Web Search over Structured Data

B.Tech. Project Report
Stage 1

Submitted in partial fulfilment of the requirements for the degree of

Bachelor of Technology (Honours)

Student:
Ayush Kanodia
Roll #: 110050049

Guides:
Dr. S. Sudarshan
Dr. S. Chakrabarti

Department of Computer Science and Engineering
Indian Institute of Technology Bombay
Mumbai 400076, India

Abstract

In the domain of Web Search, search over structured as well as semi-structured data is now being increasingly used to supplement traditional text based search. Structured search techniques are fast becoming first class citizens in mainstream web search. This is clear from the fact that front line web search engines invest large amounts of their resources in providing entity results to users for selected queries, over and above URLs, as is the traditional way. Structured web search poses its own set of challenges, and there is a proliferation of work in this direction. The issues range from creation of semi-structured knowledge bases, using these knowledge bases to better annotate free form text data, annotating and extracting information in structured forms such as lists/tables, identifying different relevant query types and developing solution techniques for the same. In this work, we present a literature survey of some of these questions and the work which aims at answering them. We end with a proposal of future work, which we aim to undertake during the second half of this project.

Acknowledgements

I wish to gratefully acknowledge the constant support and guidance by my guides, Dr. S. Sudarshan and Dr. S Chakrabarti. Without their support, this work is a distant dream.

Ayush Kanodia
B.Tech, Year IV
CSE, IIT Bombay

Contents

Chapter 1

Introduction

In the web search world today, structured search is fast becoming essential to returning quality answers to queries. The initial paradigm of web search was to take a freeform query, and return a list of ranked web urls based on relevance of the document text to the freeform query. While this is a widely researched question, we are now shifting to the paradigm of returning entity results to certain, if not all queries. We have recognized that the information need of a web query might fundamentally be best satisfied by a targeted answer (often referred to as an entity), which might be inferred by collective evidence over numerous webpages, as opposed to a webpage which contains information about the desired answer. Note that there are at least two distinct advantages that the user enjoys in this new setup - He gets an answer based on automated collective information from multiple sources, and at the same time, he saves time by not having to delve into a web page and seek the answer himself.

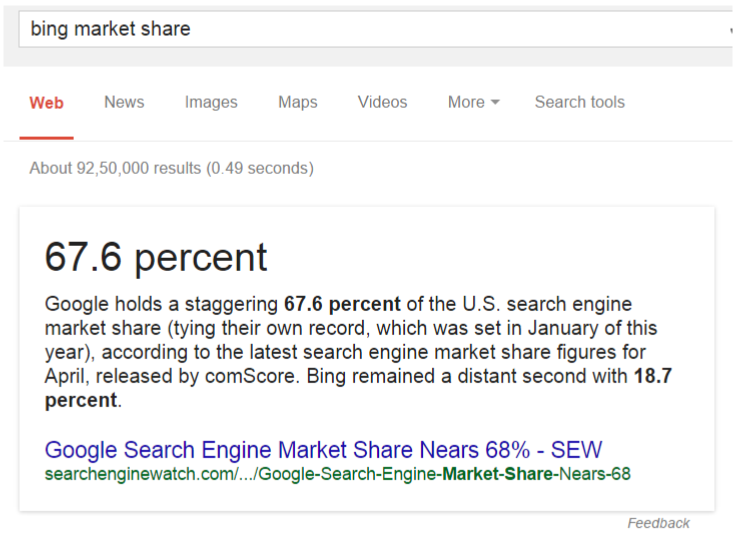
1.1 Strategy

The strategy to answering queries with entities entails two somewhat independent steps.
The first is of curating knowledge bases which are a repository of structured data. These knowledge bases are further used to annotate entity mentions on existing webpages. Both these questions, the first of creating these knowledge bases, and the second of using these knowledge bases to annotate existing entity mentions are areas of wide research, with numerous suggested strategies, applying general pattern recognition techniques, as well as specific and sophisticated Machine Learning techniques.

The second important step is to recognize the types of queries of interest to users of different domains, and to formalize strategies to answer such queries. It is important to recognize queries which are best answered by entities over webpages, as these indeed are fundamentally of a different nature than standard url based search over web search engines. Furthermore, it is a separate and a difficult task to devise methods and strategies to answer each specific type of query - both with regard to using the algorithms, as well as the data sources for this purpose.

1.2 Structured Data Extraction

This step refers to creation of large scale knowledge bases which are collections of structured data, and the use of these to annotate the web, wherever possible, to enable powerful entity search and complex queries over relational data. Some such knowledge bases include Freebase (`www.freebase.com`), YAGO (`www.mpi-inf.mpg.de/yago-naga/yago/`) and the Wikipedia hierarchy (`en.wikipedia.org/wiki/Wikipedia:FAQ/Categorization`).
In this work, as a step in this direction, we look at structured data extraction techniques over the web, both in free form text as well as over lists and tables. It must be noted that all of this can possibly not be fully automated, so that widespread human intervention and labour is needed. Nevertheless, minimizing such human involvement with high quality extraction is still a worthy task. Also, it is worth noting that pre indexing structured data might not be possible for certain tail scenarios, and that processing a dump of webpages on the fly might be needed at query time itself.
As a second step, an important task is, given such a knowledge base of entities, we need to be able to tell which entities referenced on a generic webpage refer to which entity in our catalogue. This can be thought of as an inverted index in terms of entities over keywords, only that there is an additional issue of resolving entity mentions (here also, assuming that we do already know of entity mentions, which in itself may not be a solved issue).
In this work, we look at basic entity resolution / disambiguation techniques for annotation of webpages. Such annotations later (in stages downstream) help web search engines improve entity search, while providing evidence for their responses. This is rather challenging, as can be seen in 1.1, which is a snapshot from arguably the best search engine of our times.


bing market share

About 92,50,000 results (0.49 seconds)

67.6 percent

Google holds a staggering **67.6 percent** of the U.S. search engine market share (tying their own record, which was set in January of this year), according to the latest search engine market share figures for April, released by comScore. Bing remained a distant second with **18.7 percent**.

Google Search Engine Market Share Nears 68% - SEW
searchenginewatch.com/.../Google-Search-Engine-**Market**-**Share**-Nears-68

Figure 1.1: A Google query on Bing's market share

1.3 Query Answering

Once we have indexed annotations as well as structured knowledge bases, along with our usual unstructured text document corpus, many interesting questions arise. The first among them is - what are some interesting questions that users might seek answers to? Indeed, a lot more can be done using entity search than just standard web search, since we fundamentally have more leverage in terms of indexed information available, as well as relational structure in knowledge bases.

In this work, we look at some such interesting queries, which are but a fraction of the many different types of queries that might be interesting. Further, we look into strategies to tackle each such query, and present overviews of different approaches for each query type. Notable queries include freeform text queries (Telescopic Queries), queries with explicitly mentioned type and context requirements, open question answering, ”near queries”, table augmentation queries and quantity (consensus) queries.

An interesting and notable point in this context is that while the number of different queries that might be interesting are many, there is no unifying framework over these different queries. It might be interesting to explore the common features of each such query type, and the fundamental commonalities and differences between the ideas and techniques employed by numerous search techniques over these different queries. Also, the proliferation of research over numerous different types of queries is an indicator of how enabling structured data extraction on the web can be.

1.4 Organization of the Report

The rest of the report is organized as follows: We first talk about various information extraction questions, and proposed solution overviews for each of them. We then talk about various query answering questions, and proposed solution overviews for each of these. Finally, we propose a high-level plan of future work, and conclude.

Chapter 2

Structured Data Extraction

The first step to move towards search over structured/semi-structured data is to identify and extract structured data. This may be in the form of entities in free form text, or text in web tables and lists, as well as relations between entities in both free form text as well as structured data. Further, once we extract such data and create a knowledge base or a catalogue of this information, we must annotate mentions of such entities on other webpages, which can be considered as the equivalent of indexing over structured data. The goal of this chapter is to:

- Identify structured data extraction/annotation tasks
- Present work on solving such tasks

2.1 Entity Resolution

Suppose we have a knowledge base of entities, along with an ontology of their types, viz. a type hierarchy. An examples of such a knowledge base is Freebase, and an example of such a type hierarchy is the Wikipedia category hierarchy. Our task is to annotate entity mentions in free form text using entities from such a knowledge base. For our purpose, the existence of such knowledge bases can be pre assumed.

The following are samples of free form text that might be found on webpages:

- After the UNC workshop, **Jordan** gave a tutorial on non parametric Bayesian methods
- After a three season career at UNC, **Jordan** emerged as a league star with his leaping ability and slam dunks.

While the first of the above mentions of Jordan refers to the Machine Learning researcher Michael Jordan, the second mention is that of Michael Jordan, the famous basketball legend.

These text snippets are an apt example of the difficulties faced in the process of entity disambiguation. There are numerous signals that need to be exploited in order to achieve accurate entity disambiguation. In the above context, it is important to note that UNC is a common signal in both cases, which does not give very strong evidence. However, the signal "tutorial" and "non parametric Bayesian methods" clearly indicates which Jordan is being talked about. Similarly, in the second case, "league star", "leaping ability" and "slam dunks" are strong signals that the basketball player is being talked about in this context.

The first rudimentary approaches to entity disambiguation looked at snippet contexts of identified named entities. They used prior learnt models to classify each such entity mention based solely on its local context. This had certain limitations. The first of these is that this approach might not able to disambiguate very well, since there may not be comprehensive enough signals in the surrounding text of an entity mention. The other limitation is that such an approach is supervised, so it requires labeled data. To some extent, training data is already available from sources such as Wikipedia, which already has entity annotations on its webpages.

