"In Ersilia, to establish the relations that sustain the city's life, the inhabitants stretch strings from the corners of the houses, white or black or gray or black-and-white according to whether they mark a relationship of blood, of trade, authority, agency."

(Italo Calvino, Invisible Cities)

Abstract

Relation Extraction has gained much attention in recent years because of availability of huge amount of information in electronic form. Major portion of this electronic information is constituted by World Wide Web documents. The goal of this project is to identify useful information (*i.e.* entities and their relations) from unstructured text of web pages and record it into a form so that it can be queried whenever required in future and can be used for data mining and other applications. However, existing relation extraction systems are not fully automatic and require significant human effort when adapted to new domains.

This project presents an automatic and domain-independent solution to the problem of relation extraction. It is based on a semi-supervised technique called Snowball (Agichtein and Gravano, 2000) with variations to adapt it to work on highly unstructured text of web documents. The experimental results evidenced the achievement of high precision and recall of our system on extraction of different types of relations in unstructured text.

The main contributions and achievements of the project include:

- Adapting Snowball to work on web documents. The underlying technique was originally implemented and tested on offline corpus.
- Use of state-of-the-art named entity tagger which causes less number of incorrect relation identification errors than the errors in the results reported by Agichtein and Gravano(2000).
- Exploiting term-frequency weighting scheme and vector space model to convert the context into its vector representation which optimized the performance of the system while working on huge amount of unstructured text collected from web documents.
- Application of flexible cosine vector similarity measuring technique to match two contexts which ignores minor variations of the contexts.
- Generation of patterns with high selectivity to prevent recognition of incorrect relations by use of constraint-based pattern evaluation strategy.

Acknowledgments

I would like to take this opportunity to thank all the people who helped me to complete this thesis. I would first like to thank my supervisor, Dr. Steve Gregory, whose insights and guidance have helped me throughout the project .Thanks to Philip Topham and S. Jonnalagadda of Lnx Pharma who helped me clarifying different concepts related to the project.

Finally, my deepest thanks to my family who provide the love and support I can always count on. To my dad, my mom and my siblings, I love all of you so much!

Table of Contents

List of Tables and Figures

Tables

Figures

Chapter 1

Introduction

This chapter starts with the motivation and significance of the project. It then discusses the aims and objectives of the project followed by brief overview of the organization of rest of the thesis.

1.1 Motivation and Significance

Currently, the World Wide Web is a major source of all kinds of information (Doddington, *et al.*, 2004; Brin, 1998). People often need to just query search engines in order to get exactly the information they require, but this is not always the case because the information may be in the form of unstructured text scattered among millions of web pages. For example, people may need to know which researchers, or groups of researchers, in the field of computing make the most significant contribution to an organization's publications. This needs vast amounts of unstructured text to be mined to extract every piece of information answering such queries. For humans, it is very tedious and almost infeasible to go through a gigantic list of electronic documents to bring together all the required information. Moreover, human endeavours to collect such information may be biased or inaccurate. So we need a machine-based, less error-prone and unbiased solution to finding out about contributions and successes of individuals, groups, and organizations (Jonnalagadda, *et al.*, 2009). Collecting such information involves extracting entities and collating instances of predefined relations among them.

Organizations usually make long-term decisions after assessing their current market position relative to leading competitors, where electronic documents, such as e-news, blogs, articles, *etc.*, constitute a major portion of their informational resources. So, they allocate a fixed portion of their annual budget to hiring people to extract such information. Jonnalagadda, *et al.* (2010) state (as cited in Knowles & Linn, 2004) that on average pharmaceutical companies spend about 24% of their budgets allocated for marketing operations on gathering human and other informational and material resources in order to find out which research centres contribute the most to the field and which are the most outstanding teams.

In biomedical sciences, finding out the relations among biomedical entities, such as diseases, organisms, genes, and proteins, has been under research since the last decade (Jorg, *et al.*, 2010) because discovering such relations manually requires direct lab experimentation or in-depth study of published material, both of which require the additional hiring of specialized people. Therefore, an automated system is required that would recognize entities and information about how they are related to each other.

A solution to this problem was proposed by Berners-Lee *et al.* (2001) while describing semantic web pages. They suggested marking up web pages with some meta-information about page format, language structure, and page semantics. Such information may be embedded in web pages in the form of a well defined standard such as XML tags (Mcdonald, 2005) and filters. But it is infeasible to manually equip billions of web pages to contain enough semantic tags to cope with the wide variety of information extraction required because just a few people share common information requirements (Brin, 1998). This implies that static methods for acquiring information have very specific applications and cannot be adapted to fulfil any other informational requirement. A realistic course of action could be to teach computers to annotate the information according to the varying requirements of humans.

Relationship extraction is a subset of a wider domain for finding out a variety of information called Information Extraction. It is an application of natural language processing and is defined by Tatar and Cicekli (2009) as analysing text documents written in natural language to find required information. The tasks involved in information extraction include entity recognition and extraction, event extraction, and relation extraction. In other words, it involves converting unstructured text into structured values to fill up the pre-defined informational fields to be added into databases for future retrieval and application. The two main tasks of information extraction are to identify concepts and their associations. However, this project is mainly focused on enabling computers to extract relations between named entities from unstructured text as sufficient work upon entity extraction has already been done (Bikel, *et al.*, 1999).

1.2 Aims and objectives

This project aimed to develop a system that recognizes and extracts semantic relations between any pair of named entities bounded by a relation from the unstructured text of WWW documents. It brings into play an existing bootstrapping approach (discussed in later chapters) to automate the system and help it achieve its overall goal.

The study also emphasizes the significant coverage of various relations between entities with the highest level of automation and lowest level of human intervention possible, while retaining accuracy in the required output. To realize this objective, we have chosen a semi-supervised approach that requires training the system through just a few labelled examples.

We intended to make use of the unstructured text of www documents written in English as a corpus, rather than using a multi-lingual tactic to test and evaluate the system. Furthermore, entities bounded by local binary relations will be focused on here.

The system should also be domain-independent and can be shifted to any other domain without requiring any reworking of the existing implementation and sustaining the actual performance.

1.3 Organization of rest of the Thesis

Chapter 2 elaborates the theory related to perception of relation extraction task. It emphasizes the basic terms and concepts of relation extraction and analyses the properties of a relation with examples. It also describes evaluation metrics to be used for the task of relation extraction and finally structures the problem of relation extraction in formalized form.

Chapter 3 presents the Literature Review of main tasks of information extraction. It also covers existing successful approaches and tools related to three activities involved in relation extraction task *i.e.* sentence simplification , entity recognition and relation extraction along with their advantages and disadvantages relative to each other.

Chapter 4 covers the overall design and implementation details of the project. It also includes the underlying technique, used as a framework for the project design and the strategies used at different steps. The main algorithms used to design the whole system and some core implemented modules along with platform specifications are also discussed here.

Chapter 5 covers the testing and evaluation of the system in detail. The performance of the implemented system is analyzed through experiments carried out on two different datasets using same parameter

settings to get comparable results. The chapter closes with a summarized discussion of the results obtained from the experiments.

Chapter 7 finally presents conclusions of the study and provides guidelines for future extensions with some feasible solutions to those extensions.

Chapter 2

Theory of Relation Extraction

Relation extraction is a sub-problem of information extraction (Mcdonald, 2005) and is a task of finding semantic relations between entities from text written in natural language. It can be applied to different domains for different purposes; for example, in Bio-informatics in can be used to uncover gene-protein or protein-protein interaction. It can also be used for question-answering, *e.g.* filling in a pre-defined information field such as person's name in a query asking who is the Queen_of (England, ?); finally, relation extraction can be used for ontology construction, as in weather forecasting.

This chapter intended to elaborate on the basic knowledge of entity and relation extraction tasks. It describes the task of relation extraction with examples, discusses the most important terms and concepts related to the task, and explains its applications and evaluation factors.

2.1 Entity

An entity is generally anything having individual existence. A named entity can be defined as an identifiable concept in text and is related to other entities (Mcdonald, 2005). In ACE (2004), named entities were categorized as being a person, location, organization, facility, or geo-political entity. In a sentence, an entity can be referred by something other than its name. Such references are called entity mentions. It can be signified by any of following three terms:

- Name mention: representing an entity by a proper noun, *e.g.* The city of *Bristol*.
- Pronoun mention: referring to an entity by a pronoun, *e.g. he* has merged *his* business with that of *his* friend.
- Nominal mention: an entity can also be referenced by a nominal expression and does not make use of only a single proper noun or a pronoun, *e.g. The two cats* seemed to be running.

An entity may consist of a single word or may be multi-word and may have some pre-defined boundary (Roth & Yih, 2002). In a multi-word entity, its mention can be further categorized as head- words and extent words. Head words contribute more than extent words in an entity representation and are, therefore, the most important kind of entity mention (GuoDong, *et al.*, 2005); for example, in the phrase *The University of Bristol*, *University* is considered the head word, while *the*, *of*, and *Bristol* are extent words of the entity.

2.2 Relation

Generally, a relation is a natural or logical connection between two or more objects. These objects may not necessarily have physical existence, but rather, may be abstract entities (Miller, *et al.*, 1998); examples of abstract entities include an amount, with or without a measuring unit, or an event. From the perspective of natural language processing and information extraction, a relation can be defined as an association between identified concepts, evidenced by some text written in a natural language. Here,

concepts are also called named entities. In simple notational form, let **R** be a relation between entities $\mathbf{e_1}$ and $\mathbf{e_2}$, the relation can then be written as follows:

$$\mathbf{R_C\ (e_1,\ e_{2,})}$$

Here, **C** is the context where this relation holds. The above predicate is a representation of a binary relation, which demonstrates that the arity of a relation is actually a number of entities in such relation to one another. Continuing this analogy, an n-ary relation can be written as follows:

$$\mathbf{R_C\ (e_1,\ e_2,\ e_3,\ldots,\ e_n)}$$

A simple relation contains at least two entities and a piece of information depicting the relation that is used by humans when doing manual extraction or by a relation extracting tool in the case of an automatic extraction system. Some examples of relations are depicted in Table 2.1.

Sentence	Named Entities	Event/Time/Role	Relation
Eddie Murphy acted as a Prince of Zamunda in a movie called *Coming to America*	Murphy (Person), Zamunda (Location)	Prince (Fictitious Role)	Person-Location
William is the first son of George.	William (Person), George (Person)	Is (Time not Considered Important in this situation)	Person-Person
Simmons hired Erik as a managing director.	Simmons (Organization) Erik (Person/Employee)	Hired (Action in Past)	Organization-Person or Organization-Employee
Microsoft may not agree to merge with IBM.	Microsoft (Organization) IBM (Organization)	may not agree (Action speculated as it has not yet happened)	Organization-Organization

Table 2.1: Examples of relations and their analyses

Unlike entities, relations may not always be confined to a single sentence. Relations that are confined to and can be analysed in a single sentence are called local relations. Whereas relations that involve two

entities or are dispersed across more than one sentence are called non-local or cross-sentential relations (Miller, *et al.*, 1998). This kind of relation is illustrated by the following example:

Fighter jets that were to be sold to China were traced back to the Pentagon. The parts were not sold directly to different foreign countries, but were obtained by a middleman named James Clerk.

Here, the relation between the *Pentagon* and the *fighter jets* can be extracted from the first sentence and is, therefore, an example of a local relation. However, the relation expressed by the phrase *to be sold* (or the relation between *China* and *James Clerk*) is an example of a cross sentential relation.

To develop a tool for automatic relation extraction, it must first be fully tested on local, sentence-level relations; it can then be updated to recognize more complex, non-local relations.

The examples in table 2.1 demonstrate that a relation:

- Must contain two or more entities which are interdependent in some way.
- Must have a context representing such a relation.
- May/may not be time or event-based (past, present *etc.*).
- May be shown to be negated and/or speculative in context.
- May be fictitious and may not hold between entities in reality.

2.3 Relation vs. Non-Relation

A relation must be mentioned in a document either though explicit text in the same document or it must be supported by understanding of the document. But if such relation is recognized from the conclusion drawn by readers of the document, it will be of a transitive or an ambiguous nature, such an unstable association that is not truly a relation because every reader can draw a different conclusion from the document; therefore, it is considered a non-relation.

If a reader is also using some knowledge either collected through personal experience or any text source other than the given document to portray a relation, it will also be regarded as a non-relation. Consider the following example:

"Adam, who is an English police officer, protected his house by an automatic security system and …"

The above example suggests a relation Employee_of (Adam, England) but a reader may also add some knowledge collected from the outside world that England is a part of the United Kingdom to draw a transitive relation Employ (Adam, United Kingdom) or if a reader knows that the United Kingdom is located in Europe, then he may also identify another unstable relation Employ (Adam, Europe) which, of course, requires some prior geographical knowledge on the part of the reader. Such transitive associations are actually considered as non-relations.

2.4 Types of Relation

In Automatic Content Extraction (ACE) meeting held in 2004, relations were categorized on the basis of their general features into five types: part, located at, near, social, and role. Meanwhile, in the seventh Message Understanding Conference (MUC) meeting held in 1998, these relations were classified into

four general categories: place, person, organization, and location. This categorization is based on the semantics of a relation because context perception is used to determine its type. On the contrary, some relations cannot be categorized through context insight, but require context analysis using grammar. Such relations are syntactic in nature; for example, Noun-Preposition-Noun, Subject-Verb, or Verb-Object.

On the basis of how a relation is supported by its context, a relation falls into one of the general categories *i.e.* explicit and implicit relations.

2.4.1 Implicit Relation

A relation, that is not supported by explicit words but is evidenced by the contextual inference of the reader within the scope of the document where the relation holds. Simply stated, implicit relations are passed on to the reader by a natural understanding of the document. Such relations are most likely to link two named entities, not their mentions (Kambhatla, 2004; GuoDong, *et al.*, 2005). Consider the following example:

"Bill Gates, owner of Microsoft, dealt with Skype to buy the software product. Andrew Jones, his legal advisor, disclosed it before media while speaking to reporters last Sunday… "

In the above example, there is no explicit syntactic connection between mentions which shows that Andrew Jones is currently serving as an employee in an organization named Microsoft because the implicit relation Employee-Organization (*Andrew Jones*, *Microsoft*) can only be drawn by understanding the corresponding context. This shows that such relations are harder than explicit relations to be recognized by machines.

It is worth noting here that even if the reader of the context already has some knowledge related to the context of implicit relations and if it is not evidenced anywhere within the whole document (where these implicit relations take place), such knowledge cannot be used to portray an implicit relation.

2.4.2 Explicit Relation

A relation is said to be explicit between any two entity mentions if it is supported by the text that syntactically links those mentions (Kambhatla, 2004; GuoDong, *et al.*, 2005). Mostly, such relations hold between mentions of entities other than names. In this type of relation, two entities may be linked by an event or one entity may work as a syntactic modifier of another entity.

2.4.2.1 Modification

If an entity mention syntactically modifies or provides a description of another entity mention, the resulting relation formed between those entities falls generally under the category of explicit relations. Consider the following examples:

> *"Paul has accused Osborne of prioritizing the interests of the richest people over those of the poor"*

In the above example, the relation Accuse (*Paul*, *Osborne*) is explicit because Paul is used as a predicate adapter of Osborne.

> *"J. Michael Banks, 42, of Evans Street…"*

Addresses, as in the above example, are a major source of explicit relations between entities. The relation Located (*J. Michael Banks, Evans Street*) can be drawn from the above example.

"Gordon Brown, the Prime Minister of Britain, visited Wembly stadium."

Here, the title 'prime minister' is a descriptor of the role of an entity named 'Gordon Brown'.

2.4.2.2 Events

Events also link one entity mention with another through explicit relation substantiation in the text.

"Carol, a gunman was killed by a man named Jimmy when he got into a crowd of rioters to prevent their destructing actions"

The relation Killed (*Carol*, *Jimmy*) holds in an event of attempting to prevent destruction caused by rioters and falls under the explicit relation category.

