Entity Relationship Keyword Search on Semi Structured Web Data

B.Tech Project Stage 1 Report

D V Deepankar Reddy
Roll No : 09005060

under the guidance of

Prof. S. Sudarshan

Department of Computer Science and Engineering
Indian Institute of Technology, Bombay

November 2012

Abstract

KeyWord Search now a days has been one of the most important methods of Information Retrival, with the advent of search engines like Google have revolutionized the information availablilty on the web, but the traditional keyword search has a drawback that is that it returns pages in the web as the results, then the user has to go through these pages to get to know the bit of information he was looking.

But there have been recent efforts to change this, it has been observed nearly 40% of the searches are aimed to get a single entity as the result. So the solution to above problem is to get back entities to the keyword search rather than pages and while returning the entities aggregate the information that is spread across various sources. Our main focus lies on the first part of the problem which is to return a single entity as the result to a given keyword search. But still there have not been any good systems to tackle this until date on a very large scale (web data) and on a very wide range of queries.

In this work, we mainly focus on two already existing systems *WikiBanksERQ* and *WikiCSAWERQ* which are developed by our seniors. We try to improve the results from the above two systems by thoroghly explaining the problems the above two systems are facing and giving our own ideas on the ways these problems can be tackled

Chapter 1

Introduction

KeyWord Queries which return Entities can be varying complexity, but the most of Entity Queries (KeyWord Queries which return Entities as results) can be classified or inferred from the following two types of queries

1. *Entity Near Queries* : These queries are of the form find Entites of *Category* : *Keyword1* near *KeyWord2*.
   For eg : *Countries in Europe*, here the translation would be *Category:Country* and *near:Europe*. Here the main problem is to first the countries in the web pages and find the evidence for that country near the keyword Europe

2. *Entity Relationship Queries* : These queries consists of two or more Entity Near Queries along with an additional relation keyword specifying the relation between the entities in question. For eg . *Find companies in Silicon Valley whose founder were Stanford University Gradutes.* In above example, if we expand it into q sor t of query language we get

   ```
   FROM CITY x, COUNTRY y
   WHERE y:["Europe"]
   AND x,y:["capital"]
   ```

   In the above example in addition to executing the two near queries we also have to find the evidences for both of them are related by the relationship keyword, finding the evidences for these and correctly interpreting the evidences present is the main challenging task in the entity realationship queries

There are lot of other problems associated with these type of queries, user will not know the structure of the queries as discussed in the above example, and also it is impossible to make strucuted query since it will need the knowledge of internal representation of data etc, and also each user has his own way of querying and hence it is difficult to standardize these kind of tasks.

Another problem associated is that extraction of relations or finding the evidences for relations, since most of the relationship keywords have different meaning according to different contexts it is difficult to find out the real lemma behind the user query. In the case of finding evidences the problems are to correctly recognize false positives that boost some wrong results, one solution to tackle this problem is to pre extract all the meaningful relations but in this only few relationships are extracted leading to decrease of the scope of queries. Recently the big search engines have also recognized these problems and have created relation ship extractors such as Google Squared[x] and Yahoo Correlator[x].

Few recent works in this area include systems like EntityEngine, WikiBanksERQ, CSAW system, Open Information Extraction(Open IE). EntityEngine proposes the formal structure of the entity-relationship query and we will be using the same structure to describe entity-relationship queries. WikiBanksERQ models Wikipedia data as graph and uses this for answering *near* and *Entity-Relationsip Queries*(ERQ) by doing BANKS sort of search on this graph. Open IE doesn't rely on fixed relations i.e one doesn't need to specify target relations in advance and extracts new binary relations using Reverb on web scale. CSAW system has large number of entities annotated in Web crawled corpus and uses the same for returning entities with score to queries.

This report is organized as follows. Chapter 2 describes the Literature survey we have done in the area. In Chapter 3 we will describe *WikiBanksERQ* . In Chapter 4 we will descirbe *"WikiCSAWERQ"* which is a Web Scale Entity Relation Search Engine, wiki in the name is due it takes the help of wikipedia in scoring to boost it results. The Chapter 5 will describe the problems these systems are encountering and suggests some possible upgrades.

