**Identifying Relationships for Entity Relationship Queries**

Submitted in partial fulfillment of the requirements

of the degree of

Bachelor of Engineering

by

Neha Tembe, Roll No. 12106B0026

Ahmed Sabeeh, Roll No. 12106B0035

Dhruv Kharwar, Roll No. 12106B0068

Under the Guidance of

Prof. Pankaj Vanwari



Department of Computer Engineering

Vidyalankar Institute of Technology, Wadala (E), Mumbai

2015-2016

**Project Report Approval for Bachelor of Engineering**

This project report entitled ***Identifying Relationships Using Entity Relationship Queries*** by Neha Tembe, Ahmed Sabeeh and Dhruv J. Kharwar is approved for the degree of ***Bachelor of Engineering*** in ***Computer Engineering***.

Examiners

1. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_
2. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Supervisors/ Guides

1. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_
2. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Head of Department

1. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Principal

1. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Date:

Place:

**Declaration**

We declare that this written submission represents our ideas in our own words and where others' ideas or words have been included, we have adequately cited and referenced the original sources. We also declare that we have adhered to all principles of academic honesty and integrity and have not misrepresented or fabricated or falsified any idea/data/fact/source in our submission. We understand that any violation of the above will be cause for disciplinary action by the Institute and can also evoke penal action from the sources which have thus not been properly cited or from whom proper permission has not been taken when needed.

|  |  |  |  |
| --- | --- | --- | --- |
|  | Name of the Student | Roll No. | Signature |
| 1. | Neha Tembe | 12106B0026 |  |
| 2. | Ahmed Sabeeh | 12106B0035 |  |
| 3. | Dhruv Kharwar | 12106B0068 |  |

Date:

**Abstract**

The information available on the web, with regard to certain entities is in latent, unstructured form. Hence, the main aim of our system, using information extraction (IE), is to understand the semantic equivalence of phrases and map them to a canonical form.

Our system will deal with the issue of comprehensive gathering and systematic organization of patterns for an open set of relations. The system can then be queried to show a relationship between entities. The system will be able to detect and disambiguate named entities in text and extract binary relations between entities based on patterns in textual or semi-structured contents.

This system can be used to organize a large number of relational patterns into sets of synonymous patterns and finally into a hierarchy of entities. It can boost information extraction and knowledge based population tasked by means of its repository of paraphrases for relations. It also enables advanced search over *“subject-predicate-object”*.

**Acknowledgement**

At the very outset of this project synopsis, we would like to extend our sincere and heartfelt thanks to all those who have helped the three of us with this endeavour. Without their active guidance, cooperation and encouragement we would not have made headway in this project. We are indebted to Prof. Pankaj Vanwari for his conscientious guidance, insights, input and encouragement in helping us complete this project.

We would also like to extend our gratitude to Vidyalankar Institute of Technology and the Department of Computer Engineering for providing us with the opportunity and means to complete our task.

Thanking you,

Neha Tembe

Ahmed Sabeeh

Dhruv J. Kharwar

**Table of Contents**

|  |  |  |
| --- | --- | --- |
| **Sr. No.** | **Page Title** | **Page No.** |
| 1 | **Project Overview** | 1 |
| 2 | **Introduction and Motivation** | 2 |
|  | **2.1** Theory behind the concept of the project |  |
|  | **2.2** Problem definition |  |
|  | **2.3** Need for the system |  |
| 3 | **Analysis and Design** | 4 |
|  | **3.1** Software development |  |
|  | **3.2** Flow of the project |  |
|  | **3.2.1** Preliminary survey |  |
|  | **3.2.2** Feasibility study |  |
|  | **3.2.3** Cost analysis |  |
|  | **3.2.4** Process model |  |
|  | **3.2.5** Dataflow diagram |  |
|  | **3.3** UML diagram |  |
|  | **3.4** Technologies used |  |
|  | **3.4.1** Hardware and software requirement |  |
|  | **3.4.2** Introduction to programming tools |  |
| 4 | **Project Time and Task Distribution** |  |
|  | **4.1** Timeline chart |  |
| 5 | **Implementation** |  |
| 6 | **Test Cases** |  |
|  | **6.1** Graphical user interface |  |
|  | **6.2** Test cases |  |
| 7 | **Conclusion and Future Scope** |  |
| 8 | APPENDIX A: **User Manual** |  |
| 9 | APPENDIX B: **Classes and External Libraries** |  |
| 10 | APPENDIX C: **Input and Output for Test Cases** |  |
| 11 | **Bibliography** |  |