2.5 Entity Recognition Task

Entity identification is a preliminary task for the relation extraction task (Jorg, *et al.*, 2010) and it refers to processing the given text for finding out and classifying entity mentions. In ACE (2004), it is defined as an Entity Detection and Characterization (EDC) task. This is one of the most difficult tasks (Doddington, *et al.*, 2004) of the whole process of relation extraction because every relation has two or more entities. Once named entities are accurately spotted, they can be analysed within the context to find out how they are related to each other.

The tasks of extracting entities and evaluating relations are inter-dependent, so failure of the previous task will have obvious effects on other tasks and cause an overall incorrect result (Tatar and Cicekli, 2009; Roth& Yih, 2002). That is the reason that most of the techniques are presented on the assumption that text has been pre-processed through the most reliable entity recognition tools. The things that make entity recognition difficult are described in Tatar and Cicekli (2009) and Jonnalagadda *et al.* (2010):

- Unstructured text doesn't follow any pre-defined format for placing entities.
- There is no standard naming convention for naming entities. Entity names may also contain some commonly used verbs, adjectives, and other words. Hence, it is difficult to test and figure out the correctness and accuracy of such systems.
- Entities' names are abbreviated and replaced by the original name in different ways.

Some terms or words in a sentence are not entities but are used to describe a named entity. These are called co-references of an entity. Consider the following sentence, for example:

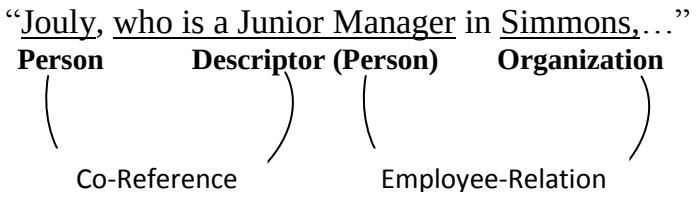

Figure 2.1: An example representing Entity Terminology

Analysing a sentence semantically to recognize entity-names and classifying those into classes and sub-classes (Jonnalagadda, *et al.*, 2010) can help in recognizing the relation accurately. In MUC-5, the

concept of hierarchical structure was used in the information extraction process which seemed to be useful, especially when extracting entities (Grishman & Sundheim, 1996).

2.6 Relation Extraction Task

Relation extraction is a vital task of information extraction (Grishman & Sundheim, 1996; Doddington, 2004) and is an example of an application of natural language processing. In ACE (2004), relation extraction task was defined under Relation Detection and Characterization (RDC) as the task of identifying explicit as well as implicit relations between entity mentions detected by preceding EDC tasks in natural language text. Once entities are detected, relation extraction involves extracting the associations used to bind those entities. A technique of implementing this task must first be able to evaluate whether the relation holds or not. If so, it must further identify which type of relation it is.

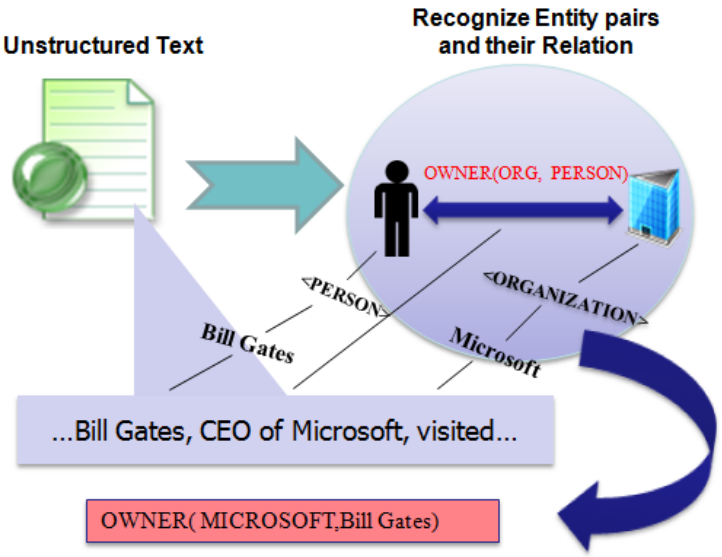

Figure 2.2: An illustration of entity and relation extraction task

Associations between entities can be of different types. An association may denote a possession or membership of one thing to another; for example, an owner, employee, or member of some organization, club, or society. Associations may also show a physical linkage of one entity with another; that is, one thing can be placed, built, or located near another. Finally, an association can show some social or personal relation: a parent, a supervisor, *etc.* (Hasegawa, *et al.*, 2004; Doddington, *et al.*, 2004).

An important application of relation extraction task is automated question-answering (Hirschman & Gaizauskas, 2001; Voorhees, 2000). Existing search engines and information retrieval systems retrieve documents having high similarity with users' queries where users expect answers in the form of a string of few words written in natural language (Hirschman & Gaizauskas, 2001). Furthermore, queries in their original interrogative form are not acceptable by information retrieval systems, but need to be made in an assertive form to get the documents containing text and answering the questions.

Automated question-answering is defined in the 8th Text Retrieval Conference (TREC) (Voorhees, 2000) as the task of answering a question in Doc_ID, Answer String format. Here, Doc_ID is a unique representation of the document containing the required answer and the answer string is the shortest possible answer up to maximum length of one paragraph. Questions of the following types were used in the task description and evaluation; they can be answered through a relation extraction task:

- Where is the Taj Mahal?
- Who was the first American in space?
- Who was the sixteenth president of U.S.?

A simple bag-of-words technique with document similarity can retrieve the required answers up to a minimum length of one paragraph, but more precise answers consisting of exactly one or a few words require a more sophisticated text processing technique than this (Voorhees, 2000). Such sophisticated processing can be done by relation extraction systems. A relation extraction system can be used to fill up a pre-defined template and find results in structured text which can then be queried for getting required information.

Another important application of relation extraction is in the field of bioinformatics. Much of the voluminous amount of bio-medical information is in published form but has not yet been put into practical applications because of its unstructured form and large quantity (Chiang, *et al.*, 2004). Also, developments in the life sciences heavily depend on the development of techniques for identification and extraction of bio-medical entities and their associations (Saric *et al.*, 2006). Computational linguistics has been providing a number of tools and techniques to mine bio-medical text; for example, the Gene Information System (GIS) proposed by Chiang, Yu, and Hsu (2004) is used to extract information related to genes, their functions, and bio-medical associations with other bio-entities and their interaction with each other.

Additionally, relation extraction can be applied to other domains to solve certain problems. It can be applied to find and anonymize the text in order to maintain data privacy. It can also be used to construct ontologies by automatically evaluating relations between concepts in ontologies. Ontologies are used to capture some specific information from text such as weather forecasting, linkages among different geo-graphic locations, *etc.*

2.7 Steps in Relationship Extraction

The process of relationship extraction can be broken down into a number of tasks. Some of them are optional and added to boost performance, such as simplifying the sentences, while others are considered as basic constituent tasks, like named entity recognition and identification of relations. The following are commonly found steps in relation extraction tasks:

- First, sentences containing relations are identified and simplified using sentence simplification methods and tools in order to expedite the extraction process and increase performance parameter values.
- Once sentences are simplified, the output of this task is taken by another task called named entity recognition, which involves identifying names of entities in the given text such as location, organization, person, *etc.*, and performing other operations on it such as classification, sub-classification, and tagging; the operations are performed depends on the type of technique being used.

- Once all entities are identified, their context is parsed to recognize relations among those entities, which is the required output.

2.8 Evaluation Metrics for Relation Extraction

There are a number of existing approaches for extracting relations from free text that will be discussed later, but to find out which one is perfect is difficult. Some techniques are best suited to extracting a relation from text related to one specific domain and do not work well when shifted to another domain since they may not be able to identify domain-specific jargons with high versatility. Some techniques are domain-independent and work well on general text, such as newspaper articles, blogs, and stories. Therefore, their accuracy and performance may vary when they cope with different kinds of text input. Although we cannot guarantee the correctness and accuracy of a system by simply testing it on just one kind of text, common evaluation methods can provide common ground to all types of relation extraction techniques and can make possible the comparison of different approaches.

Four measuring factors work as the criteria on the basis of which we can consider one technique preferable over others *i.e.* level of correctness, recall, precision and f-measure.

2.8.1 Level of Correctness

The simplest and most basic criterion for evaluating the performance of a relationship extraction system is to measure how much it is able to recognize and extract out the required information correctly by testing it with a wide variety of text (figure 2.3). The tendency of a system to cope with false or confusing data also helps in determining its level of correctness. Tatar and Cicekli (2009) state that the efficiency and success of an information extraction system depends on how often it is able to recognize an entity correctly and its ability to distinguish entities from non-entities, so we can test our system by giving it text containing relations as well as non-relations. Hence, we need to observe the system by measuring the number of:

- **False-positives** or the number of relations that are identified by the system as instances of some relation type, but actually are non-relations.
- **False-negatives** or the opposite of False-positives. This is the number of relations contained in the text given to a system that are considered by the system as non-relations.
- **True-positive**s are the number of relations that exist in the text and are recognized by the system as instances of certain relation types.

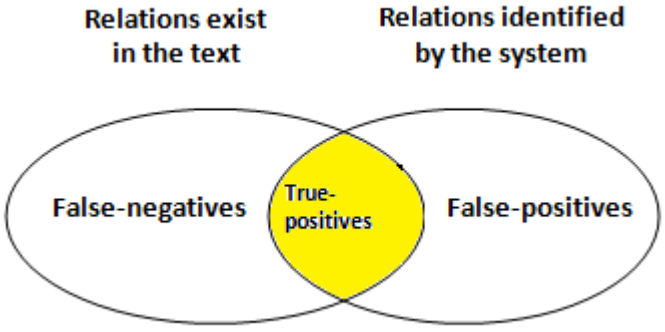

Figure 2.3: An illustration of level of correctness

2.8.2 Recall

Recall is the ratio of the number of relations correctly extracted from the input text to the number of relations that actually exist in the text (Hasegawa, *et al.*, 2004). In other words, it is the measure of coverage of a system in recognizing relations. Here, if N_c represents the number of <Entities, Relation> pairs extracted by a system and N_K represents the total number of <Entities, Relation> pairs that actually exist in the text, but may or may not be detected by the system. Then, according to Hasegawa *et al.* (2004), recall R of a system can be calculated as follows:

$$R = \frac{N_C}{N_k} \ldots\ldots\ldots\ldots \text{(2.1)}$$

2.8.3 Precision

Precision determines the measure of the number of correct relations among all the relations recognized by the system (Hasegawa, *et al.*, 2004). In other words, it represents how accurately the system finds <Entities, Relation> pairs. It differs from recall in that it is the ratio of the number of correct relations to the number of all relations spotted by the system, rather than actual number of relations that exist in the text. If N_r represents the total number of <Entities, Relation> pairs found in the given text by the system and if the number of accurate pairs from all the detected ones are represented by $N_{c,}$ then, according to Hasegawa and *et al.* (2004), precision P can be determined as follows:

$$P = \frac{N_C}{N_r} \ldots\ldots\ldots\ldots \text{(2.2)}$$

2.8.4 F-measure

As we know, precision and recall are two important measures of an information extraction system's performance; however, it is more desirable to have a single performance factor. So if we know the recall and precision of a system, then we can take the simple harmonic mean of the both (Makhoul, *et al.*, 1999). Consider a system has recall R and precision P then, according to Hasegawa *et al.* (2004), its F-measure F can be written as follows:

$$F = \frac{2PR}{P+R} \ldots\ldots\ldots\ldots \text{(2.3a)}$$

This can also be expressed as

$$F = \frac{PR}{1/_2(P+R)} \ldots\ldots\ldots \text{(2.3b)}$$

2.9 Some important concepts of Relation Extraction

2.9.1 Annotated Corpus

Generally, a corpus (plural corpora) is a collection of two or more text bodies. A text body may represent an article, recorded speech, an online blog, or any other written or spoken text. From the context of linguistics, McEnery and Wilson (1996) defined it as a finite-sized, machine-readable sample which must be a true representative of the overall diversity of the language from which it is taken. A scientific corpus must be available to and used by all researchers as a standard reference for evaluating the results of their

studies. The success of MUC ('Message Understanding Conference', n.d), Senseval ('SENSEVAL', n.d), ACE ('Automatic Content Extraction', n.d), and TREC ('Text Retrieval Conference', n.d) conferences in making information extraction a rapidly-developing and highly-focused application of natural language processing is sufficient to bear out the importance of a corpus.

An annotated corpus can be defined as an assortment of texts with added mark-ups representing some domain-specific and/or language-specific information (Pyysalo and *et al.*, 2007), such as syntactic dependency, entity relations, *etc.* The availability of an annotated corpus is an important requirement for information extraction since it provides the gold standard for training and evaluating tools and techniques designed for such purposes.

Bio-Infer (Pyysalo and *et al.*, 2007) and Penn Treebank (Mitchell *et al.*, 1993) are examples of widely used corpora used for developing and testing entity and relation extraction systems. Bio-Infer (Bio - Information extraction resource) is a manually annotated corpus specifically designed to be used by bio-medical information extraction systems. Penn Treebank (Mitchell *et al.*, 1993) is an American English language corpus composed of 4.5 million words. The corpus has been annotated to give information about syntactic structure mainly including parts-of-speech (POS).

2.9.2 Parsing

Generally, parsing is a process of analysing and dividing a text string into its syntactically logical elements and identifying each element's role within the given string. From the perspective of information extraction, parsing contributes much in providing syntactic dependency of text phrases which in turn helps in recognizing concepts and their associations.

The parsing types used most widely for relation extraction are full-parsing and shallow parsing. An illustration of both parsing types is presented with examples in figure 2.4. Full-parsers analyse the structure of a given text as a whole. Such a parser will keep recurring until it generates a full parse structure containing all the terms present in the original text string. Although it is more time and space consuming than shallow-parsing, it can be applied in order to resolve syntactic ambiguities in some situations (Yakushiji *et al.*, 2001).

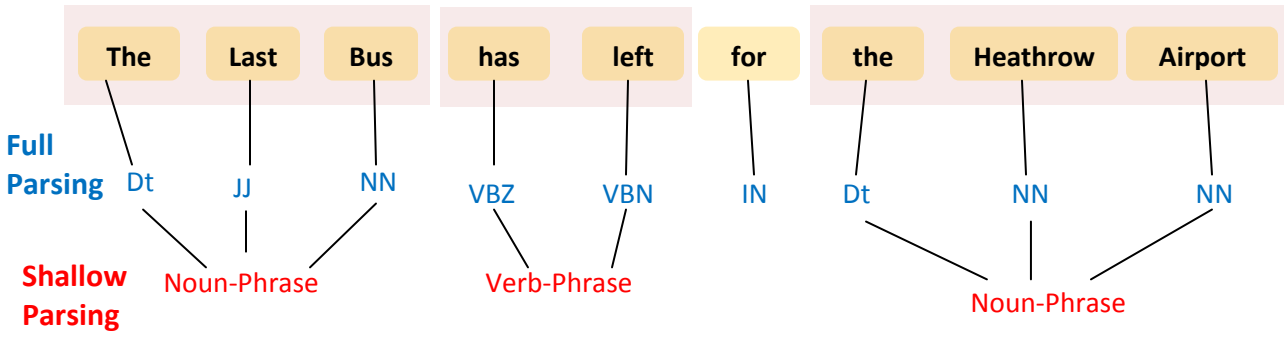

Figure 2.4: An example depicting full-parsing and shallow-parsing. In this figure, DT=determinator, NN=noun, IN=interjection, VBZ=auxiliary verb, VBN= main verb, and JJ= adjective.