Chapter 2

Literature Survey of Entity Query systems

Chapter 3

WikiBanksERQ

In this system we mainly consider wikipedia as the single source of information, wikipedia is preprocessed into a graph.In this system most of the entity is supposed to have a wikipedia page and generally this *homepage* is returned as a result of the queries. The Wikipedia is already annotated so that any ambigutiy between the entities are removed (for eg: Gandhi word in a text can refer to many things and its meaning generally depends on the context). For better results this system uses three position based feautures so that it can assess the quality of the evidence. The first is *proximity* that is the distance between the entity and the keyword. The second is *Ordering Pattern* of the evidence. The patterns which occur more with a keyword (mostly relationship) are considered to be of greater weightage. The third is *Mutual exclusion* when there is a collision of the evidences for different entities in the same text, the evidences are deemed ineffective.

3.1 DataModel

The basic datamodel is a Graph $G = (V, E)$ where V is a set of vertices and the E is the set of edges between the above vertices, vertices represnt the entites and documents where as the edges generally represent the evidence. The graph is a multigraph as there can be multiple evidences for the same pair of vertices(entites). This size of the graph is small enough to fit into the memory of the computer.

Vertices are modelled as documents with text descriptions and a set of pairs of the form $(term, offset)$, where term is the keyword and offset is the position of the term from the start of the document. If the vertex does not represent any document then offsets of the terms present in the text will be all zero. An example of non-document vertex can be the category vertices

In the Edge Set E, all the edges are directed and are the hyperlink from one document to the another and each edge has an offset which is the offset within the document which the source vertex represents. Edges can also have associated labels, which can be used to categorize the edges in various ways, for eg: the evidence edges, the category edges where the category represents the hierachy or the belongedness of the entity.

Category hierachy is followed using YAGO ontology. Wikipedia data is indexed using *Lucene*. Mapping from the keywords to nodes is stored in a full text lucene index along the offset information. We only store the nodes and the edges.

3.2 Query Execution

Queries are executed in the following way

1. First the near queries are executed and the all the potential Entites are found for all the near queries.
2. Now For each of possible tuple of entities we try to find the evidence and using the evidence we will boost the score of the tuple.

So we will discuss the near query execution in the detail, following we will discuss the execution of the ERQ queries in the above method.

3.2.1 NearQueryModel and Execution

A near query can be simply described as

```
find Entities of (Category: C) near K
```

Where C is the set of categories or entity types and K is a set of near keywords.The following datastructures are used in the above query execution:

- *nearKeywordList*: The set of the keywords after near provided the user.
- *nearKeywordOriginSet*: The set of document pages that contain the keywords in the *nearKeywordList*. Here we can use two types of semantics either the AND semantics in which all the keywords must be present in the document or the OR semantics in which some of the keywords must be present in the document.
- *relevantCategorySet*:The set of relevant categories to which the target entity should belong to

The near queries are executed in the way similar to described in the banks[?]. The method is explained below

- Firstly near and categories keywords are combined and Lucene index is consulted to get all the relevant categories, this is the *relevantCategorySet*. Here the near Keywords are also combined because some times it helps in making the query easy. For ex : *Countries near europe where french is spoken* where category is country and near keywords are europe, french can be translated to a query which is *Countries of europe whare french is spoken* where category is Conutry of europe and near keywords is french.

- Using the Lucene index we get the documents which containing the Keywords in *nearKeywordList*, this set of documents is the *nearKeywordOriginSet* using one of the semantics described above. Intial score is given to these documents depending on the *Node Prestige* which is the pagerank score of the document and *Lucene Score.*
- Now traverse the outlinks of the *nearKeywordOriginSet* and set of Entities near the origin set are found, while traversing the score (activation) is spread to neighbours through outlinks based on the proximity of the link.
- Now select the entities which belong to a category in *relevantCateogrySet*, and use the score is calculated using the activation spread, node prestige, Lucene score and then order the entites by score

3.2.2 Scoring for NearQueries

The main advantage for this system is the method of activation spreading explained in[?] because it can answer the queries where the transitivit is the key to answer.
For eg: in the query *universities near Nobel prize* you need find the persons near nobel prize and the university near that person.

Initial Activation

The score given to the origin nodes is the initial activation. It is based on the *NodePrestige* which is based on the number of outlinks for that node (page rank) and *LuceneScore* which depends on the number of keyword hits that are present in the document. It is natural to use the *LuceneScore* since if the document contains all the keywords then most proabably that document is the evidence document.
The two scores can be combined mutiplicatively:

$$[LuceneScore^{\lambda}] * [NodePrestige^{(1-\lambda)}] \tag{3.1}$$

or additively:

$$[LuceneScore * \lambda] + [NodePrestige * (1 - \lambda)] \tag{3.2}$$

Activation Spreading

This part mainly depends on the proximity of the outlinks along with a constant attenuation factor μ which allows only part of the activation to be spread to the neighbours. The amount of the activation spread is depends on the edge weights which are again dependent on the proximity.It can be taken that fraction spread is the proximity function.