**List of Tables**

|  |  |  |
| --- | --- | --- |
| **Sr. No.** | **Name of Table/Figure** | **Page No.** |
| 1 | Spiral Model |  |
| 2 | DFD Level 0 |  |
| 3 | DFD Level 1 |  |
| 4 | Use Case Diagram |  |
| 5 | Project Task Distribution |  |
| 6 | Project Timeline |  |

**1. Project Overview**

As we know, the data available on the internet is not in a structured format. So our main aim is to detect and disambiguate named entities in text and extract binary relations between entities based on patterns in textual or semi structured contents.

In addition to this, we systematically harvest textual patterns from text corpora to group similar patterns into sets. For this purpose, we make use of a dictionary of entity-class pairs. For example: knowledge bases like Freebase or DBpedia. In order to extract relations, we refer to the data available from Wikipedia.

Finally, we will have collection of semantically-typed relational patterns. It joins entities which are related to each other by mapping them.

For example: Media sources may use the verbal phrases *“received”* or *“was honoured with”* to say that a person won an award.

**2. Introduction and Motivation**

**2.1 Theory behind the project**

In recent years, it has become clear that for most information extraction tasks manually constructed systems are inadequate, because adapting such systems to domain changes is very expensive and time consuming. Thus, there is a growing interest in using machine learning techniques.

Both supervised and unsupervised learning techniques have been applied. Requiring a large annotated corpus, supervised techniques quickly becomes a less desirable option. Annotated corpora are expensive to create, and the need for a new corpus for each product type makes this approach highly impractical for knowledge capture from product reviews. On the other hand, unsupervised techniques tend to be more domain independent, and are often much cheaper to develop and modify.

Many years of research into Natural Language Processing (NLP) and Machine Learning (ML) have resulted in a suite of freely available algorithms capable of extracting information from general text. The specialized nature of text requires the development of new algorithms or modification of existing algorithms.

Today, every business person may have to access an overwhelming amount of potentially relevant information that is continuously produced in various media with varying interpretations. Identifying relationships using entity relationship query focuses on detecting and disambiguating named entities in text and extract binary relations between entities based on patterns in textual or semi structured contents. Our system will deal with the issue of comprehensive gathering and systematically organizing patterns for an open set of relations.

With the huge growth of World Wide Web (WWW), a wealth of many different subjects has become available online. This has opened the opportunities for user to benefit from the available data in many interesting ways. The explosive growth and popularity of the world-wide web has resulted in a huge amount of information sources on the internet. However, due to the heterogeneity and the lack of structure of Web information sources, access to this huge collection of information has been limited to browsing and searching.

**2.2 Problem definition**

Our main aim is to map predicates to phrases in a knowledge base. It identifies relationships for entity relationship query. This system can be used to organize a large number of relational patterns into sets of synonymous patterns and finally into a hierarchy of entities. It can boost information extraction (IE) and knowledge based population tasked by means of its repository of paraphrases for relations. It also enables advanced search over *“subject-predicate-object”* data.

**2.3 Need for the project**

It will be possible to organize a huge number of relational patterns into sets of synonymous patterns and finally into a hierarchy. It can boost information extraction and knowledge based population tasks by its repository of paraphrases for the relations.

It can improve information extraction by associating type signatures with patterns. It enables advanced search over subject-predicate-object data. It will be of interest to database community.

Above all, our system will allow computers to detect or understand that many phrases can have the same semantic meaning. In other words, computers will be able to understand that humans can say one thing in many ways. This is something that they currently fail to do. An example of this is shown below:

|  |  |
| --- | --- |
| Query: *dhoni plays for team* | Query: *dhoni member of team* |
| Result:  C:\Users\neha tembe\Pictures\Screenshots\Screenshot (19).png | Result:  C:\Users\neha tembe\Pictures\Screenshots\Screenshot (20).png |

**3. Analysis and Design**

**3.1 Software Development**

We will make use of the following software during the execution phase of our project:

1. **Hadoop**

Hadoop is an open-source software framework for storing data and running applications on clusters of commodity hardware. It provides massive storage for any kind of data, enormous processing power and the ability to handle virtually limitless concurrent tasks or jobs.