Another type of parsing, shallow parsing (Abney, 1991; Molina & Pla, 2002) or base-phrase chunking, is considered to play an important role in the task of finding out relations (GuoDong, *et al.*, 2005) and is a precursor to full-parsing (Abney, 1991). Unlike full-parsing, shallow parsing is used to partition given text into non-recursive segments. Each segment is a group of syntactically-related words and is recognized by a syntactic phrase type. The resulting segments are also called chunks. Shallow parsing does only partial analysis of text and has been captivatingly successful in replacing full-parsing in most circumstances. Shallow parsing is also used to solve other natural language problems, including machine translation, text summarization, and speech recognition (Molina & Pla, 2002). Identifying noun, verb, or prepositional phrases from a sentence is an example of shallow parsing.

2.9.3 Local and Global Context

Generally, a context is a set of informational circumstances or a situation surrounding an object, event, concept, action, statement, or term under consideration and interprets its meaning. From the perspective of entity and relation extraction, context refers to text within which entities co-occur and is used to illustrate how such entities are bounded by an instance of some pre-defined relation type. Context may be local or global (Giuliano *et al.*, 2006; Giuliano *et al.*, 2007a ; Giuliano *et al.*, 2007b). Global context is sentence or string, where entities, along with some relation, exist; meanwhile, local context refers to a text chunk of limited size which may be a part of global context around entities. Given an entity pair with relation R, local context is used to determine the individual contribution of each entity in establishing relation R; *e.g.*, in protein-protein interaction, local context can be used to determine the agent and target of the relation.

Giuliano *et al.* (2006, 2007) and Bunescu and Mooney (2005) determined the following three frequently observed context positions, where words are most likely to depict a relation between a pair of entities:

- **Left-middle:** tokens (also called words) preceding the first occurring entity and between the two entities, *e.g.* the CEO of [Organization], Mr. [Person].
- **Middle:** only the words between the two entities, *e.g.* [Person], president of [Organization].
- **Middle-right:** the words between the two entities and the words following last occurring entity, *e.g.* [Person], an [Organization]'s experienced software engineer.

2.10 Formalizing the problem

Consider a sentence S, written in a natural language with a sequence of **n** number of entities. Here, each entity is represented by a unique index in an increasing order of their occurrence as illustrated below:

$$\mathbf{E = \{e_1, e2, \ldots, e_n\}}$$

or defined over the following schema:

$$\mathbf{T = \{t_1, t_2, \ldots, t_m\}}$$

Once for each entity pair e_i in E, its type type t_j has been determined from schema T; the problem is to first find out a relation associating each entity pair $< e_i , e_{i+1}>$ in E:

$$\mathbf{R=\{ r_{(1,2)}, r_{(2,3)}, \ldots , r_{(k-1,k)}\}} \qquad \text{Whereas, } \mathbf{r_{(L,L+1)} = <e_L, e_{L+1}>}$$

Here, L= a whole number, used to index an entity

L+1= an index used to identify entity at subsequent location to Lth entity.

To solve this problem, it can be further simplified by applying 'divide-and-conquer' strategy and dividing the problem into two sub-problems. The first sub-problem is to find out whether a relation $r_{i,i+1}$ is a relation associating both entities or is a non-relation. If a relation exists between the two entities, the next step is to identify the class of relation the instance belongs to and classify it accordingly.

Simply, given any two entities the whole problem of relation extraction is about finding the answers to the following two questions;

1. Does any relation exist between the two entities?
2. If the answer to the above question is yes, which type of relation is it?

Here, we are most interested in finding out binary and local relations between any two entities of the following schema:

T = {Person, Locations, Organization}

It means that the binary relations we can determine are as follows:

- Person-Person
- Person-Location
- Location-Location
- Person-Organization
- Location-Location
- Location-Organization
- Organization-Organization

2.11 Summary

This chapter described the basics of relation extraction. It is specifically designed for a novice reader to build up enough background to understand the subsequent chapters. We started with the elaboration of examples of entities and relations; this was followed by discussion of what the relation actually is and how it can be distinguished from a non-relation. Then we discussed two generic relation types: implicit and explicit. The chapter proceeded with illustrations of entity and relation extraction tasks, the commonly followed steps for extracting relations, and the metrics to be used for evaluating the performance of the overall system; that is, the level of correctness, precision, and recall and some of the basic concepts used frequently in the information extraction domain. Finally, the chapter concluded by portraying the general formulation of the problem to be solved in this thesis. The next chapter presents a number of existing approaches to solve the problem at hand.

Chapter 3

Literature Review

3.1 Introduction

Relation extraction has long been considered the most important and challenging area of computational linguistics. As relation extraction is a sub-problem of information extraction, traditional information extraction approaches can be applied to the task of relation extraction.

This chapter will review the existing attempts made to solve the problem of relation extraction. It also presents a review of different tools and techniques designed to carry out all three steps of the relation extraction task (section 2.8).

3.2 Message Understanding Conferences (MUCs)

The concept of automating information extraction (including relation extraction) was first initiated in MUCs (Grishman & Sundheim, 1996). These conferences were initially aimed at developing systems and underlying techniques to automatically analyze and extract specific information from military text messages. In each conference, participants were given some sample text and were asked to develop certain tools and techniques to extract specific data field values given in the templates to be filled up. All the developed systems were tested by inputting some text containing information, which can used to fill up rep-defined template fields and the results were then checked against manual information extraction.

In MUC-6, tasks related to Entity Recognition and other information extraction were under research and more emphasis was given to the portability of Information Extraction systems (Grishman & Sundheim, 1996). Rather than developing domain-specific systems, groups were given the goal of making them versatile enough to analyze text from any domain. Such domain-independent systems can be adapted easily to fulfill the changing requirements for information extraction.

MUC-7 was the last MUC, held in April 1998. In this conference, the emphasis was given to improving the performance of information extraction systems and enabling them to cope with more complex tasks (Marsh, 1998). The tasks involved in MUC-6(Grishman & Sundheim, 1996) and MUC-7(Marsh, 1998) are shown in Table 3.1.

3.3 Automatic Content Extraction (ACE)

After MUC-7, the work was carried on by ACE meetings (National Institute of Standards and Technology, 2000). The tasks defined in ACE (Doddington, *et al.*, 2004) include: Entity Detection and Classification (EDC), Relation Detection and Characterization (RDC), and Event Detection and Characterization (VDC). Simply, ACE tasks include extracting three informational objects with their elements and attributes, *i.e.* entities, relations and events. In the EDC and RDC task, entities and relations are classified and sub-classified according to pre-defined taxonomy. The implementation of most of RDC tasks involved supervised learning approaches, for which we need much effort and time to prepare a large amount of annotated corpora (Hasegawa, *et al.*, 2004).

Task Name	Description
Tasks Common to MUC-6 and MUC-7	
Named Entity	Involves recognizing named entities such as Person, Location, Organization *etc.* and enclosing it with proper Standard Generalized Markup Language tag.
Co-reference	Finding out the co-referring words and phrases such as "he", "the same young man" etc., for all information including that found in Named Entity and Template Element tasks.
Template Element	Looking into the given text and extracting the information about entities if any evidence found.
Scenario Template	Extracting all the events in the input text and entities relevant to that event.
Tasks added to MUC-7	
Template Relation	This newly added task involves finding out about how entities present in the input text are related to other entities in it, irrespective of their domain, such as, "Employee_of","Organization_of"and"Manufacture-of"etc.
Multi-lingual Entity Task	Enable the Named Entity Task for text written in one of two other languages, *i.e.* Chinese and Japanese.

Table 3.1: Tasks defined in MUC-6 (Grishman & Sundheim, 1996) and MUC-7 Marsh (1998)

3.4 Sentence Simplification

In order to find out a relation, a system must first search for the sentences which contain entities showing some association within the given text. Once appropriate sentences are taken from the given unstructured text, sentence simplification methods can be applied to expedite the performance of relation extraction system. Here, the term sentence simplification should not be misunderstood with its normal interpretation,

i.e. to make the sense of a sentence clear, so that one can understand its meaning easily. Here, it means the application of certain tools and techniques to remove all unnecessary words and symbols while preserving the syntactic correctness and information about relations in the sentence (Hasegawa, *et al.*, 2004). Simply, such removal of words should not affect the recall and precision of any information system. Applying automatic methods for sentence simplification has shown recall to be improved up to 8% without losing persistency in precision (Jonnalagadda & Gonzalez, 2010).

In any text-mining process, text must inevitably be first parsed to extract the required information, for which natural language parsers are used and the performance of such natural language parsers will have significant impact on the performance of the overall text-mining process (Jonnalagadda, *et al.*, 2010). However, the normally used natural language parsers can perform the task of sentence simplification, but they may fail to make accurate decisions when confronting Jargon related to a specific domain (Jonnalagadda & Gonzalez, 2010). So we need to update and extend the functionality of the existing natural language processors or develop separate tools that pre-process the text to filter and pass on the simplified sentences to other phases of the Relation Extraction System.

Another considerable problem is that text may contain different kinds of sentences with a varying level of complexity. Increasing the complexity of sentences will result in a decreased performance of the overall process. It is more convenient to convert a more complicated sentence into two or more simpler ones.

Jonnalagadda *et al.* (2010) suggested an algorithmic solution to reduce such complexity level of sentences using the associations between words identified by link grammar parsers and punctuation symbols. The flowchart representation of such technique is shown in figure 3.1. The process of breaking down a complex sentence into two or more simpler sentences can be recursive because each of the resultant simpler sentences may still be complex enough and can be further decomposable into simpler sentences. Some systems also use the taggers, which label each word according to its related Parts of Speech (POS), while others make use of a syntactic parser that labels each word by analyzing its role in a sentence and its relatedness to other words within the same sentence. Parsers analyze a sentence syntactically, so they are less accurate than POS taggers, but are much faster than POS taggers (Jonnalagadda & Gonzalez, 2010).

Jonnalagadda & Gonzalez (2010) also introduced another method of sentence simplification and implemented it in BioSimplify, which increased the recall by about 20% and f-Score around 7%. This approach generates all possible simpler versions of the given complex sentence by replacing all possible shorter alternatives in place of replaceable sentence elements. Mostly in noun phrases just pre-modifier can be removed *e.g.*

"The newly found GBV-D virus causes Hepatitis".

In this sentence, the pre-modifier 'newly' can be removed without losing the syntactic information required by a relation extraction system.

3.4.1 BioSimplify

Biosimplify is an example of a domain-specific tool which can simplify medical text (Jonnalagadda & Gonzalez, 2010). BioSimplify improved the parser's accuracy by up to 4.23% (Jonnalagadda, *et al.*, 2010; Jonnalagadda & Gonzalez, 2010). It maintains a list of genes' names with a unique one-word meaningful identifier and replaces long gene names with those pre-assigned identifiers (Jonnalagadda & Gonzalez, 2010). Other than this gene name replacing and cross-referencing it with identifiers (Jonnalagadda & Gonzalez, 2010), it uses link grammar to parse normal biomedical text (Jonnalagadda, *et al.*, 2010) and two important steps to simplify sentences for the parser, *i.e.* semantic analysis and syntactic analysis.

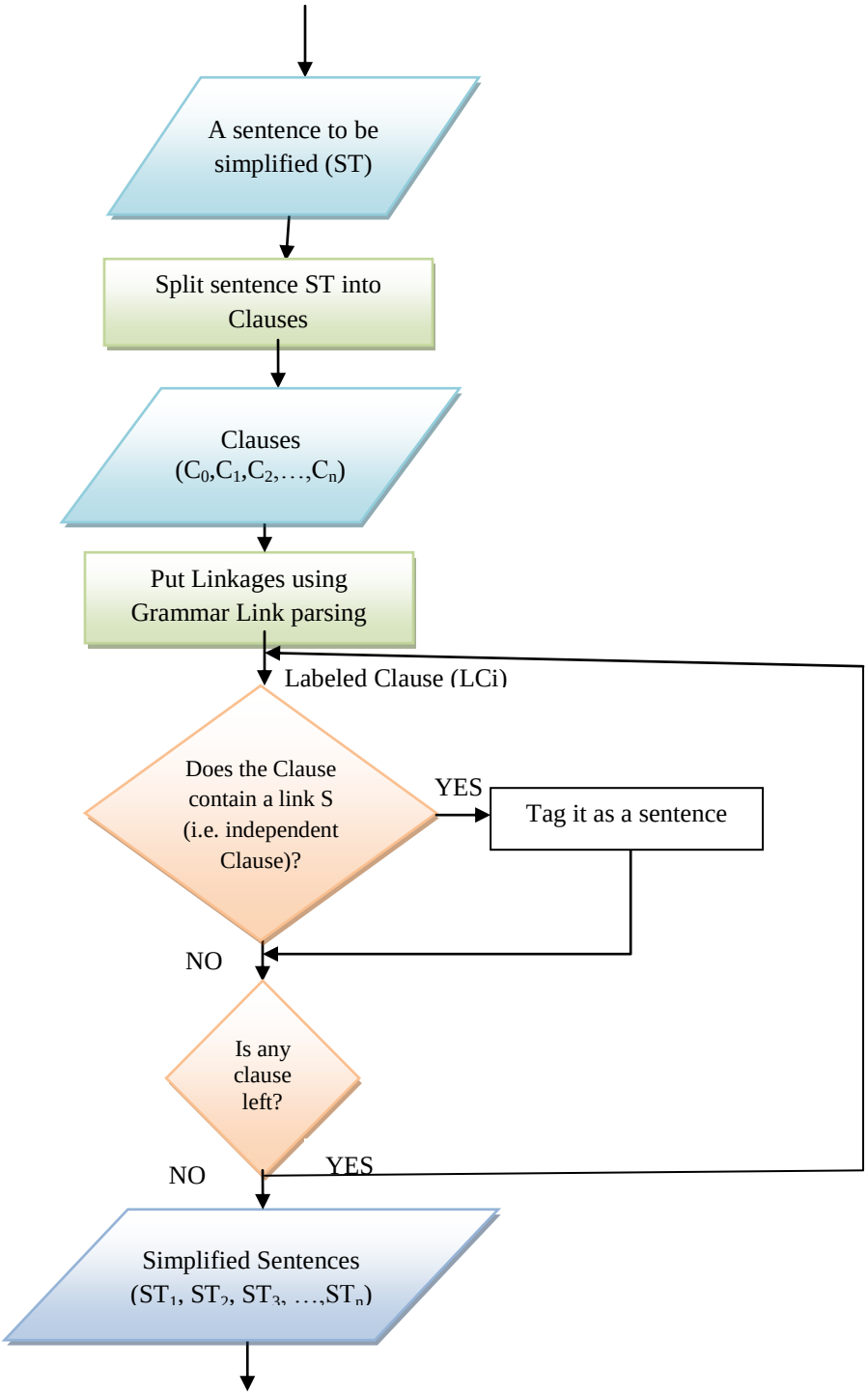

Figure 3.1: Flowchart of Complex Sentence Breakdown technique described by Jonnalagadda *et al.* (2010)

3.4.1.1 Semantic Analysis

A sentence is semantically analyzed for finding out and replacing gene names, phrases representing an entity and any phrase or word that is not usable in any way in finding out the required information to be extracted, such as the heading of a section of text (Jonnalagadda, *et al.*, 2010). All other multi-word

proper nouns are replaced by more generic one word identifiers called head nouns (Jonnalagadda & Gonzalez, 2010) *e.g.*

"Mostly, changes in OCA type-1 cause albinism which has not been found frequently"

is replaced by

"Mostly change in GENE12 causes disease which has not been found frequently".

Here, the information about the relation between *OCA type-1*and *albinism* is lost and cannot be recovered unless we maintain a list of sub-types of genes and diseases but it will decrease the performance instead of increasing it. So here replacing the word 'disease' with 'albinism' results in relation of OCA type-1 with all the diseases and hence will lead to false relation extraction.

3.4.1.2 Syntactic Analysis

A sentence is syntactically analyzed to find out which words will not cause the loss of grammatical information which is used by relation extractors while performing their task, if they are removed from a sentence (Jonnalagadda & Gonzalez, 2010). Sentence simplification tools determine automatically if eliminating a word or a phrase will cause a loss of syntactic information. Dependency and statistical parsers can be used to check for grammatical correctness, but according to Jonnalagadda & Gonzalez (as cited in Siddharthan, 2006); it is very difficult to develop an unsupervised system for determination of the grammatical correctness of a sentence.