To tackle these issues, global consensus needs to be used, at least over a document. The key idea is that over a document, it might be easier to disambiguate entities together with better confidence, than disambiguating them individually. This is because it is believed that a document comes from a coherent topic distribution (if not a single topic), which brings in a fundamental correlation between the entities mentioned on the document.

2.1.1 Latent Dirichlet Model for Unsupervised Entity Resolution

In [1], a Latent Dirichlet model is used for Entity Resolution. This work aims to resolve references when they are connected to each other via relational links, as in the bibliographic domain, where author names in papers are connected by co-author links. Now, entity resolution becomes collective in that resolution decisions depend on each other through the relational links. Collective resolution, as discussed in this work, performs better over independent pair-wise resolution. We now describe briefly the LDA Model for authors.

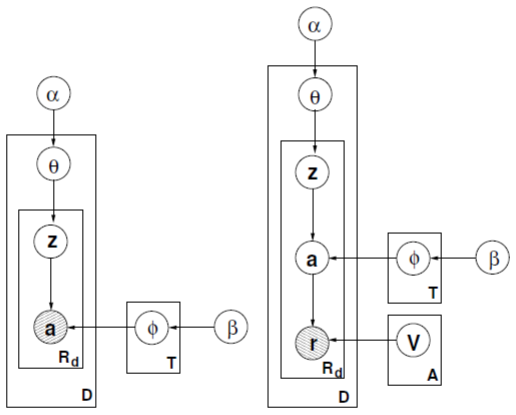


Figure 2.1: (a) LDA model for authors and (b) LDA model for authors when the authors are not observed

LDA Model for Authors

Consider a collection D of documents and a set of A authors who write these documents. Let R be the set of author references $a_1, a_2, ..., a_R$ in these D documents. Each document may have multiple authors, and we first assume that we can observe exactly the author for each document. for the i^{th} author reference, a_i denotes the author it corresponds to and d_i denotes the document it corresponds to. Further, there is the notion of collaborative author groups. These are groups of authors which tend to co-author together. It is assumed that there are T different groups, and that each author reference a_i has an associated group label z_i. The probabilistic model is shown using plate notation in 2.1 The probability distribution over authors for each group is represented as a multinomial distribution with parameter ϕ, where there is one such parameter for each author group. The authors of each paper are modelled as coming from a mixture over all groups, and this distribution is again a multinomial with parameter θ. For a particular document, θ is drawn from a Dirichlet prior with hyperparameter α; similarly each ϕ is drawn from a Dirichlet prior with hyper parameter β. This is depicted by the plate diagram in 2.1(a), where the authors are assumed to be perfectly observed.

Note that this scheme makes the assumption that a document can come from multiple author groups, and since each author is sampled independently, this permits a document to have repeated authors. This is a model limitation.

However, this model assumes that the author identity can be determined unambiguously from each author reference. When we are dealing with author names, this may not happen. 'Alfred V. Aho', 'Alfred Aho', 'AV Aho', etc. are different ways of expressing the same name, for instance. This cannot even be made rule-based, for instance, since 'S.Johnson' and 'Stephen C. Johnson' might not even refer to the same author.

To capture this, an extra attribute, v_a is associated with each author a. An extra level is added to the model that probabilistically modifies the author attributes to generate the references r. The corresponding plate diagram is shown in 2.1(b).

The inference problem, given the general LDA model turns out to be intractable in general. An approximation sampling method is used for inference.

2.1.2 Other Work

Topic modelling is one method of collective disambiguation of entity references. However, other methods for collective disambiguation of entity mentions have also been explored. There has also been work on improving the efficiency of such a process.

- In [2], a probabilistic graphical model is used for collective entity disambiguation of Wikipedia entities in Web Text
- In [3], a probabilistic graphical model is used for annotating web tables
  - To associate types with columns
  - Annotate pairs of columns with binary relations
  - Annotate table cells with entity IDs
- In [4], efficient data structures (for in memory storage of disambiguation models) and storage strategies (for annotation indices) are presented

2.2 Extracting Structured Data

Many web information resources present relational data - however, in a format that makes it friendly for human readers. Therefore, while there is great potential for use of relational data if it can be extracted correctly, the task of extraction is difficult. A sample html snippet of relational data is 2.2.

```
<HTML><TITLE>Some Country Codes</TITLE>
<BODY><B>Some Country Codes</B><P>
<B>Congo</B> <I>242</I><BR>
<B>Egypt</B> <I>20</I><BR>
<B>Belize</B> <I>501</I><BR>
<B>Spain</B> <I>34</I><BR>
<HR><B>End</B></BODY></HTML>
```

Figure 2.2: An HTML snippet example depicting relational data found on a webpage - of countries and country codes

2.2.1 Wrapper Induction

In order to use such sources of information, it is necessary to translate such data as in 2.2 to its relational form. One of the ways to do that is to use a *Wrapper*. A wrapper is a procedure (specific to a particular information source), that translates data to relational form. Prior to their automation, wrappers were manually coded for sources of information. As can be thought, this was a painstaking and error prone process.
In [5], a new technique for learning a wrapper has been proposed. This process is referred to as wrapper induction. The system learns a wrapper by generalizing from example query responses. A PAC analysis generates bounds on the number of examples to learn a reasonably satisfactory wrapper.

Wrapper Classes

In 2.2, we find that particular strings ($< B >$ and $< I >$) appear in regular positions before and after the data strings of interest. It appears that countries are surrounded in between html bold tags, while codes are between italic tags. However, the strategy of looking at just the LR(Left-Right) surrounding strings of data strings fails, since not all occurences of $< B > ... < /B >$ denote a country. Hence, a more flexible wrapper - the HLRT wrapper is used. 2.3 shows the template for HLRT wrappers. Note that instantiating the wrapper with the six tuple ($< P >, < HR >, < B >, < /B >, < I >, < /I >$) yields the wrapper in 2.3. The formal definition of an HLRT wrapper is therefore, a vector of 2K + 1 strings for a domain with K attributes for tuple. The first string (h) marks the end of the header, while the last (t) marks the beginning of the tail. For attribute k, l_k and r_k are delimiters.

Constructing Wrappers by Induction

Given a sample of example webpages, we need to learn a wrapper for the information resource that generated them. For us -

```
ExecuteHLRT($\langle h, t, \ell_1, r_1, \ldots, \ell_K, r_K \rangle$, page $P$)
  skip past first occurence of $h$ in $P$
  while next $\ell_1$ is before next $t$ in $P$
    for each $\langle \ell_k, r_k \rangle \in \{\langle \ell_1, r_1 \rangle, \ldots, \langle \ell_K, r_K \rangle\}$
      skip past next occurence of $\ell_k$ in $P$
      extract attribute from $P$ to next occurence of $r_k$
  return extracted tuples
```

Figure 2.3: The HLRT wrapper template

```
BuildHLRT(labeled pages $\mathcal{E} = \{\ldots, \langle P_n, L_n \rangle, \ldots\}$)
   Note that each label $L_n$ partitions page $P_n$ into its
   attributes, separated by the strings between tuples
   and between the $K$ attributes within a tuple.
  for $k = 1$ to $K$
    $r_k \leftarrow$ any common prefix of the strings following each
          (but not contained in any) attribute $k$
  for $k = 2$ to $K$
    $\ell_k \leftarrow$ any common suffix of the strings preceding each
          attribute $k$
  for each common suffix $\ell_1$ of the pages' heads
    for each common substring $h$ of the pages' heads
      for each common substring $t$ of the pages' tails
        if (a) $h$ precedes $\ell_1$ in each of the pages' heads; and
           (b) $t$ precedes $\ell_1$ in each of the pages' tails; and
           (c) $t$ occurs between $h$ and $\ell_1$ in no page's head; and
           (d) $\ell_1$ doesn't follow $t$ in any inter-tuple separator
        then return $\langle h, t, \ell_1, r_1, \ldots, \ell_K, r_K \rangle$
```

Figure 2.4: The **BuildHLRT** algorithm

- **Instances** correspond to webpages, and **Labels** correspond to pages; tuples.
- **Hypotheses** correspond to HLRT wrapper template parameters

Wrapper induction proceeds by accumulating a set η of labelled sample pages. On each iteration, an algorithm, **BuildHLRT** is called with η, which returns wrapper w. Learning stops when we have a good enough w.

A wrapper is said to be consistent with a labeled page if it generates the label for the page. Figure 2.4 shows the **BuildHLRT** algorithm. It iterates over all choices for all the delimiters, stopping when a consistent wrapper is encountered. One of the limitations of this approach is that it does not consider hierarchy in the relational data, which is looked at in [6]

2.2.2 EXALG

While wrapper induction has been successfully applied to the problem of extracting relational data from webpages, one of its major limitations is that it is a supervised method - and requires considerable amount of training data. In [7], an algorithm is presented, which takes, as input, a set of template generated pages, and deduces the unknown template used to generate the pages. It extracts, as output, the values encoded in the pages. Unlike previous work in this area, this approach is unsupervised. One of the major challenges that needs to be handled in this context is that the schema embedded in a webpage is often not flat - but it is hierarchical and semi structured. This makes the task of definition as well as automatic recognition of templates harder. Unlike other work in this area, this work does not assume that HTML tags always form part of the template of the page,and cannot be part of the data, although this is statistically likely. However, it must be noted that this work does not aim to semantically name or label the extracted data. That is a postprocessing step. At the same time, it is assumed that all the input pages conform to a common schema and template.

Definitions

A type is recursively defined as:

- *Basic Type*, B, represents a string of tokens.
- If $T_1, T_2, ...T_n$ are types, then their ordered list $< T_1, ..., T_n >$ is also a type, formed using a tuple constructor of order n
- If T is a type, then $\{T\}$ is also a type constructed from T using a set constructor.