We will be using Hadoop to distribute the pre-processing tasks across the cluster. These tasks include the processing of Wikipedia pages, the RDF triples from DBpedia and the formation of a graph linking entities along with a collection of semantically equivalent phrases.

1. **Virtuoso**

Virtuoso Universal Server is a middleware and database engine hybrid that combines the functionality of a traditional RDBMS, ORDBMS, virtual database, RDF, XML, free-text, web application server and file server in a single system. Rather than have dedicated servers for each of the aforementioned functionality realms, Virtuoso is a *“universal server”*; it enables a single multithreaded server that implements multiple protocols.

We will use Virtuoso to upload the RDF triples. This will enable us to query the graph database for the various categories of entities.

1. **Neo4j**

Neo4j is a graph database management system developed by Neo Technology, Inc. It is an ACID-compliant transactional database with native graph storage and processing. The problem with Virtuoso is that it cannot take weighted edges. In Neo4j, everything is stored in form of an edge, a node or an attribute. Each node and edge can have any number of attributes. Both the nodes and edges can be labelled. Labels can be used to narrow searches.

In our system Neo4j is used to store the category graph whose edges comprise of the semantically equivalent predicates and their weights.

**3.2 Flow of the project**

3.2.1 Preliminary survey

We are going to refer to a project made by Miss. Bhakti Hinduja and her group under the guidance of Prof. Pankaj Vanwari. They have used the concept of *“sense making”* which focuses on making sense of ambiguous contexts and continuously making the found knowledge more precise by disambiguating the context.

The used effective analysis tools to find the key entities and their relations in the sense making task. Their most relation identification work focused on the relations like *“is-a”* or *“part-of”*, which expresses the connections between entities in a hierarchical structure.

Using project information, we aim to detect and disambiguate named entities in text and extract binary relations between entities based on patterns in textual or semi structured contents. Also, we make an effort to systematically harvest textual patterns from text corpora, to group similar patterns into sets, resulting into a subsumption hierarchy.

3.2.2 Feasibility analysis

The assessment is based on an outline design of system requirements, to determine whether there is technical expertise to handle completion of the project. The concern is whether the proposal is technically feasible.

**Operational feasibility**

Operational feasibility is a measure of how well a proposed system solves the problems and how it satisfies the requirements analysis phase of system development. Using our approach and concept, there is no loss of information as also, it can improve Information Extraction. The current system is planned by taking into consideration the requirements needed to implement the system so that it can be successfully completed in the given amount of time.

**Economic feasibility**

This involves questions such as how much time is available to build the new system, the amount of resources required and the tools to be used. As mentioned in the software requirements section, all the tools that we need are available easily and thus our project is feasible in terms of resources required.

3.2.3 Cost analysis

As per the hardware requirements, we need 8 GB RAM. Normal availability of RAM is 6 GB, so we need another 2 GB RAM which will cost around Rs. 1000 to Rs. 1500. Also, we need a SSD which will cost around Rs. 8000.

3.2.4 Process model

The model that we use for our project is the *“spiral model”*. The spiral model is similar to the incremental model, with more emphasis placed on risk analysis.

The spiral model has four phases:

1. Planning
2. Risk analysis
3. Engineering
4. Evaluation

A software project repeatedly passes through these phases in iterations (called Spirals in this model). The baseline spirals, starting in the planning phase, requirements are gathered and risk is assessed. Each subsequent spiral builds on the baseline spiral.

**Planning phase**

Requirements are gathered during the planning phase.Requirements like *“BRS”* that is *“Business Requirement Specifications”* and *“SRS”* that is *“System Requirement specifications”*.

**Risk analysis**

In therisk analysis phase, a process is undertaken to identifyrisk and alternate solutions. A prototype is produced at the end of the risk analysis phase. If any risk is found during the risk analysis then alternate solutions are suggested and implemented.