3.5 Named Entity Recognition

3.5.1 Dictionary-based Approaches

Recognizing entities from unstructured text involves parsing the words in a given text and finding out entity names. Earlier systems were dictionary-based and required much effort and human intervention. Furthermore, such systems also presented problems when resources were changed such as dictionaries or entity evaluation rules (Tatar and Cicekli, 2009), but now they tend more towards a machine-based automatic learning approach. Most dictionary-based systems are domain-dependent and can recognize an entity if it is added to its reference-dictionary (Jorg, *et al.*, 2010), and hence failure in recognizing entities by a system predicts the failure in subsequent essential tasks of recognizing relation. However, both approaches have some advantages and disadvantages, so no one can completely overwhelm another one.

3.5.2 Rule-based Approaches

All entity recognition systems require some pre-defined rules, on the basis of which relations are evaluated from the given text. Earlier systems were based on a hand-crafted rule-based approach (Jonnalagadda, *et al.*, 2010) in which all the rules to be followed need to be fed into the system manually. Most rule-based systems follow Natural Language Parsing (NLP) techniques and their output is based on combined semantic as well as syntactic implications (Sharma, *et al.*, 2010). But rules must be pre-defined, which requires lots of domain expertise, human effort and time in design and implementation. This approach gradually revolutionized towards supervised (Jonnalagadda, *et al.*, 2010), Semi-Supervised and unsupervised machine-learning approaches. Systems following a supervised approach contain a training module which takes annotated data as a sample and then learns rules from it. Such systems need to be trained by significant number and variety of true and false examples to enable the system to extract the required information.

3.5.3 Dictionary-based Versus Rule-based Approaches

In a dictionary-based approach, a reference dictionary can contain other related informational fields for an entity, so if one such entity is found successfully, other information can also be easily related to it, *e.g.* in medical sciences it is important to have protein names along with allocated unique IDs for making them useful in other information sources, but such a dictionary-based approach has two main flaws. One is that it fails to identify entities if they are short named (Tsuruoka & Tsujii, 2003), which causes a low precision level. Another shortcoming is that it cannot recognize miss-spelled names, causing a lower number of successful attempts in entity recognition (Tsuruoka & Tsujii, 2003), *e.g.* if '*School of Bio-Medical Sciences*' is written as '*school of Biomedical Science*' or '*School of medical science*' then the dictionary-based systems will be unable to recognize all the variants unless all are added to a reference dictionary.

The first problem can be solved by a machine-based entity recognition approach, in which intelligent programs called classifiers work with high precision; such classifiers are trained by real-world corpus, but machine learning systems cannot relate any associated field such as IDs in proteins' case (Tsuruoka & Tsujii, 2003), as discussed above. The solution to the second problem is to make such tools more flexible so that variants of a name can also be recognized. It can be achieved by the use of regular expressions. Another solution is to calculate the number of deletions, additions, and editing operations required to convert entities found in text into their variants found in reference dictionary (Tsuruoka & Tsujii, 2003).

In order to compensate for the disadvantages of both approaches as discussed above, Tsuruoka and Tsujii (2003) divided the task of recognizing entity names into two phases:

- Recognition phase: in this phase, a dictionary-based approach is used to recognize the words or terms that are most likely to be entities.
- Filtering phase: Results of the recognition phase are further refined to get more accurate results. Here we make the application of classifiers of a machine-based approach that is trained to intelligently spot out named entities while leaving behind all false recognition.

3.5.4 Probabilistic Approaches

Tatar and Cicekli (2009) made use of the modified form of the N-Gram technique along with the automatic rule-based learning method for extracting protein names from biomedical articles. They used a probabilistic approach for relating one or two tokens of given text and called them Uni-Grams or Bi-Grams respectively. According to this approach, we analyze each token from the given text to find out:

- Its probability of being an entity in whole, because some names are represented by just one-word.
- How much it is likely to be the first word of an entity name.
- The probability of it being the last word of an entity name.
- The likelihood of a token being an important element of an entity name.

The first three measures are related to probabilities which are Uni-Gram in nature, while the last one is the measure of Bi-Gram. We can express this Bi-Gram model in notational form as follows:

$$P_{Entity}(T_1, T_2, T_{3\,,\ldots,} T_n) = \begin{bmatrix} P_{single} \\ P_{first}, P_{part}\,(T_2|T_1), \ldots, P_{part}(T_i|T_{i-1}), P_{last} \end{bmatrix} \quad \ldots\ldots\ldots\ldots (3.1)$$

The statistical model used for SIFT (Miller, *et al.*, 1998) is based on a probabilistic approach for finding out the probability of a whole parse tree, which is obtained by multiplying the probabilities of individual elements. We can express it in mathematical notations as:

$$P(T) = \prod_{C \in T} C|F \ldots\ldots\ldots\ldots\ldots\ldots (3.2)$$

Here, P= Probability,
T = Parsed Tree,
F = Conditional Factor, and
C = All Tree Components such as words, Tags etc.

Hakenberg, *et al.* (2010) also used a technique called Conditional Random Fields based on the probabilistic approach to find out protein names, in which all words which are likely to be protein names are assigned a probability value and those with the highest value are recognized as an entity name and labelled.

3.5.5 ONER

This tool is developed specifically for extracting organization names from free text and has shown the F-measure to reach up to 99.6% (Jonnalagadda, *et al.*, 2010). As most of the relations found from unstructured text such as web pages associates some commonly found entities such as people, locations and organizations, ONER is considered an important tool in recognizing such entities and extracting information related to it. It is being used by government agencies to find out the leading contributing groups and research centers and their particular domain of research (Jonnalagadda *et al.*, 2010). The system is also helpful in connecting research organizations with industrial groups who need to use specific informational pieces from large collection of text files. Their requirement can be fulfilled by a relation extraction system which stores specific information from unstructured text into database field values for future querying and informational retrieval.

ONER uses a multi-layered supervised rule-based approach and also makes use of lexicons for information extraction (Jonnalagadda *et al.*, 2010). First, it replaces all acronyms with their full forms. It then tries to recognize entities and understand its surrounding context using dictionaries and pre-defined rules *e.g.* if a word is followed by any organizational key word such as a "Group of Companies" or a person's name followed by a scientific term, then it is considered an organization.

This approach cannot be fully reliable. Consider an organization name which starts with some location name, *e.g.* "Qatar Airways". Qatar is extracted as a country, leaving behind just the word "Airways".

3.5.6 IDENTIFINDER™

IDENTIFINDER™ (Bikel *et al.*, 1997) is a Named Entity Recognition System implementing a named Entity task defined in MUC-7 (Marsh, 1998). It is trained to identify a number of entities called classes from text using the HMM Model (Miller *et al.*, 1998).

Each individual Entity is represented by a node of HMM or Hidden Markov Models (Freitag & McCallum, 2000) and is called here a Region. Each state of the model falls into one of the regions. Each input is compared with all regions. One default region is also defined here as Not-an-Entity, which contains the input which does not fall into any of other entity regions. It implies that:

$$Count(R) = C_E + 1 \ldots\ldots\ldots\ldots\ldots\ldots (3.3)$$

Here, R is total number of Regions, C_E is number of entity classes, and 1 is added for a Not-an-Entity class.

3.6 Relation Extraction

3.6.1 Co-occurrence Approach

According to the co-occurrence approach (Sharma, *et al.*, 2010), if two entities seem to co-occur frequently and are not separated by large context, they are likely to be related to each other (Coulet, *et al.*, 2010). However, it cannot confirm or estimate what kind of relation they will have. So we use a number of tools and techniques to accurately extract such relations. The co-occurrence approach can be applied to only local relations, but if a relation is between any two entities occurring in more than one sentence, it will no longer be applicable.

Rosenfeld and Feldman (2006) developed a system called Unsupervised Relation Identification and Extraction System (URIES) for identifying binary relations from large collections of text using a Unsupervised Relation Identification (URI) technique, based on the co-occurrence approach. During the self-learning stage, URI adds all entity pairs into a table of candidate relations which co-occur in the same sentence frequently.

The co-occurrence approach was extended by another approach called the Link-based approach (Gordon & Lindsay, 1999; Hristovski, *et al.*, 2005). According to this approach, if two entities co-occur frequently with some common entity, they are likely to be associated by a relation. For example, if entity A occurs with entity B, and entity C also occurs with entity B, then A and C entities are likely to be related to each other.

3.6.2 Verb-centric Approach

Feldman proposed the 'Noun-Phrase | Verb |Noun-Phrase' (NVN) template (Sharma, *et al.*, 2010) for extracting relations between any two entities. One flaw in this approach is that if one of the noun phrases are missing then the sentence is not considered to contain any relation. So a modified rule-based approach based on 'Subject| Verb |Object' template, overcoming the flaws of NVN was proposed by Lin and Pantel (2001). It states that find out the main verb phrase first and then find out the Subject and Object fillers of it.

Sharma and *et al.* (2010) proposed the verb-centric approach based algorithmic solution (figure 3.2) to the relation extraction problem. The algorithm used by Sharma *et al.* (2010) can recognize local and binary relations and requires a preset verb list. The valid sentences for the algorithm must at least contain a single entity and a main verb. The algorithm consists of three important steps, *i.e.* entity tagging, main verb identification and verb itemization.

3.6.2.1 Entity tagging

The first step is identifying and tagging all entities in a given sentence. If a whole sentence is recognized as an entity by entity tagger, it is thrown out of the list of valid sentences. Also, sentences with overlapping or nested entities are discarded, *e.g.* The Duke of *Edinburgh*. Here, *Edinburgh* itself is also an entity.

3.6.2.2 Main verb identification

All the valid sentences retained by the entity tagging step are checked for verb occurrences. Verbs are recognized through a pre-set word list containing all main verbs. There are a number of existing verb lists which can be used here such as Wordnet (Miller, 1995) and Verbnet (Meyers, 2005).

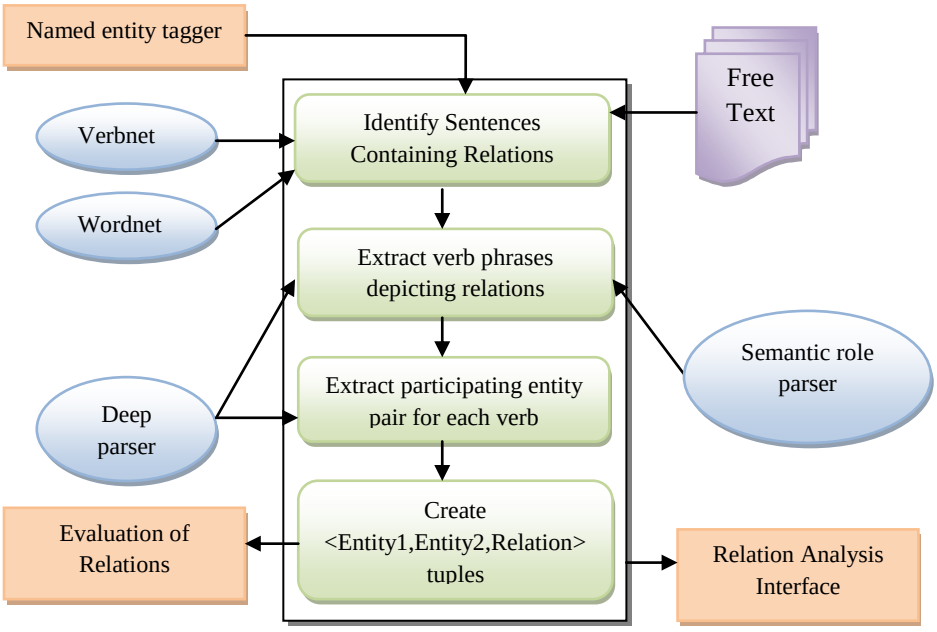

Figure 3.2: Overview of the Verb-Centric Algorithm proposed by Sharma *et al.* (2010)

3.6.2.3 Verb itemization

The algorithm can recognize binary relations containing one main verb and an entity pair, but a sentence written in natural language text may contain complex relations with more than one main verb and a pair of entities. Most sentences containing conjunctive structures cannot be easily analyzed for relation extraction. These conjunctive structures are recognized by using external parsers such as OpenNLP (Stoyanov, 2010). The parse tree output by parser is then traversed in pre-order form until the deepest node labelled by a main verb is found. When this deepest verb is found, the entities found to the immediate left and right are associated with the main verb to chunk and extract the text segment containing both the entities and associated verb. This process is repeated recursively until all chunked text segments contain just single main verbs and a pair of entities. Finally, all the main verbs along with their associated entities are extracted as a relation tuple.

One advantageous feature of the verb-centric approach is that it can evaluate the semantic implications from context to figure out the missing entity of a relation by identifying the semantic role of each noun phrase. Another feature is that it breaks up a complex One-to-many or Many-to-one relation into simpler binary relations, and hence each relation can be extracted individually. But its high dependency over main verbs limits itself and fails to evaluate relations from those phrases, clauses or sentences. For example:

1. "*S. Brisbane, the fourth public editor of New York Times resigned. . . .*" and
2. "*Hans Albert Einstein was a Swiss-American engineer"*

In the above examples, main verbs do not play any role in finding out the relations between entity pairs such as <*S.Brisbane, New York Times*> and <*Hans Albert Einstein, America*>.

3.6.3 DIPRE

DIPRE (Dual Iterative Pattern Relation Extraction) is a semantic bootstrapping technique proposed by Surgery Brin (1998) that was initially used to extract <author,title> relation but can be generalized to find any binary relation and can even be extended to obtain any n-ary relation. It takes some sample seed tuples initially and finds all occurrences of such entity pairs from web documents. When found, it learns new patterns from occurrences of such seed tuples and extracts new tuples, matching these learned patterns and so on, *e.g.* we input a small number of <Employee, Employer> sample pairs (associated by an organization) along with a URL where such relations occur. Using such URLs, it searches for text portions containing given <Employee, Employer> pairs and learns a number of patterns. Using these patterns, it creeps to other pages of the same domain and extracts entity pairs and so on. Figure 3.3 shows a flowchart diagram of this approach.

As stated by Brin(1998), Pattern generation is the most critical stage of the technique as it may impinge on all successive stages and iterations because bogus patterns will result in the discovery of many false positives to be fed into subsequent iterations. Therefore patterns need to maintain high coverage while retaining low error rate. So a pattern structure should contain more information than an entity pair and the surrounding context. Brin (1998) defined the pattern structure P as a 7-tuple:

$$P = [E_1, E_2, \text{Order}, C_{\text{Left}}, C_{\text{Middle}}, C_{\text{Right}}, \text{URL}]$$

Here, E_1 and E_2 are two entities associated by some relation. Order is the order of occurrence of entities *i.e.* whether entity E_1 occurs before E_2 or vice versa. C_{Middle} is a string between E_1 and E_2. C_{Left} is the longest common character sequence of left context of all occurrences up to maximum number of characters N preceding E_1 (or E_2, if it occurs before E_1). C_{Right} is the longest common character sequence in right context up to maximum N number of characters following E_2 (or E_1, if it occurs after E_2). URL is the longest matching prefix of URLs of all the occurrences of E_1 and E_2.