An instance of a schema is defined recursively as:

- An instance of the basic type, B, is any string of tokens
- An istance of type $< T_1, T_2, ..., T_n >$ is a tuple of the form $< i_1, i_2, ..., i_n >$, where i_j is an instance of type T_j, for every j.
- An instance of type T is any set of elements $e_1, ..., e_m$ such that $e_i(1 <= i <= m)$ is an instance of type T.

A template T for a schema S, is defined as a function that maps each type constructor τ of S into an ordered set of strings $T(\tau)$, such that,

```
⟨
⟨html⟩1⟨body⟩2
      ⟨b⟩3 Book4 Name5 ⟨/b⟩6 *
      ⟨b⟩7 Reviews8 ⟨/b⟩9
      ⟨ol⟩10
      {⟨
          ⟨li⟩11
              ⟨b⟩12 Reviewer13 Name14 ⟨/b⟩15 *
              ⟨b⟩16 Rating17 ⟨/b⟩18 *
              ⟨b⟩19 Text20 ⟨/b⟩21 *
          ⟨/li⟩22
      ⟩}
      ⟨/ol⟩23
⟨/body⟩24 ⟨/html⟩25
⟩
```

Figure 2.5: A sample template

- If τ is a tuple constructor of order n, $T(\tau)$ is an ordered set of $n+1$ strings $< C_\tau 1, ..., C_\tau(n+1) >$.
- If τ is a set constructor, $T(\tau)$ is a string.

A sample template is shown in 2.5. A natural definition of rendition of values (tuples) following a given schema and a template follows naturally from the above defintions.

Statement

Given these defintions, we are faced with the following problem - Given a set of pages, created from some unknown template, schema, and tuples (values), deduce the template and the tuples from the set of pages alone.

Solution Overview

The solution algorithm, called **EXALG**, computes ”equivalence classes” in the first stage - sets of tokens having the same frequency of occurence in every page in our set of pages. EXALG retains only the equivalence classes that are large and whose tokens occur in a large number of input pages.

Such equivalence classes are called LFEQs (Large and Frequently occuring Equivalence classes). In its second stage, EXALG builds an output template T and a schema S. In order to construct S, EXALG first considers the root LFEQ - the LFEQ whose tokens occur exactly once in every input page. Lets call it η_1. EXALG determines the positions between consecutive tokens of η_1 that are non empty. If a non-empty position does not have any equivalence classes within it, then it infers the type in this vacancy to be B, the basic type. Else, it recursively constructs the type for this vacancy. Thus, EXALG constructs the schema and template for a given page. Once this is done, it uses this schema and template to extract values from this set of webpages. **Challenges:** Some of the challenges in this process include

- differentiating roles of tokens based on context - for instance name has different roles in first name and last name
- differentiating roles of tokens based on already found LFEQs

2.2.3 IEPAD: Information Extraction Based on Pattern Discovery

Yet other methods of information extraction have been looked into. In [8], extraction rules are learn automatically from web pages, using a pattern recognition technique based on suffix trees. The system can automatically identify record boundaries by repeated pattern mining and multiple sequence alignment. This process is completely unsupervised, and involves no human effort or domain specific heuristics. The block diagram for the extraction rule generator of IEPAD is shown in 2.6. What follows is a description of the IEPAD extraction rule generator:

- Each html tag token is assigned a unique bit string, while all other text strings are assigned a common bit string. All these bit strings are of the same length. Refer to 2.7 for an example of the assignment for the text in 2.8
- A PAT tree constructor (PAT tree stands for Patricia Tree, which is a variant of a suffix tree) constructs a suffix tree out of the translated text on the page. A sample PAT tree for the data in 2.7 is shown in 2.9.
  - As a PAT tree is already a suffix tree, every internal node already corresponds to a label which is **right maximal** (cannot be further extended to the 'right').

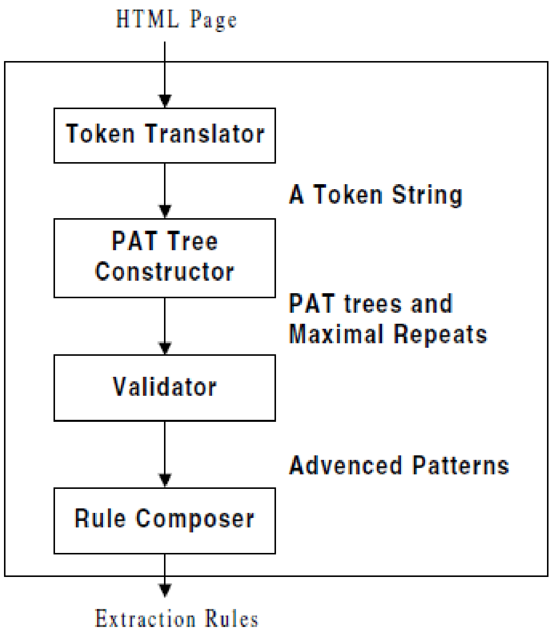


Figure 2.6: IEPAD extraction rule generator block diagram

<B>Congo</B><I>242</I><BR>
<B>Egypt</B><I>20</I><BR>$

Figure 2.7: An instance of relational data

Html(<B>)	*000*
Html(</B>)	*001*
Html(<I>)	*010*
Html(</I>)	*011*
Html(<BR>)	*100*
Text(_)	*110*

Figure 2.8: An instance of the translated tags and text by IEPAD

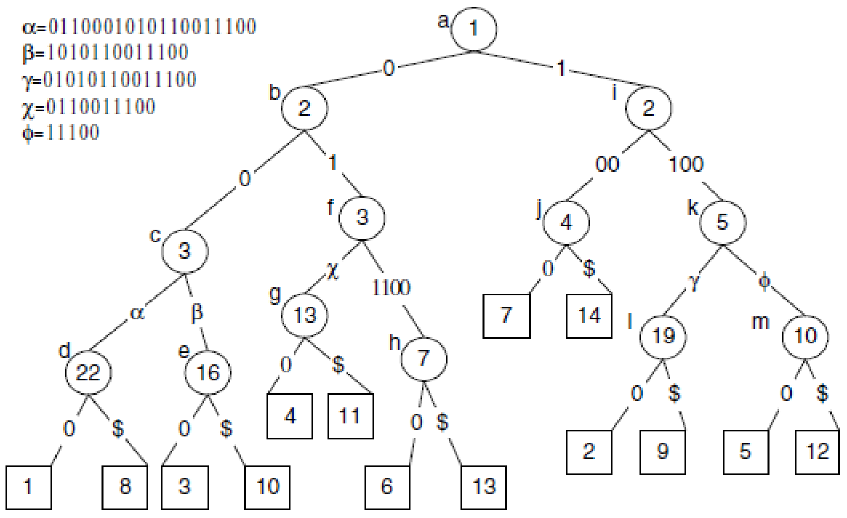


Figure 2.9: A PAT tree for the data in 2.7

- The PAT tree tries to find sequences of text which are **maximal repeats** (which cannot be further 'extended' to the left or to the right). For, this, we just need to find which nodes are **left maximal** (cannot be 'extended' to the left). All such nodes are maximal repeats, as they are all already right maximal. For this to be true, it must happen that at least two leaves in the node's subtree should have different characters to the left of them.

- Once such maximal repeats are generated, the patterns are validated by a validator based on
  - **Regularity**: This is computed by measuring the standard deviation between multiple occurences of the maximal pattern
  - **Compactness**: This is computed based on the density of the extracted maximal repeat on the webpage
  - **Coverage**: This is a measure of the coverage or span of text encompassed by a maximal repeat

- Finally, after validation, rules for extraction are generated for a valid maximal repeat. This is the pattern which will be now used to mine tuples.

One of the major disadvantages of IEPAD, even as mentioned earlier, is that it considers all HTML tags to be part of the template, and not the text. Other methods of pattern discovery are also used in the extraction of structured data. For instance, in [9], Partial Tree Alignment of the Document Object Model is used to find out repititive patterns.

2.3 Relation Extraction from Text

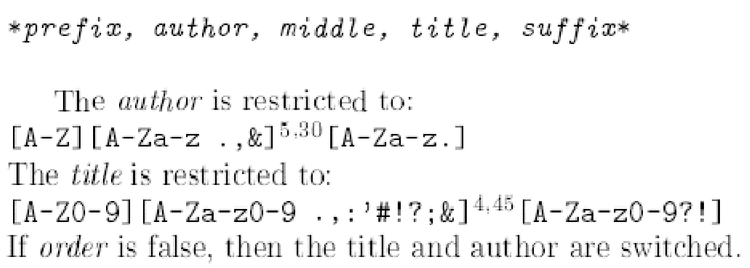
In the previous section, we considered the extraction of relational data from webpages, where it was presented to the reader in a human friendly format. Here, in this section, we consider the issue of extracting relational data, but from free form text. It is true that a lot of relational data also appears as such, and such sources of information can also enrich our sources of data greatly.

2.3.1 DIPRE

In [10], the problem of extracting the specific relation of (author, tilte) pairs from the Web is considered. This work is the basic introduction of an idea, which was picked upon and explored in detail subsequently. A small seed set of (author, title) pairs (the seed set was as small as just 5 tuples), was taken as input. Then, all possible occurences of those books on the Web (or a subset of the Web) were found. From these occurences, patterns were recognized for the citations of books. Then the Web was searched for these patterns, and new books were found. This combined set of books was now treated as the new seed set, and the entire process was repeated, until there was a large enough list of tuples.