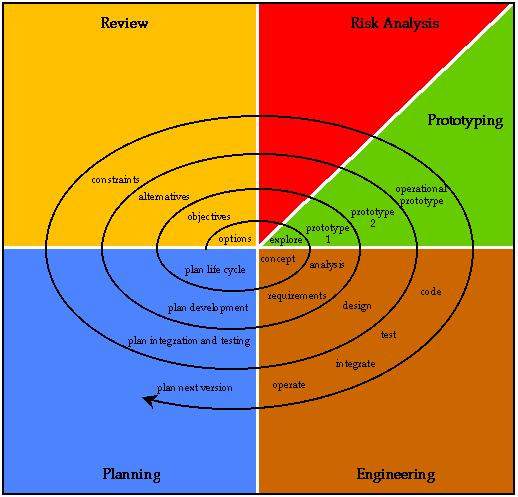
**Engineering phase**

In this phase software isdeveloped, alongwith testing at the end of the phase. Hence in this phase the development and testing is done.

**Evaluation phase**

This phase allows the customer to evaluate the output of the project to date before we move the development on to the next spiral.

Diagram of the spiral model:



3.2.5 Data flow diagram

**DFD – Level 0**

**C:\Users\neha tembe\Downloads\DFD - Level 0.png**

**DFD – Level 1**

**C:\Users\neha tembe\Downloads\Level-1.png**

**3.3 UML diagrams**

**Use case diagram**

**C:\Users\neha tembe\Downloads\UseCase.png**

**3.4 Technologies used**

**3.4.1 Hardware and software requirements**

**3.4.1.1 Hardware requirements**

* 64-bit processor
* 4 GB of RAM
* 500 GB hard-disk drive (minimum, preferably SSD)
* 5 PCs in a cluster (minimum)

**3.4.1.2 Software requirements**

* Linux 64-bit operating system
* Hadoop
* Virtuoso
* Neo4j
* OpenNLP
* Netbeans IDE

**3.4.2 Introduction to programming tools**

1. **Hadoop**

Hadoop is an Apache open source framework written in java that allows distributed processing of large datasets across clusters of computers using simple programming models. A Hadoop frame-worked application works in an environment that provides distributed storage and computation across clusters of computers. Hadoop is designed to scale up from single server to thousands of machines, each offering local computation and storage.

The Hadoop framework is made up primarily of the following four modules:

1. **Hadoop Common**

These are Java libraries and utilities required by other Hadoop modules. These libraries provide a file system, OS level abstraction and contain the necessary Java files and scripts required to start Hadoop.

1. **Hadoop YARN**

This is a framework for job scheduling and cluster resource management.

1. **Hadoop Distributed File System (HDFS)**

This is a distributed file system that provides high throughput access to application data.

1. **Hadoop MapReduce**

Hadoop MapReduce is a software framework for easily writing applications which process big amounts of data in-parallel on large clusters (thousands of nodes) of Commodity hardware in a reliable, fault-tolerant manner. The term *“MapReduce”* actually refers to the following two, separate tasks that Hadoop performs:

* **The Map task**

This is the first task, which takes input data and converts it into a set of data, where individual elements are broken down into tuples (key/value pairs).

* **The Reduce task**

This task takes the output from a map task as input and combines those data tuples into a smaller set of tuples. The reduce task is always performed after the map task.