To illustrate this, consider the following sentence:

"A famous narrative poem Venus and Adonis was written by William Shakespeare"

Using Brin's defined pattern structure, it can be written as:

[*Venus and Adonis*, *William Shakespeare*, 1, 'A famous narrative poem', 'was written by', ' ', 'http://shakespeareauthorship.com/howdowe.html']

One advantage of this approach is that it is a semi-supervised approach, and unlike supervised approaches it does not use any tagging and classification, so it does not need any annotated corpora. As this technique works by identifying the duality between the current pattern that is used for searching and target relations to learn new patterns, it may not analyze the given text semantically to differentiate the contexts. To understand this we compare the context of following two sentences:

Example-1: "James serves as a manager in Simmons"

Example-2: "Core i3 CPUs serves as a backbone in Intel Computers"

Example-1 shows <Employee, Organization> relation while example-2 emphasizes the important role of one subpart in a system. A comparison done by pattern matching through regular expressions will evaluate both the sentences to be similar, which is not a fact. This may be because of some overly general patterns. To solve this problem, Brin proposed a pattern scoring scheme to refine to get reliable patterns

out of all generated patterns. According to that scheme, the confidence score of pattern P can be calculated as:

$$\text{Conf}(P) = |P.\ C_{Left}| \,.\, |P.\ C_{Middle}| \,.\, |P.\ C_{Right}| \ldots\ldots\ldots\ldots (3.4)$$

Here, | S | denotes the length of string S. If a pattern has a score less than some preset threshold, it is thrown out of a pattern list. However, the pattern scoring strategy increased the precision of the technique but still allows room for bogus patterns to be inserted into a reliable patterns' list.

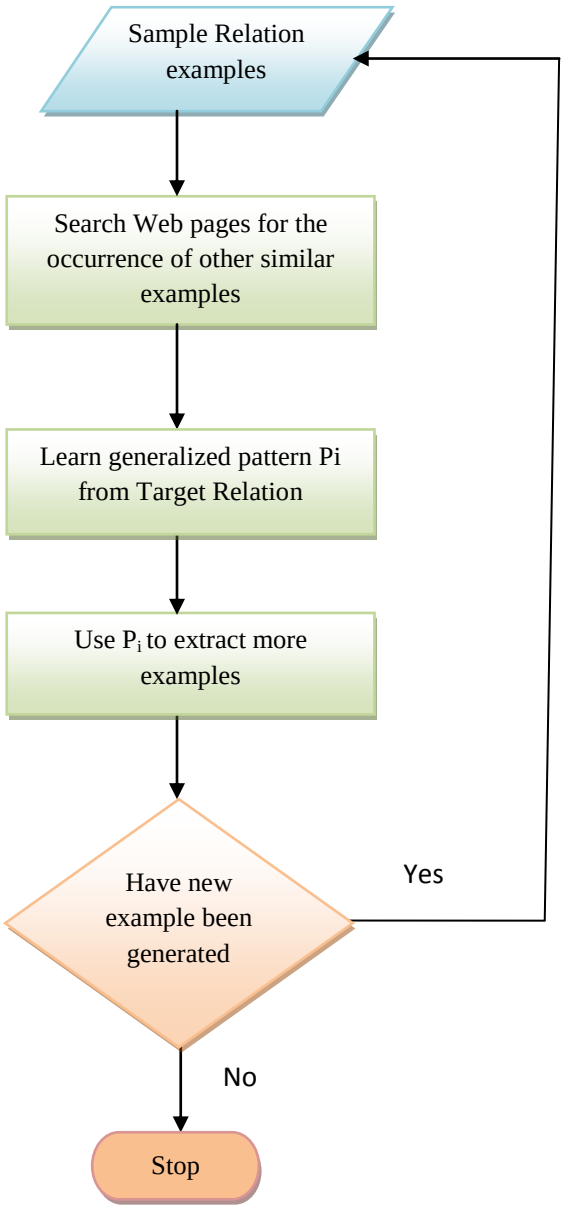

Figure 3.3: Flow Chart of DIPRE technique proposed by Brin (1998)

One solution to this pattern generality problem was proposed by pattern Hu and *et al.* (2007) through a system called Semantic Pattern Getter (SPG). It generates pattern text segments containing seed tuples using the following three steps:

1. Find text segments containing seed tuples and convert them into 5-tuple pattern representations, *i.e.* both the entities and their surrounding context.
2. Instead of finding a single longest common sequence of characters from all contexts, it finds all common subsequences of characters using a bi-sequence alignment algorithm (Sammeth, *et al.*, 2003).
3. Patterns are further refined using the validation rules specified by Hu *et al.* (2007).

SPG also introduced a new scoring scheme to get patterns with a high level of accuracy and coverage. If $P_{positive}$ and $P_{negative}$ denotes the number of positive and negative tuples respectively generated by pattern P in the current iteration and P_{new} denotes the total number of newly generated tuples by P.

$$Conf(P) = [log_2(P_{new} + 1)]^{\beta} \times \left[\frac{P_{positive}}{P_{positive+P_{negative}}}\right] \ldots\ldots\ldots (3.5)$$

Here, β represents a percentage of correct tuples extracted from the total number of patterns generated.

Since DIPRE makes heavy use of HTML tags while extracting relations, it cannot be applied to plain text files which do not contain any tags, *e.g.* Wall street journal's news articles. Bobrovnikoff (2000) presented cluster-based extension to it to adapt it to work on any kind of unstructured text without relying on these visual formatting tags.

Brin's proposed technique divided the pattern groups into sub-groups using first character following common URL strings of the group which are not applicable for offline corpus. Bobrovnikoff (2000)'s extended adaptation of the technique uses the middle context of the patterns as a clustering attribute. It means patterns sharing common middle context are clustered to together.

3.6.4 Context-based Clustering Approach

This is an unsupervised approach for discovering relations from a large amount of unstructured text (Hasegawa *et al.*, 2004). In this technique, if the same types of entities occur in a similar context; they are clustered to together and each newly generated cluster is then named by the most frequent word of cluster context. This technique consists of three main steps: entity tagging, context vector creation, clustering and cluster labeling.

3.6.4.1 Named entity tagging

A tagger identifies and tags each entity according to its type. To facilitate the task of relation extraction, entities can be further categorized into sub-classes by using extended named entity taggers, *e.g.*

"Microsoft has agreed to buy Internet telephone company Skype"

would be outputted by tagger as

"<ORG>Microsoft</ORG> has agreed to buy Internet telephone company <ORG>Skype</ORG>".

3.6.4.2 Context vector creation

Here, entities are paired and considered to be associated to each other using a co-occurrence approach (section 3.6.1). Once these co-occurring entity pairs are formed, the context of each entity pair is converted into its vector representation using a bag-of-words model (Joachims, 1996). In this model, a text string is converted into distinct words where each word is represented by a unique index and is associated with a unique weight which is calculated by some weighting scheme. Although representing a string into bag-of-words model is simple, it loses information about a sequence of words in the string.

The weighting scheme used by Hasegawa *et al.*(2004) is tf * idf (Salton and McGill, 1983), which is the inner product of term frequency (tf) in its context and the inverse document frequency (idf).

3.6.4.3 Entity pair clustering

Clustering (Hastie *et al.*, 2001) can be defined as a process of grouping together objects with similar properties in such as way that all the objects in a cluster have same properties, whereas objects of two different clusters have different properties. Objects can be clustered using either a hierarchical or non-hierarchical method (William & Ricardo, 1992).

In the hierarchical method, successive clusters are created and nested clustering can also be performed inside a cluster, whereas in the non-hierarchical clustering method, nesting clusters are not created but rather m clusters are divided into n fixed groups. Single-link (Florek *et al.*, 1951; Smeath, 1957; Johnson, 1971), and complete-link (Williams, 1967; Lance,1967) algorithms are examples of hierarchical clustering, whereas single-pass (William & Ricardo, 1992) clustering is an example of non-hierarchical clustering.

Hasegawa *et al.* used the hierarchical clustering method for clustering the similar context vectors. Similarities between the context vectors of any two entity pairs are calculated by the cosine similarity measure, which is a robust method of calculating the similarity between vectors (Tata & Patel, 2007).

3.6.4.4 Cluster labeling

Each cluster is assigned a label that must be representative of the cluster, and gives an idea about which type of relation the cluster members are instances of. To carry out this, a label is constituted by combining the most frequent words in the context.

Labeling the clusters may be difficult if the contexts of two different clusters share common terms, and if the labels chosen for both the terms are exactly identical then these clusters will no longer be distinguishable.

The main advantage of this approach is that it does not require any annotated corpora or labeled examples like weakly-supervised and supervised approaches, but because there is no user intervention therefore systems implementing techniques based on unsupervised approaches have less precision than those based on supervised or semi-supervised techniques.

3.6.5 SIFT

SIFT (Statistics for Information from Text) is based on the Statistical language model (Miller et al., 1998) and was first developed for implementing Named Entity and Template Relation tasks described in MUC-7 (Marsh, 1998). It successfully replaced the existing hand-crafted rule-based system named PLUM (Miller *et al.*, 1998).

SIFT is not based on a task allocation strategy in which tasks are clearly distributed, but is rather an integrated system as its model is based on a unified approach (Miller *et al.*, 1998). This can be advantageous when it is very ambiguous to get information at one stage while it is under process at another stage, *e.g.* some contents can be better analyzed by the semantic analyzing model while others needs POS taggers. So if a system has an integrated module for all tasks, all parts can work in parallel rather than waiting for an earlier element to perform its task, but this may make it difficult to debug or modify.

The system analyzes the given text in two phases. In the first phase, it tries to find out all entities present in the text and local (sentence-level) relations. In the second phase, it looks for all non-local relations (paragraph-level, document-level etc.), co-references, multiple occurrences and different variants of the entities. It works by analyzing the given text in two ways:

- **Semantic Analysis**, which involves looking into the context in which two entities occur and evaluating the relation between them.
- **Syntactic Analysis**, in which language rules are applied over the input text to analyze its structure.

One advantageous feature of this system is that if a system is trained to recognize the relations related to one domain, it can be shifted to another domain by just adding or replacing semantic annotations. Syntactic annotations can work as long as the language of the input text is same. Figure 3.4 shows a block diagram representation of training and re-using the system. Here, the system is trained with a new semantic annotation drawn from some sample text. We do not need to re-train the system with syntactic annotations to adapt the system to work on any other domain. The output is a combination of syntactic and semantic validations.

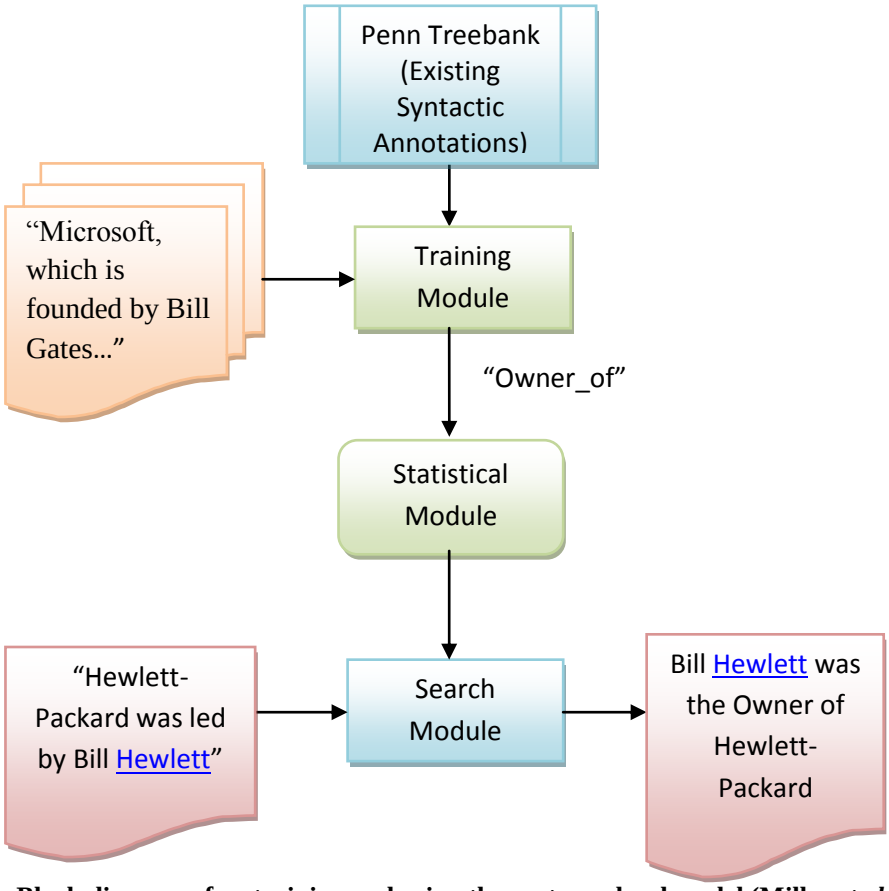

Figure 3.4: Block diagram of re-training and using the sentence-level model (Miller *et al.*, 1998)

3.6.6 Comparison of techniques

As shown in the reviewed literature presented in this chapter, the concept of automating information extraction (including relation extraction) was first initiated in MUC followed by ACE. Different techniques were developed to implement the tasks defined in MUC and ACE including Verb-Centric, Context-based clustering, SIFT and DIPRE. The verb-centric approach relies on main verbs but can handle complex sentences, and the sentences with co-references. Clustering approach does not require huge annotated corpora so it best fits for large corpora but its precision is not so high. SIFT can process sentences with ambiguous relations by analyzing them semantically as well as syntactically but its semantic analysis is domain-specific. DIPRE only needs some sample seeds to learn and extend extraction patterns, but its pattern matching technique may fail to extract relations accurately. These shortcomings are fulfilled by a weakly-supervised bootstrapping technique called Snowball (chapter 4), which is based on DIPRE.

3.7 Summary

In this chapter, we have reviewed a number of tools and approaches to relation extraction from unstructured text. Some of them proved to be more successful than others. Some were domain-specific while others can extract common relations from text. Here we have reviewed the literature related to overall process of relation extraction along with some brief history. We have seen that adding some optional tasks helped increase correctness, recall, and precision and hence the f-measure of the whole process, *e.g.* applying automatic methods for sentence simplification has shown recall to be improved by 8% without losing persistency in precision.

We discussed three main Named Entity Recognition approaches, *i.e.* rule-based, dictionary-based and probabilistic with their main distinguishing features. We have also compared rule-based with the dictionary-based approach and provided a hybrid solution to the disadvantages of both approaches.

The chapter also discussed different existing approaches proposed for relation extraction *i.e.* Verb-Centric, Context-based clustering, SIFT and DIPRE, with their main features and disadvantages also. Comparison showed the advantages and disadvantages of the reviewed approaches relative to each other. However, none of the techniques discussed in the chapter perfectly fit with our requirements so we are intended to use snowball (discussed in chapter 4) as a framework for our system design which has relatively high precision and recall than other techniques. The next chapter will present design and implementation details of our system.

Chapter 4

Project Design and Implementation

4.1 Introduction

The chapter presents a detailed description of overall design and implementation of our solution to relations extraction problem. The technique chosen to skeleton the system is the Snowball (Agichtein and Gravano, 2000), which requires minimal human supervision. As the technique is intended to be used to extract relations from an online corpus constituted by a number of web documents, a number of variations are required inevitably at different steps of the technique; these are also discussed in the chapter.

The chapter starts with the introduction of the technique used to design our system. It also presents the system overview and steps followed to implement the chosen technique. Each step is elaborated thoroughly along with its algorithmic representation and implementation details.

4.2 Snowball

In chapter 3 we studied a number of existing attempts at solving the problem framed in section 2.10. The comparative study had shown that DIPRE (Brin, 1998) was the only technique originally designed and tested on www documents but it is not flexible enough to get high coverage in capturing every mention of a relation. It also has a high risk of producing fake patterns and may possibly provide results with bogus entity pairs. To overcome the disadvantages of DIPRE (Brin, 1998), another technique was proposed as a modified and restricted form of DIPRE (Brin, 1998) by Agichtein and Gravano (2000) and is called 'Snowball' (Agichtein and Gravano, 2000).

Snowball (Agichtein & Gravano, 2000) is a weakly-supervised technique for relation extraction with more effective strategies for pattern generation and tuple extraction than DIPRE (Brin, 1998). The risk of generating incorrect tuples is reduced by adding-up advanced tactics for evaluating every generated tuple and pattern. This is the reason that DIPRE has a high probability of losing precision and recall with every iteration, whereas Snowball doesn't deviate from its original performance, which persists mostly until the last iteration. Another difference that makes Snowball preferable over DIPRE is that it uses entity tagging and a similar function rather than hard pattern matching which helps greatly in increasing coverage while reducing the number of false-positives in resulting tuples.