Pattern Relation Duality

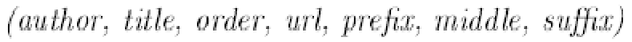
This method is thus called DIPRE - Dual Iterative Pattern Relation Expansion. It relies on a duality between patterns and relations. An important observation is that given a set of patterns, P with high precision , a good approximation to the target set can be generated. However, the converse property of generating a good set of patterns from tuples is also desired. This is done by finding all occurences of the tuples in D and discovering the similarity in their occurences. A pattern is given by an instantiation of the regular expression in 2.11, and an occurence is described by the seven tuple as shown in 2.11



prefix, author, middle, title, suffix

The *author* is restricted to:
`[A-Z][A-Za-z .,&]`^{5,30}`[A-Za-z.]`
The *title* is restricted to:
`[A-Z0-9][A-Za-z0-9 .,:'#!?;&]`^{4,45}`[A-Za-z0-9?!]`
If *order* is false, then the title and author are switched.

Figure 2.10: Patterns in DIPRE



(author, title, order, url, prefix, middle, suffix)

Figure 2.11: Occurences in DIPRE

Identification of Patterns

Once tuples are identified on a webpage belonging to the seed set, patterns are generated by ensuring that the 'order' and 'middle' parameters of all occurences is the same. The urlprefix is the longest matching prefix of all urls, the prefix is the longest matching suffix of the prefixes of the occurences, and the suffix is the longest matching suffix of the prefixes of the occurences.
Pattern Specifity The patterns allowed should not be too generic. This is because they may then trigger off the selection of incorrect tuples as answers, and this may further trigger off even more incorrect tuples being selected, as this approach is a bootstrapping approach. The specificity, for quick computation, depends on the length of the middle, urlprefix, prefix, and suffix values of the pattern. Patterns with low specificity are rejected.

2.3.2 Snowball

DIPRE was a first step in the direction of relation extraction from free form text, given a few seed tuples. However, there were numerous flaws with DIPRE.

- A large number of incorrect tuples were being tagged as correct by DIPRE
- Patterns were very strictly defined, in that they had low flexibility and only supported exact matches, which was difficult
- The reliability of newly formed extraction templates was not very high

In [11], a system called SNOWBALL is described for relationship extraction, which builds upon DIPRE in the following important ways. The main

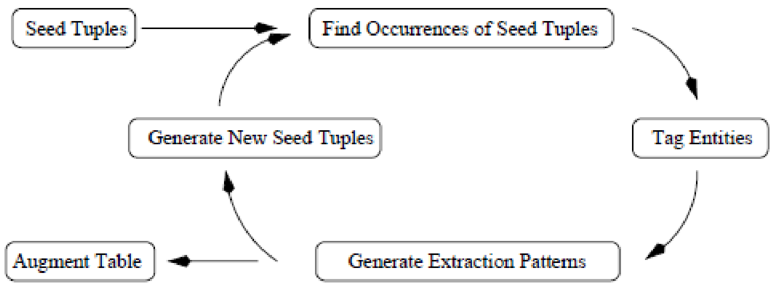


Figure 2.12: The main comnponents of Snowball

```
<ORGANIZATION>'s headquarters in <LOCATION>
<LOCATION>-based <ORGANIZATION>
<ORGANIZATION>, <LOCATION>
```

Figure 2.13: The use of named-entity tags in Snowball patterns

components of Snowball can be found in 2.12

- Named entity recognition is used to improve the quality of matches. The seed tuples are associated with their entity types, and this information is used during matching. More semantic information such as hypernymy/hyponymy could be used using knowledge bases as Wordnet. This improves extractions in many cases. For instance, for the seed tuple 'Redmond-based Microsoft', the pattern in 2.13 is used. This eliminates false tuples like 'computer-based learning' or 'alcohol-based solvents' from showing up.

- To increase coverage during matching, Snowball uses a vector model for patterns, instead of the fixed regular expression that was used by DIPRE. Thus, the *left*, *middle*, and *right* 'contexts' associated with a pattern are represented analogously as how the documents and queries are represented in information retrieval in vector space models.

- Finally, Snowball improves the evaluation of patterns and tuples based on

  - Evidences for the pattern used to generate the tuple,
  - Specificity of the pattern
  - Corroboration of a pattern by the tuples that support it found earlier

$$\begin{bmatrix}\text{protesters}\\ \text{NNS}\\ \text{Noun}\\ \textsc{Person}\end{bmatrix} \times [\rightarrow] \times \begin{bmatrix}\text{seized}\\ \text{VBD}\\ \text{Verb}\end{bmatrix} \times [\leftarrow] \times \begin{bmatrix}\text{stations}\\ \text{NNS}\\ \text{Noun}\\ \textsc{Facility}\end{bmatrix}$$

Figure 2.14: Feature vector for shortest path in the dependency parsed kernel

2.3.3 A Shortest Path Dependency Kernel for Relation Extraction

In [12], a deep NLP approach to extracting relational information from a corpus has been explored. Since this approach employs NLP, the corpus used is a newspaper corpus, where well formed sentences are very likely, and this is suited to this mode of extraction. The intuitive strategy employed is to

- First, it is assumed that relationships need to be identified only within a sentence. That is, both entities involved in a relationship lie in the same sentence. This sentence is dependency parsed, using NLP techniques
- The shortest path in the dependency tree between candidate entities is considered. The belief is that this shortest path in the dependency tree is very likely to capture most, if not all the features required to decide whether the two entities in question are in a relationship.
- A feature vector is designed using this shortest path. This feature vector includes the following features:
  - The word at each node in the dependency graph
  - The POS tag of these words
  - The NER (Named Entity Recognition) type of this word
  - The direction of the dependency relation (eg. Verb - Noun) at each level.
- The feature vector design can be noticed in 2.14
- Now, suppose we have some labelled data. We can extract feature vectors on this labelled data, and learn a model vector w, which can be used for SVM like classification over this hand curated vector which has just been described. This is how classification and inference is achieved

Limitations This is a sophisticated NLP based approach to Relation Extraction. However, it must be noted that this means that this process is more computationally demanding, as with usual NLP processes over IR. Also, importantly, this technique may work only on certain data sources - essentially those which consist of well formed sentences.

Chapter 3

Query Answering

Having looked at certain information extraction questions in the first part of this work, we now explore some of the interesting queries that must be answered. It is interesting to note that numerous different questions have been looked into via multifarious approaches. The strategy in this section will be to identify certain query types, and look at solution strategies for them

3.1 Telescopic Queries

Telescopic queries refer to typical web search queries - these queries have no well defined structure, and through query words, provide strong hints to the answers required. For instance, let us consider the telescopic query - *chinese city many international companies.* This query translates to the natural language question *Which Chinese cities have many international companies?* A sample answer list for this question (ordered list) may be $Beijing, Shanghai, Guangzhou$

The roles played by different words in this query are different. For instance, the role played by the words "chinese city" (in the context of entity search) is to provide signals about which types of responses are expected. Such words are referred to as hint words. On the other hand, words like "international companies" shows the property which the user desires about these cities. Such words are known as "selectors", they are roughly equivalent to selection predicates (as in selection over databases). For the purpose of answering this query, a corpus annotated with entities is assumed to exist.

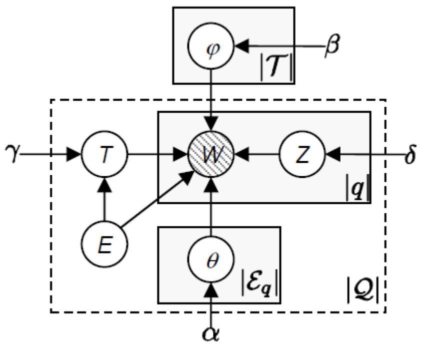


Figure 3.1: Plate diagram for generative model for joint query interpretation and ranking

3.1.1 Generative Formulation

Generative models have seen significant success in corpus modelling and entity ranking. Therefore, a natural proposition as in [13] is to use a generative language model approach to *joint query interpretation* and *response ranking.* It is important to note that these two or not done in separate, independent steps, unlike certain other formulations. An entity e is fixed, and the query words generated, by taking the following steps.

- Choose a type
- Describe that type using hint words
- Generate the remainder of the query by sampling words from snippets that mention e

The plate diagram for this formulation is shown in 3.1 We now describe the individual components in brief.
Choosing a type given an entity This is the question of designing Pr(t | e). A simple way of doing this is to use a query log with ground truth, and accumulate a hit count for each type. This is what is used.
Type description language model The assumption that hint words are conditionally independent (given the type) is used. Each type is then described by one or more *lemmas* (descriptive phrases). We use a multivariate Bernoulli distribution derived from each lemma, and in case a type has multiple lemmas, we take the maximum over all lemmas. We smooth this distribution with the word occurence probability over all types in the type hierarchy
Entity snippet language model The selection words of the query are

sampled from the set of words that occur in the snippets of the mention candidate entity in the training data. A similar distribution is used as for type descriptions. This time, the smoothing is over word occurence probability over all snippets.
Query word types We model each word in the query as being either a hint word or a selector word according to a Bernoulli distribution, using a tuned parameter.
Finally, in order to calculate the probability of a query, we take expectation over the type of the entity as well as over the query word types.