1. **Virtuoso**

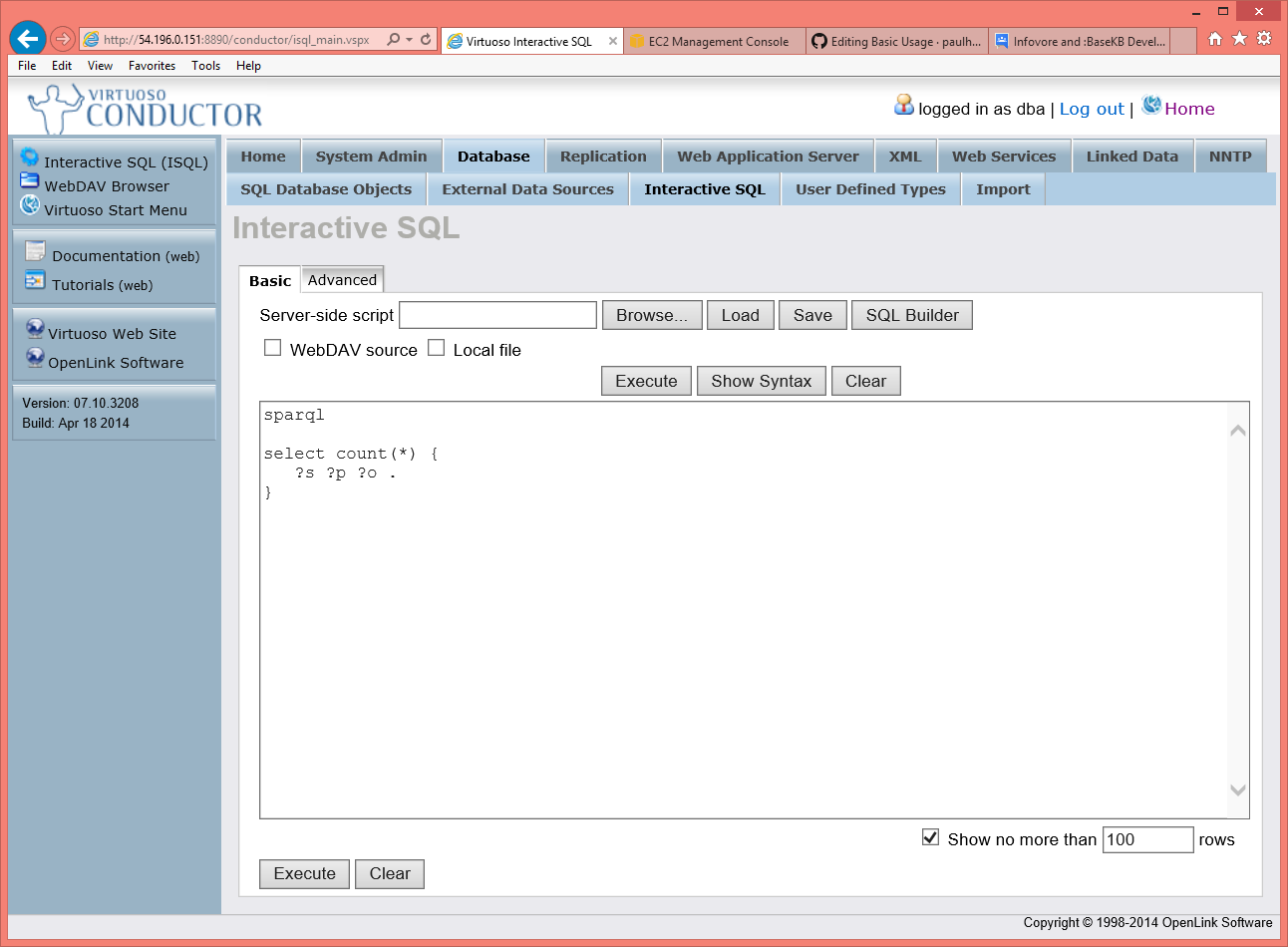
Virtuoso Universal Server is a middleware and database engine hybrid that combines the functionality of a traditional RDBMS, ORDBMS, virtual database, RDF, XML, free-text, web application server and file server in a single system. Rather than have dedicated servers for each of the aforementioned functionality realms, Virtuoso is a "universal server"; it enables a single multithreaded server process that implements multiple protocols. The open source edition of Virtuoso Universal Server is also known as “*OpenLink Virtuoso”*. The software has been developed by OpenLink Software.

Virtuoso is designed to take advantage of operating system threading support and multiple CPUs. It consists of a single process with an adjustable pool of threads shared between clients. Multiple threads may work on a single index tree with minimal interference with each other. One cache of database pages is shared among all threads and old dirty pages are written back to disk as a background process.

The database has at all times a clean checkpoint state and a delta of committed or uncommitted changes to this check pointed state. This makes it possible to do a clean backup of the checkpoint state while transactions proceed on the commit state. A transaction log file records all transactions since the last checkpoint. Transaction log files may be preserved and archived for an indefinite time, providing a full, recoverable history of the database. A single set of files is used for storing all tables. A separate set of files is used for all temporary data. The maximum size of a file set is 32 terabytes, for 4G × 8K pages.