Snowball uses the bootstrapping approach proposed by Yarowsky (1995). However, Yarowsky(1995) primarily applied his proposed bootstrapping approach for solving word-sense disambiguation problems, but its use as a general framework by Snowball (Agichtein & Gravano, 2000) and other weakly-supervised techniques proved that it can be adopted for information extraction purposes.

Snowball, as shown in figure 4.1, is a domain-independent solution to the relation extraction problem, and can easily be adopted to extract domain-specific relations such as Protein-Gene or Protein-Protein relation from the bio-medical sciences domain. Like DIPRE, it also doesn't require annotated corpus or any other human intervention other than a few entity pairs, linked by some relation. Such tuples are then used by learning patterns and extracting tuples. After evaluating reliability of patterns and tuples created by the

first iteration, the refined patterns are then used to be fed into the next iteration and so on. It keeps on iterating until no more enough reliable tuples are generated by the current iteration.

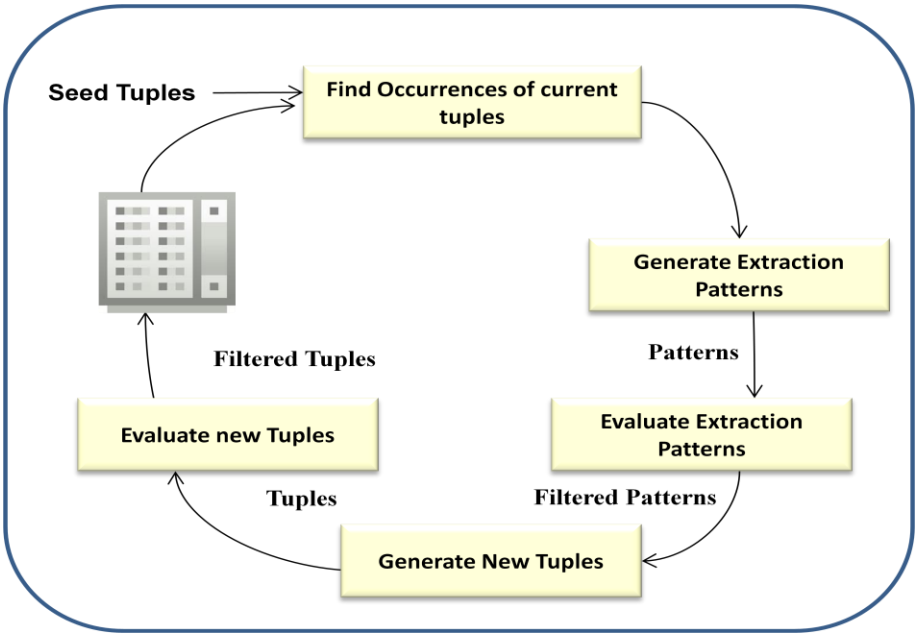

Figure 4.1: basic architecture of Snowball

4.3 System overview

A general, overall project architecture is shown in figure 4.2. I t helps with the conceptualization and working mechanism of the overall system. It also helps in understanding the contribution of each module to overall system functionality and its interaction with other modules of the system. It is worth noting here that the system is intended to work in binary and local relations. Snowball (discussed in section 4.1) is taken as a skeletal technique for designing the system. Snowball was originally designed and tested on hand-made corpus written in plain text. However, our system is intended to extract relations from the web. So the system needs to crawl to and scrap text from the URLs, given by the user or alternatively taken from a standard Google search.

The algorithmic representation of the system is as follows:

1. Preprocess the Corpus:
    a. Enter a few seed tuples. Here, all tuples are entity pairs associated by instances of the same relation.
    b. Find out text segments (*i.e.* sentences) containing any of the seed tuples, with entities within certain proximity in the text.
    c. Tag entities with their types in the sentences found in the previous step.
2. Generate patterns from the text segments and generalize those patterns by clustering similar patterns using a single-pass algorithm.

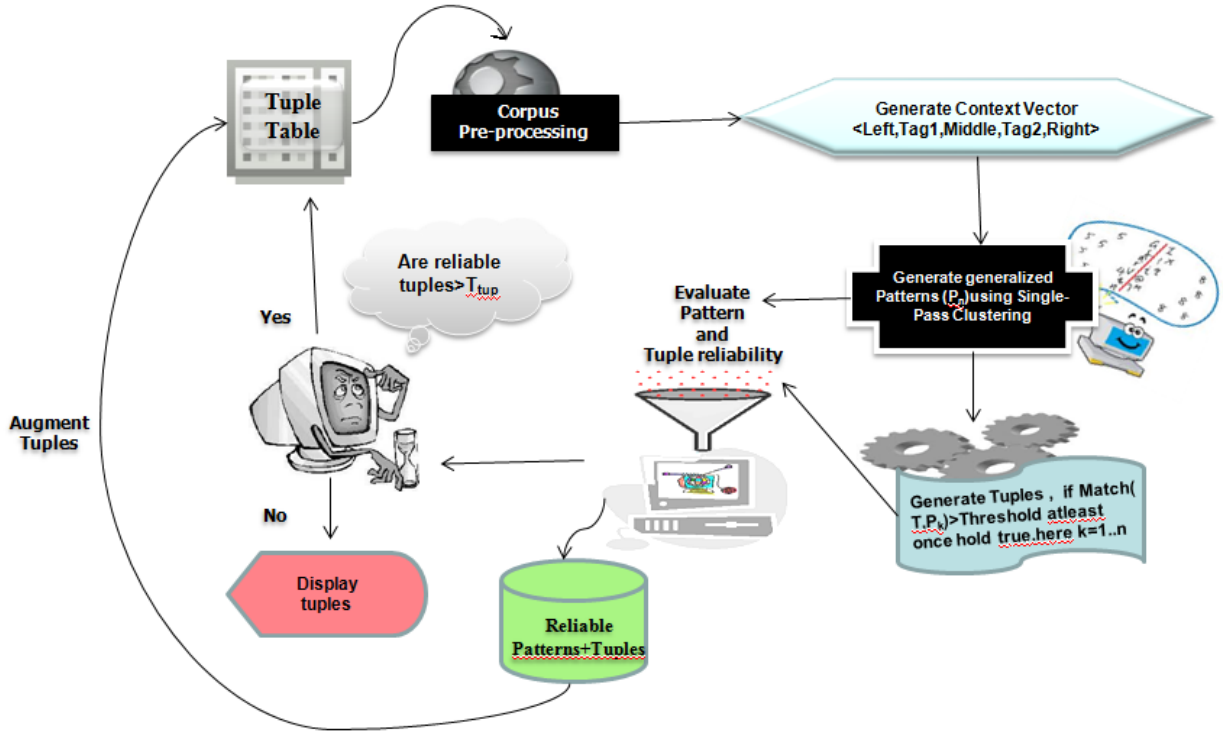

Figure 4.2: Overall architecture of our system

3. Generate Tuples:

    a. Find sentences from the same corpus containing the same entity types as those of any of the seed tuples.
    b. Construct vector representation S=<Left, entity1, middle, $entity_2$, right> from the contexts of each sentence sorted out in above sub-step a.
    c. Compare each S with the generalized patterns $P_{(1..n)}$.
    d. If Similarity(S, P_k) >= preset threshold with k=1.2…,n holds true at least once, tuples are generated.

4. Refine reliable patterns by evaluating all the generalized patterns generated in step 4.
5. Evaluate tuples to sort out tuples with enough confidence to be fed into the next iteration.
6. Move back to step 2, if the number of reliable tuples generated in step 7 is >= preset threshold
7. Output all generated tuples and Exit.

4.4 Corpus Pre-processing

As stated earlier, the system is intended to be used to extract entities and their relations from web documents rather than from hand-crafted corpus consisting of locally loaded offline plain-text files. So this step is different and more complicated than Snowball's original corpus processing step. The algorithmic representation of the step and sample output of the pre-processed corpus is shown in figures 4.3 and 4.5 respectively.

```
Algorithm preprocessCorpus ( )

For   i  := 1 to 5
    Input seed tuples and (optionally) a URL
    Add each entry into SeedTuple_Table
End For Loop

URL_List := CorpusURLsCollection( SeedTuple_Table )

 For each tuple in SeedTuple_Table
    For each URL in URL_List
       For each sentence in Text (URL)
            If  exist (tuple.entity1, tuple.entity2, sentence)
                If   characeterDistance(tuple.entity1,tuple.entity2, sentence) <=500
                          sentence := TagEntities(sentence)
                          Add sentence to valid_sentences
                End If
            End If
       End For
    End For
 End For

 Return valid_sentences
```

Figure 4.3: An algorithmic representation of corpus-preprocessing

4.4.1 Selection of URLS

Since the system is anticipated to work on online corpus, it is impossible to navigate every document present on the World Wide Web(WWW) to search for occurrences of the entity pairs inputted by the user. According to McEnery and Wilson (1996), a corpus must be finite in size. Infinite-sized corpus may rapidly eat up all available computing resources. Moreover, every web page may need to be processed more than once, *i.e.* during learning patterns, creating tuples from those patterns. Keeping in view all these factors, URLs to be navigated by the system must be limited.

In order to generate finite corpus, an algorithmic module named 'CorpusURLSCollection', as shown in figure 4.4, has been formulated that is used to create a list of all the URLs of web documents, which contain seed tuples. The module starts with user input of a few seed tuples up to maximum limit of five. Each user entry consists of an entity pair associated by a relation instance and optionally a URL of a web document which contains the text representing such relation instance. If a URL is not provided with any of the user entries, the system uses its default setting of standard Google search and fills the empty URL fields with a list of URLs of the first sixty-four results from the search. Finally, the URLs associated with

all user entries are added to make a single URL_List, which can then be used by other algorithmic modules for further processing.

```
Algorithm CorpusURLsCollection(SeedTuple_Table )

For each tuple from SeedTuple_Table
    If tuple.URL =NULL
        Resultant_List := SearchGoogle(tuple.entity1, tuple.entity2)
        Add Resultant_List to URL_List
    Else
        Add tuple.URL to URL_List
    End IF-Else
End For Loop

Return URL_List
```

Figure 4.4: Preparing a list of URLs containing seed tuples

4.4.2 Sentence Extraction

Once a list of all the useful URLs is generated; the next step is to extract text segments containing user inputted seed tuples from html files pointed to by any of these URLs. To carry this out, a sentence extracting module of the system navigates to the corresponding web document of each URL in the URL_List and scraps plain text from web documents. These sentences are filtered out by the following rules:

1. Given a seed tuple from a seed tuple table, there must at least be a single occurrence of each entity of the seed tuple in a valid sentence.

2. Once entities are matched with any of the seed tuples, the distance between such entity pair is measured in the number of characters. Sentences are not validated if the entities occur beyond than maximum allowable distance. Here a length of 500 characters is chosen as the maximum length.

4.4.3 Entity Tagging

The filtered sentences are then tagged by an entity tagger. Stanford named entity tagger (Finkel, *et al.*,2005) is used here with the CRF (Sutton & McCallum, 2006) classifying model, which is trained to annotate three types of entities, *i.e.* person, organization and location in given text by standard xml-like tags as shown in figure 4.5. The examples in the figure show that the tags used to annotate entities are:

- <PERSON>…</PERSON>
- <ORGANIZATION>…</ORGANIZATION>
- <LOCATION>…</LOCATION>

<ORGANIZATION>Microsoft</ORGANIZATION> remained more valuable than<ORGANIZATION>IBM</ORGANIZATION> after the stock market closed on Tuesday.

<PERSON>Bill Gates</PERSON> parlayed that breakthrough into industry dominance by the end of 1999, <ORGANIZATION>Microsoft</ORGANIZATION>'s market value was three times that of <ORGANIZATION>IBM</ORGANIZATION>'s, and bigger than any other <LOCATION>U.S.</LOCATION> company.

Figure 4.5: A sample of pre-processed corpus extracted using seed tuple<Microsoft,IBM>

4.5 Pattern generation

This is the most critical and time-consuming step to be implemented in the overall process because generated patterns are then used by the subsequent step to extract tuples. The sentences filtered out through the pre-processing step are converted into their 5-tuple pattern representation. Each newly generated pattern is then matched with existing patterns by calculating their context similarity score (section 4.4.3). Similar patterns are finally clustered together to form generalized patterns (section 4.4.4), which are then used by the next step to extract tuples. Figure 4.6 shows an algorithm for producing generalized patterns from a list of valid sentences created by the algorithm illustrated in figure 4.3

```
Algorithm    createGeneralizedPatterns (valid_sentences)

Cluster_list  : = {   }, Generalized_patterns : = {   }

For each Sentence in valid_sentences
    P : = generate5-Tuple(Sentence)
    clusterPattern(P, Cluster_list)
End For


For each Cluster in Cluster_list
     Add  Cluster.centroid to Generalized_patterns
End For


Return Generalized_patterns
```

Figure 4.6: An algorithm for producing generalized patterns from a valid sentence list

4.5.1 Pattern Representation

Each sentence in a validated corpus is converted into its pattern representation. Consider sentence **S** containing an entity pair <$entity_1$, $entity_2$> annotated with tags < $entity\_tag_1$, $entity\_tag_2$> respectively by the entity tagger. A pattern representation of **S** is a five-tuple segmentation and can be written as:

$$P(S) = < left, entity\_tag_1, middle, entity\_tag_2, right >\ldots\ldots\ldots(4.1)$$

Where left, middle and right are context vectors (to be discussed later). An example depicting conversion of a text segment into its 5-tuple pattern representation is shown in figure 4.7. A pattern representation includes entity types in its 5-tuple segmentation, whereas DIPRE allows strings in place of entity tags and does not need to use any entity taggers. This is the reason that Snowball patterns are more restrictive in generating false-positives than those of DIPRE, and hence it overwhelms DIPRE in getting high precision and recall.

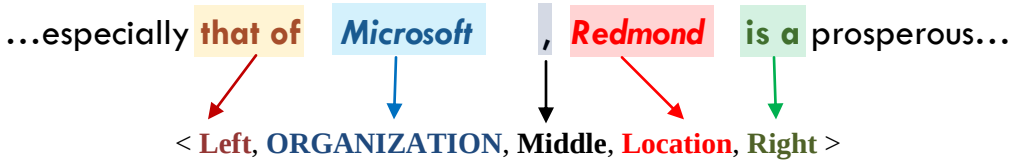

Figure 4.7: An example depicting 5-tuple representation and context chunking

4.5.2 Context-vector representation

The context around an entity pair is analyzed and is broken down into three parts, *i.e.* the left context which includes the terms before the first occurring entity, the middle context that holds the terms between the two entities, and the right context which contains the terms after last occurring entities. As it has been observed, a relation between entities is mentioned by terms close to entities, thus to process all the terms in all three contexts would be wastage of time and computational resources. Therefore a chunking window is used to trim away the distant terms which are less likely to evidence a relation. Any term falling outside of this chunking window is snipped away. For our system, a default window size is set to two terms to left and right contexts, but this can be changed easily by setting the value of an internal parameter. Mostly this is required when the system is shifted to some specific domain where relations occur mostly between distant entities. According to Agichtein and Gravano (2000), middle context is most important in capturing relations between entities so no clipping is done on middle context but rather during pre-processing. Only those sentences which have a middle context length not more than five-hundred characters are kept.

After performing window chunking, each context is converted into its vector representation. In order to do so, each term is separated and associated with a real-numbered value computed by a single weighting scheme. Here, Term Frequency (TF) is used as a weighting scheme which refers to the number of times a term appears within the corresponding context.