3.1.2 Impression Model

In [14], an *Impression Model* is used to rank entities for queries. A slightly different format for queries is used. In these types of queries, the type being sought is explicitly identified, as well as the format in which query components should occur in text. For instance, consider the following two queries

- *ow(amazon customer service # phone)*
- *uw(# title="hamlet" #image #price)*

In the first of these queries, the type of result being sought is phone number, and the query tokens should be present in an ordered fashion around the occurence of the phone number (in a fixed window size). In the second of these queries, an unordered window is allowed, while two entity types are sought. One of the selector words is of type *title*, and its value is specified to be *hamlet*.
An *impression model* is used to solve this. The model posits an observer, who visits a random subset of documents from the given set of documents with a probability equal to a measure of quality of the document (say pagerank). He samples evidence for entities via the snippets in which he sees them. He collects this evidence across webpages, and the answer is the belief having witnessed these snippets. This set of evidences for each possible entity tuple is a candidate answer, and is used to rank the set of candidate tuples. The scoring model is briefly summarized in 3.2 In this model, p_o is the observed value of probability of an entity tuple by the model proposed. It based on the following components

- **Global Aggregation**: This refers to the probability of visiting each document, and is a page rank like measure
- **Local Assessment**: This refers to the confidence measure of a piece of evidence. This is based on two measures - the confidence of annotation of entities and the probability of that context fitting the query

- **Query:** $q(\langle E_1, \ldots, E_m \rangle) = \alpha(E_1, \ldots, E_m, k_1, \ldots, k_l)$ **over** $\mathcal{D}$
- **Result:** $\forall\, t \in E_1 \times \cdots E_m$: Rank all t by computing $Score(q(t))$ as follows.

(1) $Score(q(t)) = p_o \cdot \log \frac{p_o}{p_r}$, **where**

$$(2)\quad p_o \equiv p(q(t)|D) \;=\; \sum_{d \in D} \mathbf{PR}[d] \times \max_{\gamma}\Big(\prod_{e_i \in \gamma} e_i.conf \times \alpha_B(\gamma) \times p(s|\gamma)\Big)$$

$$(3)\quad p_r \equiv p(q(t)|D') \;=\; \prod_{j=1}^{m}\Big(\sum_{e_j \in d, d \in D} p(d)\Big) \times \prod_{i=1}^{l}\Big(\sum_{k_i \in d, d \in D} p(d)\Big) \times \prod_{j=1}^{m} \overline{e_j.conf} \times \frac{\sum_s p(q(t)|s)}{|s|}$$

Figure 3.2: A scoring model for entities

As a new layer, the validation layer statistically validates the significance of the "impression". This value is represented by p_r, and we skip the details of its formulation. The final result is the score as given by $p_o log(p_0/p_r)$, which is measure of coherence between the quantities p_o and p_r.

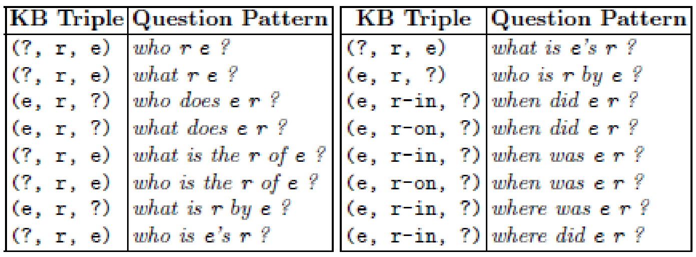
3.2 Open Query Answering

Consider a query which is presented in the form of a question

- What is a parrotfish's habitat? (southern water)
- What is the main language of Hong Kong? (Cantonese)

Different approaches to answering these types of questions have been studied in the literature. Two of these are

- **NLP approaches**: The objective of such an approach is to perform complex natural language processing on the free form question, such as parsing, and to convert the query into a more structured query, such as a database query. This process may lead to hardening of certain decisions at query time, such as the type being sought. Moreover, this method requires considerable human intervention to hand craft lexicons, grammars and knowledge base schema for the parsing to be effective
- **Transformation to Vector Representations** Another approach to this is to transform questions and answers to vector spaces. These methods do not need manually defined grammars/lexicons and can query knowledge bases independent of their schema. However, these methods do require supervision



KB Triple	Question Pattern	KB Triple	Question Pattern
(?, r, e)	*who r e ?*	(?, r, e)	*what is e's r ?*
(?, r, e)	*what r e ?*	(e, r, ?)	*who is r by e ?*
(e, r, ?)	*who does e r ?*	(e, r-in, ?)	*when did e r ?*
(e, r, ?)	*what does e r ?*	(e, r-on, ?)	*when did e r ?*
(?, r, e)	*what is the r of e ?*	(e, r-in, ?)	*when was e r ?*
(?, r, e)	*who is the r of e ?*	(e, r-on, ?)	*when was e r ?*
(e, r, ?)	*what is r by e ?*	(e, r-in, ?)	*where was e r ?*
(?, r, e)	*who is e's r ?*	(e, r-in, ?)	*where did e r ?*

Figure 3.3: Patterns for question generation from a knowledge base

3.2.1 Open Question Answering Using Weakly Supervised Embedding Models

In [15], a "weakly supervised" embedding model has been used (into a low dimensional vector space). The objective is to learn vector embeddings of the words in a question, as well as of Knowledge Base triples so that the representations of questions and corresponding answers end up being close together in the target embedding space. Key points to note are that the question and answer embeddings are learnt in the same space. Also, each tuple in the knowledge base is treated as a candidate solution to a question, and the tuples for an answer are ranked in order of closeness to the question representation.

The knowledge base that is used for this purpose is ReVerb. ReVerb is a knowledge base of the type of Freebase. It stores approximately 14 M triples, 2 M entities, and 600k relationships. It contains broad and general knowledge harvested with little manual intervention, and this is the reason it was used for this application over a knowledge base like Freebase, which has significantly more manual intervention. Due to the limitations of automated extraction, ReVerb has more noise, which includes phenomena like repeated entities for the same actual entity. At the same time, however, it offers greater coverage in terms of more number of relation types.

Traning Signal: The traning signal which is used is two fold. One, questions are created using basic NLP from the knowledge base used in the first place. For instance, Triple (e.Barack Obama, r.be born in, e.Honolulu) generates the questions 1) Where was Barack Obama born ? (Honululu) and 2) Who born in Honululu? (Barack Obama). A more detailed list of question formats can be found in 3.3. Along with this, labelled question paraphrase pairs from the WikiAnswers corpus are used for training (There are 18M such question paraphrase pairs)

Some of the important challenges in formulating this solution to this setting

$$\forall i,\ \forall t' \neq t_i,\quad \boldsymbol{f}(q_i)^\top \boldsymbol{g}(t_i) > 0.1 + \boldsymbol{f}(q_i)^\top \boldsymbol{g}(t')$$

Figure 3.4: Classification objective function for open question answering

are

- **Lexical Variability** is a major issue (over syntax, because syntax is not considered). This refers to the fact the training set used is not a good model for language for the set of questions. This is partially dealt with by using question paraphrases from the WikiAnswers database during training
- The questions generated as such might have **over simplified syntax**
- Some **semantically incorrect** tuples might be generated as a result of this process. For instance, a question like "who does e r?" can be chosen for which the answer is not a person (tuple : [e.Maven, r.be orbit, e.Mars])

Given this formulation and training data, we would want to learn embedding f (for queries) and embedding g (for tuples) such that, the condition in 3.4 holds. Here, t_i is the valid answer to the query q_i. That is, we want the triple that labels a given question to be scored higher than others with a margin 0.1. This is alternated (multi tasked) with training of the model for questions by training on pairs of paraphrases of questions from WikiAnswers.

3.2.2 Extensions to Using Weakly Supervised Embedding Models

The model mentioned, therefore, essentially learns feature representations for each word in a triple, as well as each word in a query. The feature representation of an entity/triple is then the sum of the individual word vectors for words present in the entity/triple. There are extensions to this work in two forms in [16]

- A path from the entity mentioned in the question to the candidate answer entity is encoded in the feature vector. This is done upto a fixed number of hops. That is, the relation types on the path are encoded, and not the entities
- The entire subgraph of entities connected to the candidate answer entity is included in the feature representation (upto a given coverage). That is, both entities as well as relation types are encoded

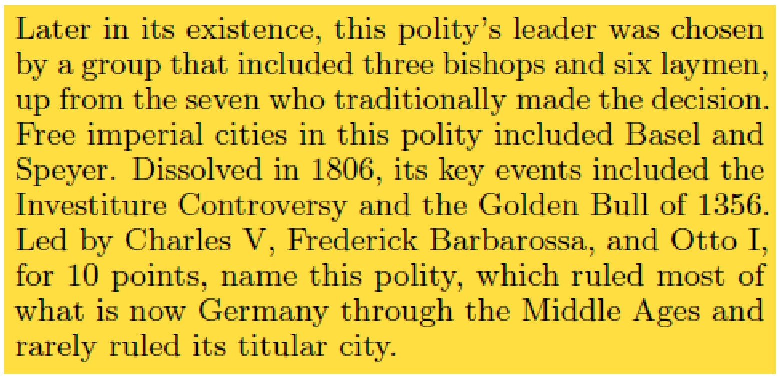

Later in its existence, this polity's leader was chosen by a group that included three bishops and six laymen, up from the seven who traditionally made the decision. Free imperial cities in this polity included Basel and Speyer. Dissolved in 1806, its key events included the Investiture Controversy and the Golden Bull of 1356. Led by Charles V, Frederick Barbarossa, and Otto I, for 10 points, name this polity, which ruled most of what is now Germany through the Middle Ages and rarely ruled its titular city.

Figure 3.5: A description of The Holy Roman Empire

3.2.3 A Neural Network for Factoid Question Answering

We now present a Natural Language Processing based approach to factoid question answering. The query is as follows: Given a description, identify the person, place or thing being discussed. For reference, see 3.5

In [17], an NLP approach is considered. The idea is to learn a vector space representation of a sentence using a DT-RNN (Dependency Tree - Recurrent Neural Network), where the structure of the neural network for a given sentence (query) is the dependency parse tree of the query. Following learning of the vector space representation, discriminative classification techniques are used, as answer embeddings are learnt in the same space. This allows us to model the relationships between answers and questions, leading to an inherently better model. The answer ”closest” to a query is chosen during inference.