Intuitively, if a keyword and a link to an entity occur in proximity in a document, we believe that the entity is related to the keyword; the closer the occurrences, the higher is the estimate of relevance of the entity to the keyword. We use this idea to define the amount of activation transferred to each of the entities linked with the document. The function to calculate the proximity of a link with a keyword must degrade as the distance between the keyword and link increases. Formally if the keyword occurs at point i, the link occurs in point j, then the proximity of the two nodes if calculated using Gaussian Kernel function is given by :

$$k(i, j) = exp[-(i - j)^2/2\mu^2] \tag{3.3}$$

Category Relevance and final Score

The entities that we found should also belong to the target type given by the user. Hence all the entities we found would given a score known as *relCatScore* based on the relevance of the category it belongs. This score for a category is found using lucene indexer where we keep an index for all the categories and keywords in them and now when we retrive the *relevantCategoryScore* we also get the *LuceneScore* and use this as teh *relCatScore*

After we get the all required scores we then combine them using either additive or mutliplicative semantics explained above, one example of them is shown below:

$$score(e) = actScore(e) * \eta + relCatScore(e) * (1 - \eta) \tag{3.4}$$

3.2.3 ERQModel

In this section we focus how to execute the erq on the above described system, the query model is same as the above but we will have more variables and some other keywords which relate the two varaibles we have. Each entity variable can be associated with zero or more predicates, there are mainly two types of predicates that we use in a entity relationship query:

- *Selection predicate*: It is for a single entity variable and it is more like *nearKeywordList* in near queries.
- *Relation predicate*: It relates two or more of the entity variables present in the query and there can be many of them in a given query

we consider an example to explain the execution of these queries:

```
SELECT x, y FROM CITY x, COUNTRY y  WHERE y:["Europe"] AND x,y:["capital"]
```

Now in the above example we have two variables x and y and relation predicate connecting these two is the *capital* and near keywords,categories are similar to near queries.

3.2.4 Execution of ERQ

The following are the steps in the execution of an ERQ query:

- First we execute the near query algorithm for selection predicates and find all the relevant entities for each variable

- Find the list of documents which contain atleast one entity for each variable present in the query(or involved in the relation predicate we are considering). These can be the potential evidences for the realtion predicate. These can be found using the inlinks of entity node. Among these consider only the pages that contain reference to the relation keyword we are conidering. This can be done using the Lucene index. This is repeated for all the relation predicates.

- Now go through each page in the list and perform cross product of all the entity list for variables to get the all possible answer tuples for this page, also note the offsets between the links these are used in score calculation and hence finally in ranking.

3.2.5 Scoring for ERQ

The overall score of a tuple is the score that is aggregated overall the predicates. The scores are weighted depending on whether the predicate is an selection predicate or relation predicate.

If the predicate is a selection predicate then the its score is exactly same as the one we calcualted for the near queries which depends on the *activation* and *relCatScore*

If the predicate is an relationship predicate then the score will depend on the number of evidences that support that relation between the entites and also we have to consider the proxmity in this case also. So in this case we will take *TokenSpan* in the evidence with in which all the entities and the relation predicate is present and the score should tend to zero for larger tokenspan and also this should aggregated over all evidences. so the score will be given by:

$$Score_p = \sum_{allevidences} exp[\frac{-(TokenSpan^2)}{2\sigma^2}] \tag{3.5}$$

Now the overall score for a tuple $\langle e_1, e_2, ..., e_n \rangle$ is given by the aggregation over all the predicates that are present in the erq

$$Score(\langle e_1, e_2, ..., e_n \rangle)) = \prod_{p \in selection\ predicates} Score_p * \prod_{p \in relation\ predicates} Score_p^{\gamma} \tag{3.6}$$

Here γ is the weight given to the relation predicate over selection predicate

3.3 Conclusion

The above system effectively answer most of the queries, but wikipedia is small when compared to the number of evidences present in the web. So there is an need for web scale ERQ engine. Also because wikipedia is small we were able to use the graph structure and fit it in memory, but in the WikiCSAWERQ we will see it is impossible to use the graph structure.

Chapter 4

WikiCSAWERQ