4.5.3 Degree of Match

Unlike DIPRE, Snowball does not do hard pattern matching while finding out similarities between any two 5-tuples. Instead, it allows for minor changes in corresponding contexts of the pattern representations being matched, such as difference in commas or any other punctuation mark *etc.* In order to measure the degree of match; consider any two 5-tupled pattern representations:

$$P_m = <L_1, t_{a1}, M_1, t_{a2,} R_1 >$$

and

$$P_n = <L_2, t_{b1}, M_2, t_{b2,} R_2 >$$

If the entity tags of both the patterns are not the same and/or not in the same order - which means if $\mathbf{t_{a1}} = \mathbf{t_{b1}}$ and $\mathbf{t_{a2}} = \mathbf{t_{b2}}$ does not hold true - their degree of match is considered as 0.0. Otherwise it is calculated as:

$$Match(Pm, Pn) = w_l \left[\frac{L_1 \,.\, L_2}{|L_1|\ |L_2|}\right] + w_m \left[\frac{M_1 \,.\, M_2}{|M_1|\ |M_2|}\right] + w_R \left[\frac{R_1 \,.\, R_2}{|R_1|\ |R_2|}\right] \ldots\ldots (4.2)$$

Here, w_l, w_m and w_r are scaling factors for left, middle and right contexts respectively. A fixed value between [0..1] is chosen to represent the relative importance of the context in the overall degree of match. As mentioned earlier in section 4.4.2, the middle context is most important in terms of capturing relations, so it is normally assigned a higher weight than the left and right contexts. As shown in equation 4.2; the left, middle and right contexts of P_m and P_n are converted to their vector representations, *i.e.* L_1, M_1, R_1 and L_2, M_2, R_2 respectively, and are matched correspondingly. It means here L_1 is matched with L_2, M_1 with M_2 and so on using cosine similarity (Frakes & Baeza-Yates, 1992) measure using equation 4.3.

$$\mathrm{COS}(S_1, S_2) = \frac{S_1 \,.\, S_2}{||S1||\ ||S2||} \ldots\ldots\ldots\ldots (4.3)$$

As depicted in equation 4.3, given any two context strings, the cosine similarity is a normalized dot product of their vector representation. So the resulting value is always normalized, *i.e.* between [0..1]. Here, we used a vector space model (Baeza-Yates & Ribeiro-Neto, 1999), which considers a string as a bag-of-terms. String similarity can also be measured by character-based technique, which involves finding out the number of insertions, deletions or substitutions required to form one string from another. The most popular character-based technique was proposed by Needleman and Wunsch (1970), in which strings to be compared are aligned in parallel first and sequences of mismatched characters called affine-gaps are calculated. The author didn't use this technique because character-based approaches work on a character-level, so they are less-accurate and incur more computational costs than those working on a string-level, *e.g.* cosine similarity measure.

After measuring the similarity of each context of pattern $P_m$with each corresponding context of the other pattern P_n, the resulting values are multiplied by their corresponding scaling weights and are added together to give the resulting degree of match between the given pattern representations.

4.5.4 Pattern Clustering

Although the natural language is flexible enough to present a relation in different ways, a single pattern is not enough to cover all instances of a relation, but if the same types of entities are bounded by the same relation type, they are mostly observed to be found in a similar context, *e.g.* the patterns: '<Person> is Prime minister of <Location>' and '<Person>, Prime minister of <Location>' represented by 5-tuple as:

< ' ', PERSON, 'is Prime minister of', LOCATION, ' '>

and

< ' ', PERSON, ', Prime minister of', LOCATION, ' '>

So patterns sharing enough common context terms are assembled together to form a group called cluster, and the process is called pattern clustering. Each cluster is represented by a central pattern called centroid. A centroid is the same 5-tuple pattern representation like other patterns but its context is formed by an intersection of corresponding contexts of all the patterns, *i.e.* the left context of centroid is formed by a collection of terms common to the left context of all the terms added to the cluster. For vector representation, each intersected term is assigned a weight that must be highest among all the weights for the term in left context of all the patterns of the cluster. The same process is repeated for forming the middle and right context of the centroid, *e.g.* the centroid for the above example would be:

< ' ', PERSON, 'Prime minister of', LOCATION, ' '>

Here, contexts are simply represented by strings rather than their vector representation, since the frequency of each term in its corresponding context is one. So the maximum weight that could be assigned to each intersected term will also be one.

```
Algorithm   clusterPattern (P, Cluster_list)

If Cluster_list is not EMPTY
   For each Cluster in Cluster_list
      If  Degree_of_Match ( P,  Cluster.centroid ) ≥ Threshold_sim
            Add pattern P to Cluster
            recalculateCentroid ( Cluster)
            Added  : =  true
      End If
   End For Loop

Else
   Added  := false
End If

If  Added = false
   Create a new Cluster and add pattern P to it
   Cluster.centroid  :=  P
   Add Cluster to Cluster_list
End If

Return Cluster_list
```

Figure 4.8: Single-pass clustering algorithm for pattern clustering

Patterns are clustered using a single-pass clustering approach (Frakes & Baeza-Yates, 1992). The single-pass algorithm used for clustering patterns is described in figure 4.8. It shows that a pattern is added to an existing cluster if it has degree of match greater than or equal to some preset threshold with the centroid of that pattern, otherwise a new cluster is generated for it.

4.6 Tuple Generation

Once generalized patterns are formed, the system is able to extract new tuples using these learned patterns. An algorithm used to implement this step is depicted in figure 4.9. To carry out the tuple generation step, the system first looks for the text segments containing any two properly tagged entities. The same document collection is used here, from where patterns were generated in the previous step. Each text segment is converted to 5-tuple representation (section 4.4.1) to enable its comparison with generalized patterns, which are also represented as 5-tuple. If the degree of match between a 5-tuple

representation of a text segment and any of the generalized patterns is greater than or equal to some pre-set threshold, then a tuple is generated. Here, each generated tuple is a binary relation R=<E_1, E_2> associating an entity pair E_1 and E_2 enclosed within the entity tags present in 5-tuple of the text segment.

```
Algorithm   extractTuples (Generalized_patterns, URL_List )

For each URL in URL_List
    For each sentence in Text (URL)
        If number_of_entites (sentence) ≥ 2
           Content  : = generate5-Tuple(sentence)
           MaxScore=0.0 , MaxScored_Pattern= NULL
            For each pattern in Generalized_patterns
                Current_Score:=  Degree_of_Match (pattern,content )
                      If MaxScore < Current_Score
                         MaxScore := Current_Score
                         MaxScored_Pattern := pattern
                      End If
            End For
        If  MaxScore  ≥  Threshold_sim
            Extract tuple <E1 ,E2> from sentence using content
            Add (<E1 ,E2>, MaxScore, pattern) to Candidate_Tuples
        End If
    End For
End For

Return Candidate_Tuples
```

Figure 4.9: An algorithmic module used for tuple extraction

A record of generating patterns along with a degree of match between the pattern and context of each generated tuple is kept for its later use during pattern and tuple evaluation phases. It is worth noting here that more than one pattern can generate the same tuple so a list of all the generating patterns must also be kept along with each tuple.

4.7 Pattern Evaluation

With Snowball being a bootstrapping algorithm, it self-sustains itself until trusted tuples can no longer be generated. Each iteration filters out a number of reliable patterns. These patterns are used to generate seed tuples which are used by the next iteration for generating new patterns and tuples. The quality of patterns heavily affects the statistics of reliable tuple generation. Two important considerations which are taken into account at each iteration are selectivity and coverage, while choosing patterns to be promoted to its subsequent iteration. Selectivity refers to our confidence in a pattern's ability in generating correct tuples. Whereas coverage refers to its ability to identify and extract all (or new) entity pairs (*i.e.* tuples), bounded by an instance of same relation type. Simply, Selectivity restricts the system from generating overly general patterns, whereas coverage balances it by enabling it to cover every mention of a relation. The algorithm used for implementing pattern evaluation is depicted in figure 4.10.

```
Algorithm  evaluatePatterns (Generalized_patterns, SeedTuple_Table, Candidate_Tuples)

For each pattern P in Generalized_patterns
  Positives : = 0, Negatives := 0
  For each candidate in Candidate_Tuples
     If   candidate. generatingpattern  = P
        If isPositive(candidate, SeedTuple_Table)
           Positives:= Positives+1
        Else if  isNegative(candidate, SeedTuple_Table)
           Negatives:= Negatives+1
        End If-Else
     End If
   End For Loop
   P.conf := Positives/( Positives+Negatives)
End For Loop

Return Generalized_patterns with confidence scores
```

Figure 4.10: Algorithm implemented to score generalized patterns

In order to restrict the system from generating incorrect tuples, each of the generated patterns is assigned a certain real-numbered weight. This weight is actually our confidence in the pattern that it will not generate any incorrect tuple. According to Agichtein and Gravano (2000), the confidence score is actually an estimation of the probability of a pattern to generate valid tuples. Each pattern is assigned a confidence score as a function of selectivity and/or coverage.

Here, the system uses a restraint-based pattern scoring strategy presented by Agichtein and Gravano (2000), which computes the confidence score of a pattern as a function of its selectivity, as the author prefers reliability over coverage:

$$Conf\ (P) = \frac{Positives\ (P)}{Positives\ (P) + Negatives(P)} \ldots\ldots\ldots\ldots (4.4)$$

Here, P is a pattern, whereas Positives (P) and Negatives (P) are the number of positive and negative tuples generated by the pattern.

4.7.1 Positive Match

If a pattern has generated a tuple $<e_1, e_2>$, there is a tuple $<e_1', e_2'>$ in a collection of seed tuples for which $e_1 = e_1'$ and $e_2 = e_2'$ holds true then the match is considered as positive.

4.7.2 Negative Match

A match is counted as negative if a tuple $<e_1, e_2>$ is generated by a pattern and there exists a $<e_1', e_2'>$ for which $e_1 = e_1'$ and $e_2 \neq e_2'$ hold true. It should be clear here that $e_1 \neq e_1'$ and $e_2 \neq e_2'$ is considered as do not care condition so it must not be counted as a negative match, but if $e_1 \neq e_1'$ and $e_2 = e_2'$ holds true then it is also counted as a negative match.

Consider a seed tuple table with tuples < Golda Meir, Israel> and < Julia Gillard, Australia> and a pattern P = <" ", PERSON, Prime Minister of, Location, " " >; which has extracted the tuples < Gordon Brown, United Kingdom>, < Edmund Barton, Australia> and < Golda Meir, Israel>. Then the confidence over P can be calculated as:

$$Conf(P) = \frac{1}{1+1} = 0.5$$

It shows that the tuple < Golda Meir, Israel> is counted as a positive match since an identical entity pair already exists in the seed tuple collection, whereas the tuple < Edmund Barton, Australia> is counted as a negative match because there is a seed tuple < Julia Gillard, Australia> with one entity difference, but the tuple < Gordon Brown, United Kingdom> is not counted at all because none of its entity matches to any entity of seed tuple at all, so it is not considered to contribute to measuring the confidence score of the pattern.

4.8 Tuple Evaluation

In order to learn new patterns, the system requires new seed tuples at each iteration. In the first iteration, a few seed tuples are provided by the user which is considered as the most reliable collection of seed tuples, but for all subsequent iterations, such user-provided seed tuple tables must be augmented with reliable tuples generated by the system. All the system-provided tuples cannot be relied on, and feeding incorrect tuples will cause low-confident patterns in the next iteration, which causes again incorrect tuples and so on. Hence, the system finally fails to extract even a single correct relation.

To filter out the tuples that have a reliability greater than some preset threshold, an evaluation strategy is needed that must be used to evaluate the newly generated tuples. Recall from the earlier discussion in section 4.5 that a record of all generating patterns for each tuple must be kept along with the degree of match between the generating pattern and the context of a generated tuple. This stored information is now used at this stage to evaluate each tuple. Consider a tuple T generated by a list of patterns P ={ P_k }, the probability that T is a valid tuple can be measured by:

$$Prob\ (T) = 1 - \prod_{k=0}^{|P|}(1 - Prob\ (P_k)) \quad \text{.....(4.5)}$$

The above equation shows that the probability of tuple T is the cumulative product of the individual probabilities of all generating patterns of T. Remember from section 4.6, Prob(T) can also be replaced by Conf(T), and can be normalized by the degree of match between the context of T and each pattern in P, so equation 4.5 can be written as:

$$Conf\ (T) = 1 - \prod_{k=0}^{|P|}(1 - (Conf\ (P_k)\ .Degree\_of\_Match(C_k, P_k))\ ) \quad \text{.....(4.6)}$$

Where C_k is the 5-tuple from where tuple T is extracted by each pattern in P. If Conf (T) is $\geq$ $Threshold_{tuple}$, it is considered a reliable tuple and is added to the trusted tuple table. If enough numbers of trusted seed tuples are generated, the system iterates to the pattern generation step (section 4.4), otherwise it stops to iterate back to pattern generation. An algorithmic representation of tuple evaluation is illustrated in figure 4.11.

```
Algorithm    evaluateTuples (Generalized_patterns, Candidate_Tuples)

For each candidate in Candidate_Tuples
  Conf : = 0, Prob := 1
  For each pattern P in Generalized_patterns
      If    P = candidate. generatingpattern
            Prob = Prob X (1- (P.Conf X Degree_of_Match (P , T. context )))
      End If
  End For Loop
  Conf := 1 – Prob
  Candidate.conf := Conf
End For Loop

Return Candidate_Tuples with Confidence Scores
```

Figure 4.11: Algorithmic representation of tuple evaluation phase

4.9 Implementation

4.9.1 Platform Specification

The system is implemented using Java 6 SE (Standard Edition), which is currently the latest major release of Oracle ('Java SE 6', n.d.). Although the system is tested on the Windows 7 operating system, the platform independence nature of java guarantees that it can be effortlessly carried to any other platform. The classical features of java such as polymorphism, inheritance, data abstraction *etc.* and its latest advancement in java 6 such as XML-support etc., fascinated the author opt it as an implementation language. Java 6 also makes provision of complex data structures. The use of these complex data types, such as Hashset, Hashmap, Arraylist and Iterators, reduces the average execution time and lines of code at many implementation points. An open source Java development environment named BlueJ facilitated frequent unit testing to check the functionality of all the methods of each class individually.

4.9.2 System Components

The overall system implementation consists of the following major components (source code in appendix 1), which contribute as core classes in integrated system functionality, with the help of other implemented supporting classes and some external libraries.

4.9.2.1 User Interface

The user interface class provides a graphical interface for initially getting a few sample examples of a relation from user. The interface consists of graphical components provided by Java swing and AWT (abstract windows toolkit) libraries. Each user entry consists of an entity pair followed by a valid URL. The URL field is by default set to the URL used for standard Google search, so if the URL is not provided; a list of valid URLs which are likely to contain user-provided entity pairs is made. This is carried out by Google AJAX API and the results are retrieved in JSON ('Java Script Object Notation', n.d.).

4.9.2.2 Sentence Extractor

Once we have a list of URLs, we exploit the list to collect valid text for corpus creation. Sentence Extractor is used to collect plain unstructured text from each of the given URLs and break it down into sentences. The navigation to each URL and scraping of plain text from HTML code is done with the help of an internal method named 'extract contents'. It makes use of boilerpipe (Kohlschütter *et al.*, 2000), an external library to ensure that plain text doesn't contain any html tags and is also free of any boilerplate text (*e.g.* advertisements, navigational contents etc.). The scrapped sentences were observed to contain trailing spaces between words such as newlines, tabs and duplicated spaces, which were removed by a regular expression:

```
Sentence = Sentence.replaceAll(\\s+ , " ");
```

The system may have to face some unexpected circumstances such as the URLs provided by the user may be unavailable at the moment when the system tries to acquire data, or it may possibly no longer be valid. The system connection to the Internet may also have gone down. To ensure that these kinds of unexpected situation do not cause overall system failure, the sensitive code is enclosed within a java-provided try-catch structure where these exceptions have also been also handled properly.