Virtuoso is made up of client and server components. These components typically communicate with a local or remote Virtuoso server, which include:

* Virtuoso drivers for ODBC, JDBC, ADO.NET and OLEDB
* Conductor, a web-based database administration user interface
* ISQL (Interactive SQL) and ISQO utilities
* Documentation and tutorials
* Samples



1. **Neo4j**

Neo4j is a graph database management system developed by Neo Technology, Inc. Described by its developers as an ACID-compliant transactional database with native graph storage and processing, Neo4j is the most popular graph database according to db-engines.com.

Neo4j is implemented in Java and accessible from software written in other languages using the Cypher Query Language through a transactional HTTP endpoint.

In Neo4j, everything is stored in form of an edge, a node or an attribute. Each node and edge can have any number of attributes. Both the nodes and edges can be labelled. Labels can be used to narrow searches. As of version 2.0, indexing was added to Cypher with the introduction of schemas. Previously, indexes were supported separately from Cypher.

**4. Project Time and Task Distribution**

**4.1 Timeline chart**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Sr. No.** | **Phase** | **Duration** | **Start** | **Finish** |
| 1 | Designing the system | 25 days | 20/12/2015 | 15/01/2016 |
| 2 | Coding | 40 days | 20/01/2016 | 28/02/2016 |
|  | * Creating prototypes |  |  |  |
|  | * Basic Implementation |  |  |  |
|  | * Further development |  |  |  |
| 3 | Testing | 40 days | 01/03/2016 | 10/04/2016 |
|  | * Unit testing |  |  |  |
|  | * Integration testing |  |  |  |
|  | * System testing |  |  |  |
|  | * Acceptance testing |  |  |  |

**5. Implementation**

The implementation of our system can be subdivided into two categories based on time of execution – *pre-processing phase* and *real-time phase*. Both phases have a number of steps that have been described in the subsequent section.

* 1. **Pre-processing phase**

1. **Extract relevant sentences from Wikipedia using Knowledge Base**

The Knowledge Base contains facts in tuple format of the form

*Subject Predicate Object*

We use these facts to extract similar predicates from Wikipedia

Using Lucene search Object in the subjects Wikipedia page. Pick up the sentence(s) containing the object

Input: Wikipedia Text corpus + Knowledge Base

Output: Sentence(s), tuple pair

|  |  |
| --- | --- |
| Input | Output |
| * Wikipedia Text corpus * Knowledge Base | Knowledge Base Tuple, Sentence(s) |

1. **Extract predicates from sentences**

In the previous phase, we have broken down the raw text into a set of sentences. Each item of the set may contain may contain one or more sentences. The main task of this phase is to extract predicates given the subject and object. The OpenNLP library is used extensively to achieve this goal.

|  |  |
| --- | --- |
| **Input** | **Output** |
| * Sentences * Subject * Object | * Set of predicates |

Each sentence is divided into a group of tokens or words. Subsequently, each word is tagged by a *“Parts-of-speech Tagger”* or *“POSTagger”*. Token matching is carried out to detect the position of the object in the sentence. Using this reference, the words preceding the object are picked up as predicates by evaluating their tags.

1. **Collection of predicates with categorical data**

The knowledge base contains a hierarchy of categories for entities. We load this information into Neo4j. Each node will represent a category and the edges between any two nodes will contain a *bag* of predicates that can associate entities of the two categories. A *bag* of predicates is a set of predicates that have the same meaning.

1. **Uploading data to Neo4j as graph for the purpose of querying**

After we collect the predicate-phase pair with their category information. Upload the data in Neo4j Graph Database using the Neo4j batch importer tool

The categories form the nodes

The bag of predicates along with the actual predicate form the edges

|  |  |
| --- | --- |
| Input | Output |
| * Categories Hierarchy graph as Nodes list * Bag of Predicates as Edges | * Graph in Neo4j with Categories as nodes and Bag of Predicates on Edges |

1. **Assigning weights to predicates**

In the previous stage, a *“bag”* of predicates was generated. All the predicates have the same meaning. It may also be possible that one particular predicate or phrase may have been found more than once. As a result, the bag will contain duplicates. This count of occurrences will be termed the *“weight”* of the predicate. Thus, for each duplicate occurrence, we will add to the basic weight count of one predicate and eliminate the duplicate from the bag.

**5.2 Real-time phase**

1. **Accept user queries in a structured form**

This is the initial phase of execution from the user’s point of view. The user will provide the system with a query through the graphical user interface (GUI) of the system. Each query must follow a specific format (as shown below).