Model

Each node n in the parse tree for a particular sentence is associated with a word w, a word vector x_w, and hidden vector h_n of the same dimension as the word vectors. For internal nodes, this vector is a phrase-level representation, while at leaf nodes, it is the word vector mapped to the hidden space. The DT-RNN has to combine the current node's vector with its children's hidden vectors to form h_n. This process continues recursively upto the root. A sample dependency tree can be seen in 3.7.
Given a parse tree, we first compute the leaf representations. For instance,

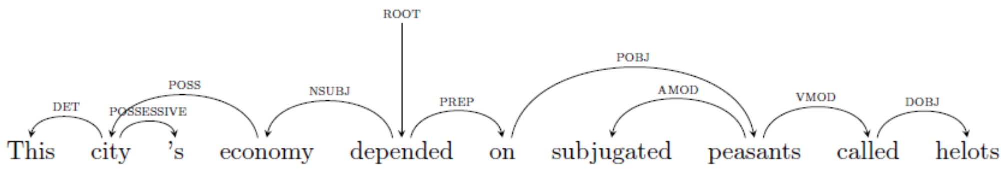

ROOT
POBJ
POSS
AMOD
NSUBJ
VMOD
DET
PREP
DOBJ
POSSESSIVE
This city 's economy depended on subjugated peasants called helots

Figure 3.6: Dependency Parse Tree of a Sentence

for 3.7, the hidden representation for h_{helots} is $h_{helots} = f(W_v.x_{helots} + b)$, where W_v transforms a word vector to the hidden space, f is a non linear activation function (eg. tanh) and b is a bias term. After the leaves, the interior nodes are processed. Continuing from "helots" to its parent "called, $h_{called} = f(W_{OBJ}.h_{helots} + W_v.x_{called} + b)$, where W_{OBJ} is an operator matrix to be learnt for each type of dependency that is considered. Finally, this process is continued to the root, where $h_{depended} = f(W_{NSUBJ}.h_{economy} + W_{PREP}.h_{on} + W_v.x_{depended} + b)$.
Intuitively, we want to encourage the vectors of question sentences to be near their correct answers and far away from incorrect ones. This goal is accomplished by using a contrastive max-margin objective function, the details of which are skipped. The gradient of the objective function as a result of the objective function is computed using backpropagation through structure.

3.3 Near Queries

We now look at the process of answering near queries. Near queries are of the following format:

- *Find Persons near Web Search*

. Here, the entity type being sought is "Person", and "Web Search" are the near key words. We allow the formulation of queries which can seek multiple entities as answers, and can specify relationship constraints between these entities. An example of such a query which specifies relationship constraints is

- *Find companies and their founders, where companies are in Silicon valley and founders are Stanford graduates*

For this query, the entity types being sought are companies and persons, where "Silicon Valley" are the near keywords for "companies" and "Stanford graduates" are the near keywords for "persons", and the person being sought should be the "founder" of the company (this is a relationship constraint).

3.3.1 BANKS-ERQ

In [18], the near query model for entity and entity relationship queries is assumed, and a model is presented to answer such queries. An integrated in-memory graph is presented to incorporate both structured and unstructured information. Each Wikipedia page/document represents an entity, which is the basic unit of search in this model. There are *category nodes* (representing Wikipedia categories), and there are *entity nodes*, representing all other Wikipedia pages). Labels are associated with edges to identify the edge type; the different types of edges in the graph are

- Document to entity edges, which link from a document to entities referenced in the document. The token offset is stored as part of the edge
- Document to category edges, denoting the type of an entity
- Category to category hierarchy denoting edges

Queries are expected in the following format: **find** C **near** (K), where C is one ore more keywords specifying the target entity type for the answer, and K is a set of keywords; For example **find** films **near** (directed "martin scorsese" "robert de niro"). Here the keyword films gives the type information C, and the set K is equal to {directed, martin scorsese, robert de niro} For multiple queries, OR semantics are used to answer the query.

Scoring Model

The idea of activation spreading is used to score the result of an entity.
Activation Spreading: Activation spreading is initiated from the nodes containing keywords, and spreads activation to neighbouring nodes.

- The initial activation from a given keyword is in proportion to the node prestige (PageRank) of each such node. Nodes that receive the maximum activation form the results of the near query.
- Each node retains part of its incoming activation, and spreads the remaining to its neighbours. For this purpose, during the initial stages of activation spreading, activation is spread to the neighbouring nodes based on the proximity of the match between keywords K and the referred entity in the document.
- Activation received from multiple sources is combined using a combining function

Activation spreading continues until the change in activation falls below a pre specified threshold.

Category Relevance The answers to a keyword query must satisfy the target type information specified in the query. The categories are indexed separately as documents. The relevance to each category is used, and the maximum of this over all categories lets us choose the most relevant category to a query.

Combining Activation and Category Relevance This is combined as: $score(e) = actScore(e) * \eta + relScore(e) * (1 - \eta)$ where $score(e)$ is the score of entity e, η is a tuned parameter, $actScore(e)$ and $relScore(e)$ refer to the activation spreading and category relevance scores of e, respectively.

Relation Predicate Scoring: Consider a relation predicate tuple $< e_1, e_2, ..., e_n >$, and the set of occurences O of the entities in the answer tuple and the keywords corresponding to the predicate appearing together in the text. The score for the relation predicate p is a negative exponential in the square of *TokenSpan(o)*, where *TokenSpan(o)* is the number of tokens present in the minimal scope in o covering all the entities and keywords.

Aggregation The aggregate score is given by $\Pi_{p \epsilon selPreds} score_p * \Pi_{p \epsilon relPreds} score_p^{\gamma}$, where $selPreds$ and $relPreds$ denote the selection and relation predicates, and γ is a tuned parameter.

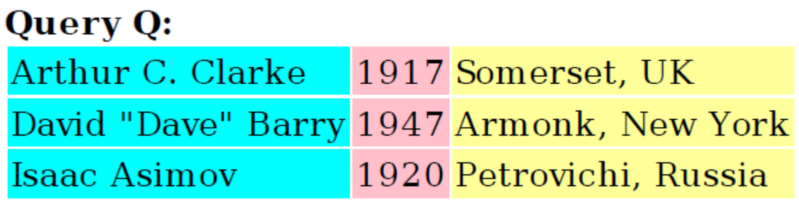
3.3.2 Object Rank

In [19], the issue of keyword specific pagerank for an object in a hyperlink graph is addressed. The ideas of pagerank are extended to find the keyword specific stationary pagerank vector for each keyword.

In this work, the domain being considered is the bibliography domain, where a subgraph of the DBLP graph is modelled. It is important to note that, regarding space requirements, the number of keywords of a database is typically limited, and less than the number of users in a personalized search system. Furthermore, this system does not store nodes with ObjectRank below a threshold value (tuned), which offers a space versus precision tradeoff. Nevertheless, the issue of the **index blowing up** remains.

The basic algorithm is the same as the random surfer model that is used in page rank.

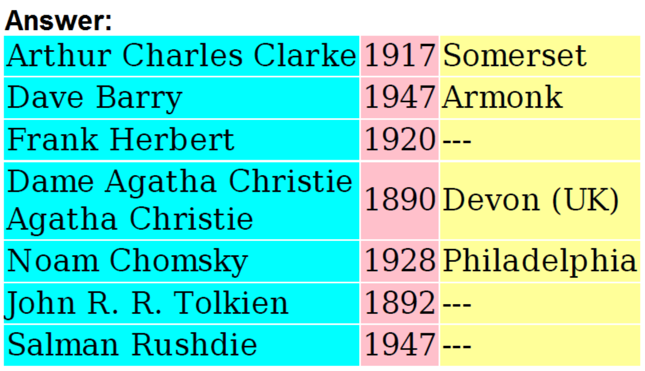
- There is a random surfer who starts from a random node (chosen uniformly at random) of the link graph



Query Q:

Arthur C. Clarke	1917	Somerset, UK
David "Dave" Barry	1947	Armonk, New York
Isaac Asimov	1920	Petrovichi, Russia

Figure 3.7: A sample table augmentation query



Answer:

Arthur Charles Clarke	1917	Somerset
Dave Barry	1947	Armonk
Frank Herbert	1920	---
Dame Agatha Christie Agatha Christie	1890	Devon (UK)
Noam Chomsky	1928	Philadelphia
John R. R. Tolkien	1892	---
Salman Rushdie	1947	---

Figure 3.8: A sample table augmentation query result

- At each step, he follows a hyperlink on the page with equal probability
- With probability α, the surfer skips back randomly to any node in the graph, because he gets bored.

The only difference in ObjectRank and PageRank is that the start set of nodes and the set of nodes to which the random surfer teleports is just the set of nodes on which that particular keyword appears.

