service. That's the bad news.

The good news is, however, that with some relatively minor extensions, it appears, from early tests, that these shortcomings may be addressable. The early data looks very good. If these early good results continue, then Topic Scout's relevance engine then could be used as the basis of a symbol relevance service.

The initial sense listing service could be quite basic, little more than calls into DBPedia. Over

time - perhaps many years - the aspiration would have a much more powerful sense listing service that would, as suggested earlier, combine data sources such as WordNet and DBPedia, as well as start including word sense induction.

Again, while the critical micro-services might improve other time, they could improve within a stable symbol crawling architecture.

Summary

This document has provided a description of a symbol crawler and its basic operations.

Two micro-services are critical to a symbol crawler, and these are *the sense listing micro-service* and the *symbol relevance micro-service*. Two otherwise identical implementations of a topic crawler could still be radically different in their results based on different implementations of these two services. But what is elegant about a symbol crawler is that, like a web crawler, its basic operation is simple. Very simple forms of these two micro-services could be built. Alternative, an implementation of a symbol crawler could have very advanced implementations of these micro-services, and produce and maintain much better symbol graphs.

A graph enrichment micro-service has also been described that can mine the graph for information, and do appropriate changes to the graph, to add richer meaning to the graph itself. While more conceptual in nature at present, this graph enrichment micro-service is also of great importance, because data mining can be, and should, used to extend the symbol graph itself.

In addition, such a symbol graph can be used to create runtime classifiers, such that text may be classified into one or more topics known to the symbol graph.

I have one last thing to say.

It is truly humbling to think of all the great work - with great people and great resources - working on the problem of knowledge graphs, both in big companies, start-ups and in academic research. I just want to get this idea of symbol crawler out there because, just possibly, it might provide a general framework that might be helpful, both to constructing simple symbol crawlers or very major symbol crawlers.

Stefan Gower