Query format: *[subject] [predicate] [object\_category]*

Example: *dhoni plays for team*

Where: subject – *dhoni,* predicate – *plays for,* object category – *team*

Subject denotes the person or topic that the search is based on. The predicate will represent the relationship between the subject and the object that we expect the system to provide as a result. Lastly, *“object\_category”* denotes that category of entity to which we expect the answer (object) to belong.

1. **Running the algorithm**
2. Using the *subject* provided by the user generate all of its categories.
3. Pick those categories which have the input *predicate* in their edges
4. Now traverse the graph to reach the destination *object-category*
5. For all the matching edges in path take the most weighted phrase and search it in the *subject’s* Wikipedia Page/Knowledge Base for all the matching predicates. If the subject and predicate in the sentence match with what we have then the output is the object entity in that sentence.

|  |  |
| --- | --- |
| Input | Output |
| * User Query as : *subject predicate object-category* | * The corresponding *object* |

**8. Appendix A: User Manual**

Using the system is a fairly straightforward process.

Steps:

1. Open the GUI
2. Enter your search query in the text box seen on screen

The query must be of the format: *subject-predicate-category*

Here, *“category”* refers to the catgeory of the desired output predicate.

**9. Appendix B: Classes and External Libraries**

**Hadoop**

Libraries include logging and core Hadoop functionalities and also handling the MapReduce framework.

List of libraries:

* commons-lang-2.6.jar
* commons-logging.jar
* hadoop-common-2.7.1.jar - *Core operations of Hadoop*
* hadoop-mapreduce-client-core-2.7.1.jar - *MapReduce operations*

**Virtuoso**

Libraries include connection establishment, querying and error reporting libraries.

List of libraries:

* icu4j\_3\_4.jar
* iri.jar
* jena-2.6.0.jar
* virtjdbc3.jar - *Connection establishment*
* virt\_jena.jar - *Jena driver for java*
* xercesImpl.jar

**Neo4j**

Libraries include import tools, graph traversal tools, logging, and other dependencies of libraries.

List of libraries:

* lucene-core-3.6.2.jar - *Used by Neo4j for indexing*
* neo4j-codegen-2.3.2.jar
* neo4j-consistency-check-2.3.2.jar
* neo4j-consistency-check-legacy-2.3.2.jar
* neo4j-csv-2.3.2.jar
* neo4j-function-2.3.2.jar
* neo4j-graph-algo-2.3.2.jar
* neo4j-graph-matching-2.3.2.jar
* neo4j-import-tool-2.3.2.jar - *Importing large dataset*
* neo4j-io-2.3.2.jar
* neo4j-kernel-2.3.2.jar - *Basic operations of neo4j*
* neo4j-logging-2.3.2.jar - *Logging functionalities*
* neo4j-lucene-index-2.3.2.jar - *Used by Neo4j for indexing*
* neo4j-primitive-collections-2.3.2.jar
* neo4j-shell-2.3.2.jar
* neo4j-udc-2.3.2.jar

**Lucene**

Libraries include indexing tools, query generation libraries and highlighting functionality.

List of libraries:

* lucene-analyzers-common-5.4.0.jar
* lucene-core-5.4.0.jar - *Basic core functionalities*
* lucene-highlighter-5.4.0.jar - *Highlight search result*
* lucene-memory-5.4.0.jar
* lucene-queries-5.4.0.jar - *Query generator*
* lucene-queryparser-5.4.0.jar - *Query parser*

**Apache OpenNLP**

Libraries include tools to load external models (sentence models, part-of-speech model, token model, etc.), detect sentences, tokenize sentences and tag tokens.

List of libraries:

* opennlp-tools-1.6.0.jar

**11. Bibliography**

[1] B. Hinduja, Y. Kumavat and M. Sambe, Extraction of Knowledge from Text, Mumbai 2014

[2] *“Freebase”*, [online]. Available at: http://www.freebase.com/

[3] *“DBpedia”*, [online]. Available at: http://wiki.dbpedia.org/

[4] N. Nakashole, G. Weikum and . F. Suchanek, *“PATTY: A Large Resource of Relational Patterns”*, [online]. Available: http://www.mpi-inf.mpg.de/departments/databases-and-information-systems/research/yago-naga/patty/