3.4 Table Augmentation Queries

A table augmentation query is one where a query is represented by a few sample rows of a table. The objective is to retrieve similar rows from a corpus. A sample query is shown in 3.8 and a sample answer is shown in 3.9

3.4.1 WWT

In [20], an unsupervised end to end approach is described, which assembles rows of a table from a few example rows by harnessing the huge corpus of

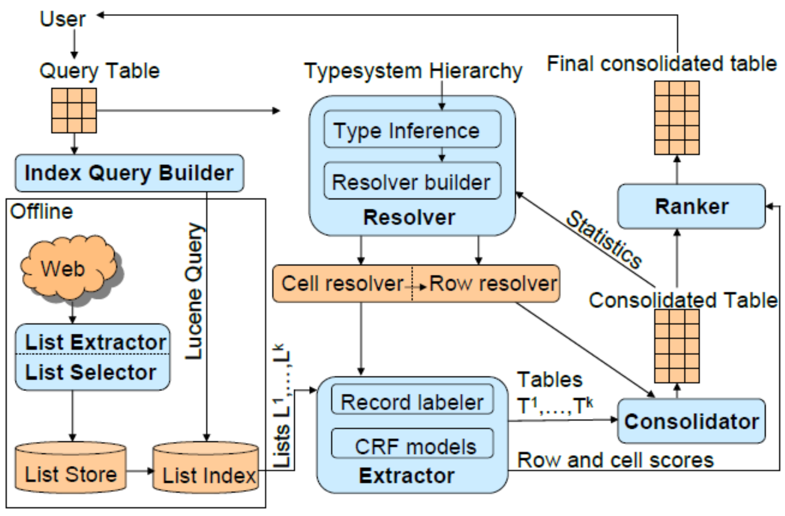


Figure 3.9: WWT block diagram

information rich but unstructured lists on the web. This can quite simply be extended to tables. The key challenges in this task include construction of new rows from very few examples, as there is an abundance of noisy and irrelevant lists that swamp the consolidation and ranking of rows.
This is also applicable to a site like Freebase or Wikipedia, which wishes to build a table of multi attribute records belonging to a new topic, starting with a few seed tuples.
The system architecture of WWT is shown in 3.9.

List Extraction

The first step in this process is to extract lists from the web which are useful. This is done using a set of heuristics.

List Index

Each list is treated as a document and indexed. In response to a table query, the index is probed by creating a bag of words query from the contents of the table. The top-K matching lists based on Lucene's inbuilt ranking criteria are returned

Resolver

The resolver takes a query table and infers the type of each column. This is based on a user configurable type hierarchy. The type information along

with the table contents is then used to build a cell and row resolver - a row resolver takes two rows and returns information about how similar they are. This module is required for use later. The cell resolvers are building blocks for the row resolver.
Here, the resolvers for a particular type are obtained from a type hierarchy - one which has a resolver built in for each type listed. This resolver is then customized to a given table of values.

Extractor

The extractor module takes the source lists, and trains extraction models on those lists. However, this implies that labelled training data is required. Therefore, the sample tuples which are given as a part the query are recognized in the current list, and these samples are used to train the model for classification. To improve this process, other tuples which have already been found can also be used. This is now used to train a CRF model to extract more such tuples from the table. For each list, a separate CRF model is trained, which is then used to extract tuples from that list, based on the best segmentation offered by the model. The extractor also attaches a confidence score with each row and cell entry for later use by the ranker.

Consolidator

A consolidator takes a set of tables output by the extractor, and merges all such tables into one table. The advantages of this are manifold:

- Certain items are extracted multiple times from multiple times. If consolidation is performed well, then these items do not have to be checked separately during ranking

- Certain items are not extracted with high confidence, but with multiple evidences, this issue is solved

- Multiple instances of extracted tuples can have incomplete information. They can be combined to form a complete tuple.

A builing block of the consolidator is the row and cell resolver which was built earlier. Once such resolvers a finalized, in order to merge tables, we look at them pairwise. Among them, since we trivially delete duplicates in the same table, we can find tuples most similar to other tables between pairs of tables based on a weighted maximum matching formulation, where the weights are the measures of similarities between rows. As a result of this process, similar rows are merged.

Ranking

Following consolidation, ranking of the tuples is performed. The major signals used in this process are.

- The number of evidences for a particular tuple
- The importance of certain columns over certain other columns

Limitation: One of the important limitations of this process in ranking is that the quality (or a pagerank like measure) of the node is not used. This might be explored, because information from high pagerank sources might be naturally more trustworthy, and can be given more importance.

3.5 Quantity Queries

The following are examples of Quantity Consensus Queries

- *Driving time from Paris to Nice*
- *Battery life of Lenovo X300*
- *Net worth of Bill Gates*

A key point to note for each of these queries is that the answer to these may be a probabiliy distribution, instead of a pointed value. The goal is to answer with such a relevant probability distribution. This application therefore, requires a departure from typical entity search, and modifications for the scenario of quantity queries are explored.

3.5.1 QCQ

In [21], the following type of queries are considere: *+giraffe, +height; foot* This query indicates that the words *giraffe* and *height* must be present in the context snippet of the mention of the desired quantity type *foot*, where the + denotes compulsory presence.
The pipeline for the QCQ system prototype can be seen in 3.10.
What follows is an overview of the pipeline:

- Fetching relevant snippets: The first step in the pipeline is to fetch relevant pages from the web which are relevant to the quantity query. For this, Web search APIs are used. However, one cannot directly look for documents based on the quantity type being sought. Hence, this

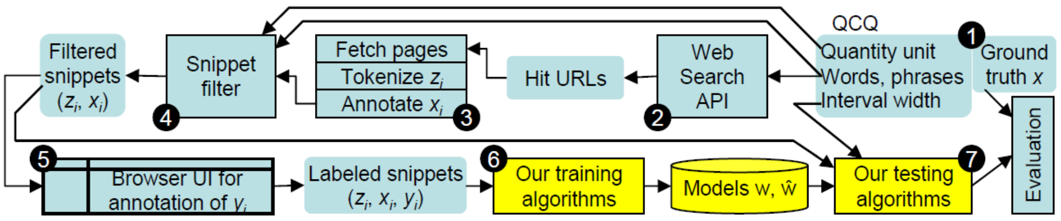


Figure 3.10: QCQ pipeline

is essentially a two step process in which words, phrases and units are sent to the search engine. In response, URLs are fetched, tokenized, and quantities are annotated. They are filtered heuristically. A pre existing annotator is used for numeric value annotation. This completes the process until step 4 in 3.10

- **Training Data**: For the purpose of model training, data is collected for snippet relevance for selected Quantity Consensus Queries

- **Feature vector Design:** For every snippet, we need to classify it according to its relevance. We therefore use a discriminative approach for this. For a snippet surrounding an annotated quantity, we design a feature vector based on -
  - Standard ranking features such as Term Frequency (TF) and Inverse Document Frequency (IDF) of the query words also found in the context of the annotated snippet
  - Jaccard similarity between query and snippet tokens
  - Proximity features, such as maximum proximity of any query token in the snippet, proximity to the rarest and most common query token.

- **Snippet Relevance Formulation**: Numerous ways of evaluating snippet relevance are explored. Among them, the notable are
  - Standard pairwise RankSVM, where the score $w^T z_i$ is used for ranking snippets. The formulation is given in 3.11. Here, Σ_{ij} ζij upper bounds the number of pair preferences violated and C balances this with $|w|$.
  - It is seen that after the RankSVM formulation, almost all snippets turn out to be relevant for a particular range of the value of the type being sought. This is a valuable signal which can be used for

$$\min_{w,\xi} \frac{1}{2} w^\top w + C \sum_{i:y_i=1} \sum_{j:y_j=-1} \xi_{ij} \quad \text{subject to}$$

$$\forall i \text{ s.t. } y_i = 1, \forall j \text{ s.t. } y_j = -1, \begin{cases} \xi_{ij} \geq 0 \\ w^\top z_i + \xi_{ij} \geq w^\top z_j + 1 \end{cases}$$

Figure 3.11: RankSVM formulation for snippet relevance

snippet relevance, as we may classify only snippets lying in such a band as relevant

  - Laplacian Smoothing - where we explicitly use the annotated values for snippet relevance calculation. Here, the idea is to smooth relevance (as first calculated from the RankSVM formulation) across snippets with the same annotated value.

- **Interval Relevance Formulation**: Following this, an approach for interval relevance is proposed, instead of snippet relevance. The value $w^T z_i$ is used in aggregates as sum of this value for all snippets in this interval (where w comes from the RankSVM formulation), pair wise difference with points lying outside this interval, and so on. Then, all intervals are enumerated, and the best of these chosen as answer.

- **Learning to Rank Intervals**: Following this, a discriminative approach to directly rank intervals is proposed. For this, the feature vector design includes features as

  - Whether all snippets contain some query word; or the rarest or most common query word
  - Number of distinct quantities mentioned
  - Features which come from the interval relevance formulation above
  - An aggregation of individual features from snippets

  Classification is again performed using a RankSVM formulation, where training relevance for an interval is a ratio of the number of relevant snippets to the total number of snippets in that interval. We skip details.

3.5.2 Quantity Queries on Web Tables

In [22], quantity consensus queries are answered using information extracted from tables available on the web. The system pipeline can be seen in 3.12 We now briefly explain various components of this pipeline.

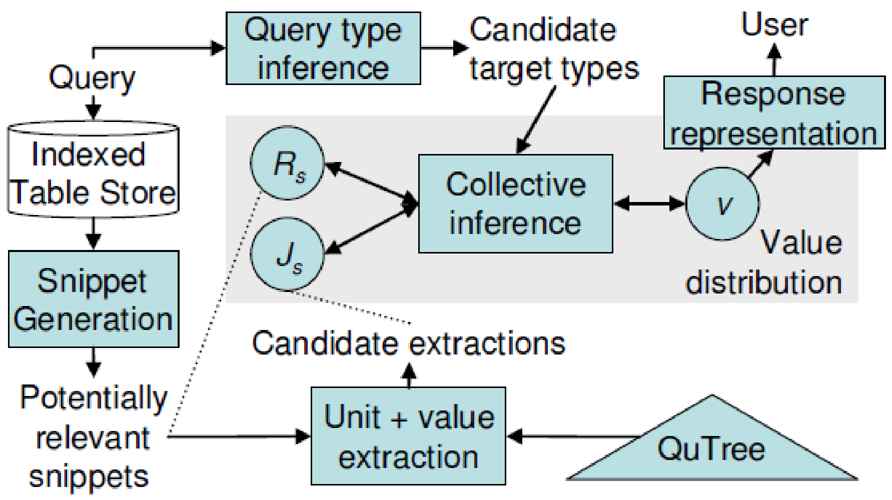


Figure 3.12: Quantity Consensus Queries on Tables - System Pipeline

- The first step is to extract relevant tables from a pre indexed corpus
- At the same time, from the query, the type being sought is inferred. This step involves usage of a type and unit hierarchy (called QuTree), which has been built specifically for this work.
- Once relevant snippets are obtained, the candidate units and values from such a table are obtained. These candidate values are annotated with a unit in QuTree, which also gives its quantity type. Both these tasks are highly challenging. For this purpose, therefore, a probabilistic context free grammar is used (PCFG).
- Once these candidates are individually extracted, they are used for collective inference, in a procedure that simultaneously estimates the relevance of each such snippet and each of its possible extractions, as well as the target quantity type. This takes into account facets like:
  - Top scoring response quantity types are preferred
  - Individually relevant snippets are preferred
  - The most relevant extractions from snippets are preferred
  - The answer is a value distribution. It is ensured that the density around values with more evidence is high
  - The distribution is normalized, to prevent overfitting

The details of the algorithm are skipped.

- The answer representation is in the form of a distribution, so is the internal representation. It is not inferred point wise. For the internal representation, non-parametric distributions such as a kernel density function is used, and for the final representation, a uniform density mixture may be used.

Chapter 4

Future Work

In this work, we have surveyed numerous techniques for structured information extraction, as well as query answering, with a heavier focus on the latter. Over the course of this project, we have deliberated on numerous possible directions of future work. We would now like to present a particular possible direction of future work which is of interest to us.

4.1 Statement

Given:

- A search query Q
- A set of candidate answers A, and
- A person's identity I

Rank members in A according to the specific interests of this person as given by his identity I as an answer to the query Q.

Note that we want to solve this issue in the specific context of structured search, as compared to generic personalised web search.

4.2 Discussion

This statement is a very broad specification of what we want to solve. There are numerous questions that need answering. For simplicity, let us consider the question: "Given a person's profile on DBLP (and/or Google Scholar), sort a list of accepted papers for a given conference (and/or year) based on the person's personal interest". Some points that may be noted in this context are:

- **Personal versus Global:** What is 'personal'? What is 'global'? For instance, in the current DBLP graph, authors are already modelled. What aspects of their persona need separate modelling?
- **Incomplete Information:** How do we handle incomplete information/noise in these knowledge bases, due to privacy reasons or information extraction inconsistencies?
- **Knowledge Sources:** What knowledge sources need to be used? If multiple repositories like DBLP and Google Scholar are used, we need to create a consolidated knowledge graph of both these resources. How do we do that? What is the model, and the extraction techniques that we use here?
- **Preprocessing vs Query Time:** Do we extract personal information for a user before hand? Or do we do it on the fly? This question is raised because we might target a large number of users, and it might not be possible to index information about each one of them before hand. What extraction techniques do we use here?
- **Ranking:** How does one design ranking algorithms for this setting? Do we view this as *connecting* a personal graph to a global graph for a search query and a user, and then use techniques like activation spreading? Can suitable topic models be used for this purpose? Should we take a supervised or an unsupervised approach?

We look forward to working in this direction in the second half of this project.

Chapter 5

Conclusion

In this work, we presented a literature survey of some of the questions relevant to structured search, as well as solution techniques for the same. Specifically, we presented

- An overview of the relevant questions pertaining to search over structured data
- Information Extraction questions and proposed solutions in the literature
- Query Answering over many different types of queries and solution techniques
- A proposed plan for future work on personalised search over structured data

We ended with a proposal of future work, which we wish to undertake in the second part of this project. We look forward to further investigating techniques in structured search and applying some of these to our proposed future work.

Bibliography

[1] Indrajit Bhattacharya and Lise Getoor. "A Latent Dirichlet Model for Unsupervised Entity Resolution." In: SIAM.

[2] Sayali Kulkarni et al. "Collective annotation of Wikipedia entities in web text". In: *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining.* ACM. 2009, pp. 457–466.

[3] Girija Limaye, Sunita Sarawagi, and Soumen Chakrabarti. "Annotating and searching web tables using entities, types and relationships". In: *Proceedings of the VLDB Endowment* 3.1-2 (2010), pp. 1338–1347.

[4] Soumen Chakrabarti et al. "Compressed Data Structures for Annotated Web Search". In: *Proceedings of the 21st International Conference on World Wide Web.* WWW '12. Lyon, France: ACM, 2012, pp. 121–130. ISBN: 978-1-4503-1229-5. DOI: `10.1145/2187836.2187854`. URL: `http://doi.acm.org/10.1145/2187836.2187854`.

[5] Nicholas Kushmerick. "Wrapper induction for information extraction". PhD thesis. University of Washington, 1997.

[6] Ion Muslea, Steve Minton, and Craig Knoblock. "A hierarchical approach to wrapper induction". In: *Proceedings of the third annual conference on Autonomous Agents.* ACM. 1999, pp. 190–197.

[7] Arvind Arasu and Hector Garcia-Molina. "Extracting structured data from web pages". In: *Proceedings of the 2003 ACM SIGMOD international conference on Management of data.* ACM. 2003, pp. 337–348.

[8] Chia-Hui Chang and Shao-Chen Lui. "IEPAD: information extraction based on pattern discovery". In: *Proceedings of the 10th international conference on World Wide Web.* ACM. 2001, pp. 681–688.

[9] Yanhong Zhai and Bing Liu. "Web data extraction based on partial tree alignment". In: *Proceedings of the 14th international conference on World Wide Web.* ACM. 2005, pp. 76–85.

[10] Sergey Brin. “Extracting patterns and relations from the world wide web”. In: *The World Wide Web and Databases.* Springer, 1999, pp. 172–183.

[11] Eugene Agichtein and Luis Gravano. “Snowball: Extracting relations from large plain-text collections”. In: *Proceedings of the fifth ACM conference on Digital libraries.* ACM. 2000, pp. 85–94.

[12] Razvan C Bunescu and Raymond J Mooney. “A shortest path dependency kernel for relation extraction”. In: *Proceedings of the conference on Human Language Technology and Empirical Methods in Natural Language Processing.* Association for Computational Linguistics. 2005, pp. 724–731.

[13] Uma Sawant and Soumen Chakrabarti. “Learning Joint Query Interpretation and Response Ranking”. In: *Proceedings of the 22Nd International Conference on World Wide Web.* WWW ’13. Rio de Janeiro, Brazil: International World Wide Web Conferences Steering Committee, 2013, pp. 1099–1110. ISBN: 978-1-4503-2035-1. URL: `http://dl.acm.org/citation.cfm?id=2488388.2488484`.

[14] Tao Cheng, Xifeng Yan, and Kevin Chen-Chuan Chang. “EntityRank: Searching Entities Directly and Holistically”. In: *Proceedings of the 33rd International Conference on Very Large Data Bases.* VLDB ’07. Vienna, Austria: VLDB Endowment, 2007, pp. 387–398. ISBN: 978-1-59593-649-3. URL: `http://dl.acm.org/citation.cfm?id=1325851.1325898`.

[15] Antoine Bordes, Jason Weston, and Nicolas Usunier. “Open Question Answering with Weakly Supervised Embedding Models”. In: *CoRR* abs/1404.4326 (2014). URL: `http://arxiv.org/abs/1404.4326`.

[16] Antoine Bordes, Sumit Chopra, and Jason Weston. “Question Answering with Subgraph Embeddings”. In: *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, EMNLP 2014, October 25-29, 2014, Doha, Qatar, A meeting of SIGDAT, a Special Interest Group of the ACL.* 2014, pp. 615–620. URL: `http://aclweb.org/anthology/D/D14/D14-1067.pdf`.

[17] Mohit Iyyer et al. “A Neural Network for Factoid Question Answering over Paragraphs”. In: *Empirical Methods in Natural Language Processing.* Doha, Qatar, 2014.

[18] Ankur Agrawal et al. "Entity Ranking and Relationship Queries using an Extended Graph Model". In: *Proceedings of the 18th International Conference on Management of Data, COMAD 2012, 2012, Pune, India.* 2012, pp. 80–91. URL: `http://comad.in/comad2012/pdf/agrawal.pdf`.

[19] Andrey Balmin, Vagelis Hristidis, and Yannis Papakonstantinou. "Objectrank: Authority-based keyword search in databases". In: *Proceedings of the Thirtieth international conference on Very large data bases-Volume 30.* VLDB Endowment. 2004, pp. 564–575.

[20] Rahul Gupta and Sunita Sarawagi. "Answering table augmentation queries from unstructured lists on the web". In: *Proceedings of the VLDB Endowment* 2.1 (2009), pp. 289–300.

[21] Somnath Banerjee, Soumen Chakrabarti, and Ganesh Ramakrishnan. "Learning to Rank for Quantity Consensus Queries". In: *Proceedings of the 32Nd International ACM SIGIR Conference on Research and Development in Information Retrieval.* SIGIR '09. Boston, MA, USA: ACM, 2009, pp. 243–250. ISBN: 978-1-60558-483-6. DOI: `10.1145/1571941.1571985`. URL: `http://doi.acm.org/10.1145/1571941.1571985`.

[22] Sunita Sarawagi and Soumen Chakrabarti. "Open-domain Quantity Queries on Web Tables: Annotation, Response, and Consensus Models". In: *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining.* KDD '14. New York, New York, USA: ACM, 2014, pp. 711–720. ISBN: 978-1-4503-2956-9. DOI: `10.1145/2623330.2623749`. URL: `http://doi.acm.org/10.1145/2623330.2623749`.