4.9.2.3 CPattern

CPattern is a 5-tuple representation (section 4.4.1) of a sentence. CPattern is so named because Java already has a class with static methods in its internal regex library. These regex classes and their methods support the formulating of the complex regular expression. The regular expression used to convert a sentence into a 5-tuple representation is:

```
Pattern="(.*)(<([A-Z]*)>("+entity1+")</\\3?>?)(.*)(<([A-Z]*)>("+entity2+")</\\7?>?)(.*)";
```

Here, entitity1 and entity2 are variables which hold names of valid entities. Before converting a sentence into its 5-tuple representation, it is first validated by sentence validator class. Sentence validator implements all the rules specified in section 4.3.2.Its main method named 'validate' signals its calling method if a given sentence contains a valid relation and is breakable into 5-tuple. It also uses another class named entity tagger, which exploits the Stanford named entity tagging (section 4.3.3) library to tag entities.

To chunk the left and right context number of terms up to maximum limit, the context chunker class has been implemented, which emulates the functionality of a chunking window (section 4.4.2). All three contexts are then sent to a method of term vector, which converts simple text strings into their vector representation by assigning weights to each context term.

4.9.2.4 Cluster

Cluster class is an abstract cluster representation of cluster objects (section 4.4.4). It has a list of pattern and a centroid of the list. Each time a new pattern is added to the list, the centroid is updated by the 'reset centroid' method. For centroid recalculation, the intersection of terms is carried out by 'find intersection' and 'remove unintersected terms' methods. To find similarities between new patterns to be added, centroid, 'cos similarity' and 'find match' methods of pattern matcher are implemented.

4.9.2.5 Tuple Generator

Tuple generator is used to extract new tuples from the same corpus from where patterns were learned. It uses a web crawler class to navigate to web pages for extracting text segments containing exactly one

entity pair; they are validated by the ‘validate tuple generation’ method of sentence validator class. Entities are tagged with entity tagger class. The resulting text segments are converted to 5-tuple using CPattern class and compared with the pattern list using pattern matcher class methods. Each extracted tuple is represented as an instance of a candidate tuple class.

4.9.2.6 Evaluator

Evaluator implements two important methods, ‘evaluate patterns’ and ‘evaluate tuples’ to evaluate generated patterns and tuples respectively. The ‘score pattern’ method is called for setting the score field of each pattern where the number of positive and negative matches are counted with the help of ‘is negative’ and ‘is positive’ internal methods. This score field is subsequently used by the ‘score tuple’ method to set the conf field of each candidate tuple.

4.10 Summary

This chapter elaborated the important aspects of the work carried out throughout the project. It explains the technique used as a basis to architect the system. Each step of the technique along with novelties introduced to customize the system to adapt to the changing requirements (*i.e.* using the technique on an online corpus rather than a hand-crafted offline corpus) as well as an algorithmic module used to implement each step has been presented. It illustrated the interaction of each module with rest of the system and its contribution to overall system functionality.

It first describes rules and strategies for a valid corpus design, which is used to generate patterns. These learned patterns are then used for extracting relations from the corpus. These newly generated patterns and tuples are then evaluated and a set of best tuples and patterns is filtered out, these are then used to bootstrap the system. Finally, we discussed the platform used to implement the system and some important implemented system components. In the next chapter we will evaluate the overall system implementation and test the novelties by analyzing results obtained by different experimental settings.

Chapter 5

Experimental Results

5.1 Introduction

In chapter 4 we presented a detailed description of the design and implementation of the work carried out throughout the system, along with an algorithmic representation of the core modules of our system.

This chapter first presents the evaluation method used for analyzing the system's performance. Following this we will present a detailed description of the experiments carried out to evaluate our system. The experiments will be carried out on two different datasets. It also presents the results of each experiment and a comparative analysis of our system to Snowball's original implementation.

5.2 Evaluation method

As our system is based on the Snowball technique, which aims at extracting every mention of a relation type, our system is also expected to perform on the same grounds. Although the system is intended to capture all valid tuples as much as possible from the given document collection, it is not anticipated to capture all occurrences of a tuple that occurs more than once in the document collection.

In order to evaluate the system's ability in efficiently extracting all valid tuples, we use two important evaluation metrics, *i.e.* precision and recall (discussed in section 2.9), which obviously require the formulation of an ideal set, *i.e.* all the tuples in the document collection. This can be possible as long as an ideal set is to be compiled from a small document collection, but it is impossible to compile an ideal table from hundreds or thousands of documents. So here we use another more flexible method called random sampling. Instead of accumulating all valid tuples into one ideal table manually, a limited number of tuples are selected randomly from each document by hand. This random collection is then used as an ideal table. The system is then evaluated by calculating the number of tuples identified by the system out of this random collection of manually extracted tuples.

5.3 Experiments

As the system is intended to extract relations from a corpus consisting of online documents, and the technique implemented by the system is originally designed for offline data and is tested on hand-crafted corpus, there is no existing evidential attempt which guarantees that it will work on online documents. The experiments are an attempt to uncover the fact that whether the implemented technique can be adapted to work on highly unstructured text of web documents. Here, it is also desirable to find out the effects of changes made to the original technique at certain decision points, *e.g.* the use of cosine similarity instead of simple dot product of vectors to get a normalized similarity score instead of individually normalizing the weight of each term, which intuitively requires changes in term weight calculation.

5.3.1 Internal parameter settings

Before commencing with any experimentation of the system, a number of internal parameter values must be set by assigning a constant value to each of them. In order to get experimental results comparable to

the performance of the original Snowball implementation made by Agichtein and Gravano (2000), all parameters are set to the same value as with the original Snowball experiments.

Parameter	Description	Value
C_L	Maximum number of terms in left context	2
C_M	Maximum number of characters in middle context	500
C_R	Maximum number of terms in right context	2
$Threshold_{Sim}$	Minimum degree of match between any two 5-tuple representations.	0.6
$Threshold_{Conf}$	Minimum confidence score for seed tuple	0.8
W_L	Weight of left context	0.2
W_M	Weight of middle context	0.6
W_R	Weight of right context	0.2
$Iteration_{max}$	Maximum number of iterations by the system	3

Table 5.1: Internal parameter settings of our system for experiments

As shown in table 5.1, a pattern P can be added to a cluster C if the degree of match between P and centroid of C $\geq Threshold_{Sim}$. The same parameter value is also used as a minimum degree of match required between any of the generated patterns and context of a tuple to be extracted. Similarly, if a newly extracted tuple has a confidence score $\geq Threshold_{Conf}$, it is also promoted as a seed tuple for extracting patterns and tuples for subsequent iterations. $W_{L,}$ W_M and W_R are scaling factors for left, middle and right contexts respectively, whereas $C_{L,}$ C_M and C_R are lengths used by chunking windows for left, middle and right contexts. As noted in section 4.5.2, the middle context is the most important in terms of extracting relations; therefore its weight is set to a higher value than weights of the left and right. For the same reason, unlike the left and right contexts; the length of the middle context is measured in the number of characters and is not chunked but rather, if a sentence has length of middle context > $C_{M,}$ then it is discarded by the system.

5.3.2 Training Collection

As the only training source for the system is a collection of just five seed tuples initially inputted by user, careful choice of input tuples is very important. The system will get high coverage in generating tuples if the seed tuples entered by the system can generate as many of the possible selective patterns as they can. Also, all entity pairs in input pairs should be bounded by an instance of same relation type.

To experiment with the system over web documents, each input entry consists of an entity pair and an optional URL. If a URL is not given, a valid URL selection is compensated by the first 64 results from a standard Google search. The URL for each seed tuple should necessarily contain at least one valid occurrence of the seed tuple and other instances of same relation type that the system is currently trained on.

5.3.3 Experiment 1

5.3.3.1 Method

This experiment tests the correctness and performance of the system on offline plain text documents. It also conducts a comparative analysis of the system's performance with the original implementation of Snowball made by Agichtein and Gravano (2000). They tested their implementation on seed tuples having <ORGANIZATION, LOCATION> relation type or more specifically on <ORGANIZATION, HEADQUARTER> and the test collection used is 'North American News Text Corpus', which is composed of news articles from different news sources from 1995-1997 provided by LDC ('Linguistic Data Consortium', n.d.), which is an open consortium which compiles different spoken or written corpuses, databases and dictionaries in different languages which are then distributed by the consortium for research purposes.

Seed Tuple	
ORGANIZATION	**LOCATION**
Microsoft	Redmond
IBM	Armonk
Intel	Santa Clara
Coca Cola Co.	Atlanta
Boeing	Seattle

Table 5.2: seed tuples used for Experiment 1

The seed tuples chosen for this experiment (Table 5.2) are bounded by same relation type to which seed tuples provided by Agichtein and Gravano(2000) belongs *i.e.* <ORGANIZATION, HEADQUARTER> but limited computational resources constrained the system from using the same document collection of 142000 individual documents. So here we use 2000 documents from the 'New York News Text Corpus' of 1996, which is a small portion of the 'North American News Text Corpus'.

5.3.3.2 Results

The evaluation method described in section 5.2 is used to verify the validity of all newly generated tuples. In order to calculate the precision of the system, one hundred random samples out of all 1210 tuples extracted by the system are taken. All the samples are then verified with Google finance ('Google finance', n.d.) for their correctness, whereas recall is calculated by creating an ideal table consisting of one hundred random samples from the test document collection.

Tuple Conf	Correct Relations	Errors		Precision	
		Incorrect Relations	Incorrect Entities	Actual	Ideal
≥ 0.0	55	3	42	0.55	0.97
≥ 0.8	89	2	10	0.89	0.98

Table 5.3: precision of 100 randomly selected tuples in Experiment 1

As depicted in table 5.3, when the confidence score is set to the minimum level, *i.e.* 0.0, the system extracted 46 incorrect relations out of 100. It has been observed that only 3 incorrect relations are due to system errors, whereas all others are because of errors caused by improper entity recognition. This causes a low precision of the system. To determine the system's precision by just considering our system's errors, ideal precision is also calculated by omitting all errors caused by the entity tagger. Ideal precision of the system is 0.97, which is much higher than the precision level calculated by considering entity tagging errors along with system errors. The system's precision is further increased by re-sampling to take only those tuples having precision not less than the preset $\text{Threshold}_{\text{Conf}}$. The results show that most of the errors caused by the system and entity tagger have vanished; therefore actual as well as ideal precision levels are increased, *i.e.* by 0.34 and 0.01 respectively.

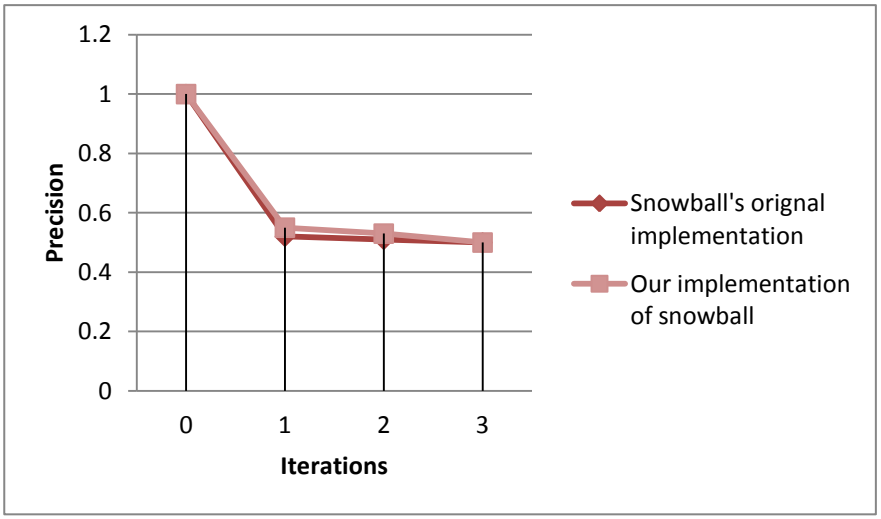

Figure 5.1: comparative analysis of the system's precision at each iteration

The comparative analysis of precision of our implementation of Snowball and its original implementation done by Agichtein and Gravano (2000) has shown (figure 5.1) that precision of both the systems

decreases with each iteration, but our system's implementation of Snowball has a slightly higher precision at each iteration than the Snowball's original implementation. It shows that variations and decisions made at different phases of the technique slightly increased the system's precision at all three iterations. The comparative analysis of both the systems (figure 5.2) is also done at different confidence levels of tuples. The results depicted that our system has approximately the same precision as Snowball's original implementation.

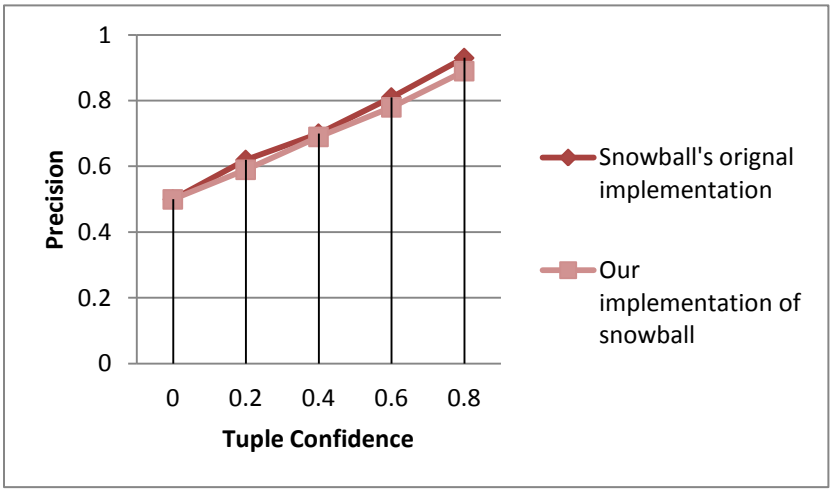

Figure 5.2: Comparative analysis of the system's precision at different tuple confidence scores

In order to calculate recall, one hundred tuples are taken from the test document collection, each of which is checked to see if it is in the list of tuples extracted by the system. The recall recorded for the first three iterations of our system and with different confidence threshold settings, is compared to that of the original Snowball's implementation, as shown in figures 5.3 and 5.4 respectively. The results of each iteration show that our system has recall which is slightly higher or equal to Snowball's original implementation. It is also depicted in figure 5.4 that our system has a higher recall than the original implementation at low confidence threshold, but increasing the threshold causes our system's recall to become lower than the original implementation.

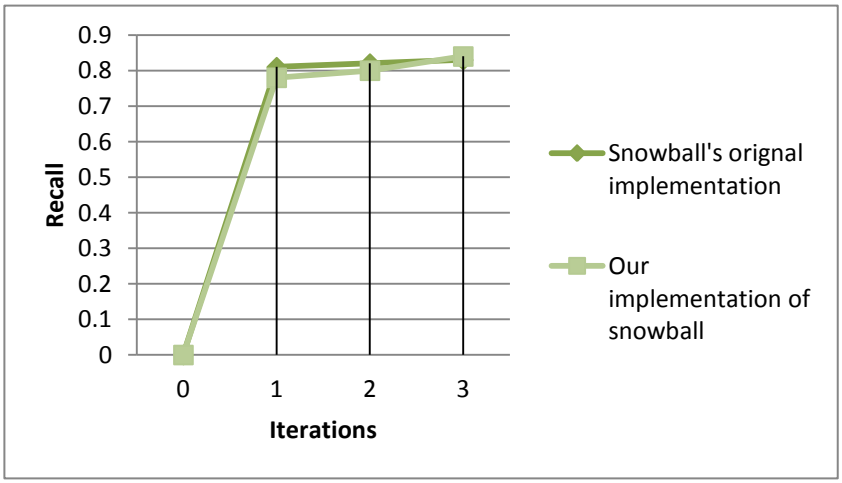

Figure 5.3: comparative analysis of the system's recall at each iteration

The analysis of the results has revealed the fact that the precision of the system decreases with an increase in the number of iterations, whereas its recall increases. Similarly, it has also been noticed from the results that an increase in the confidence threshold of tuples causes a decrease in recall, whereas the precision level is increased.

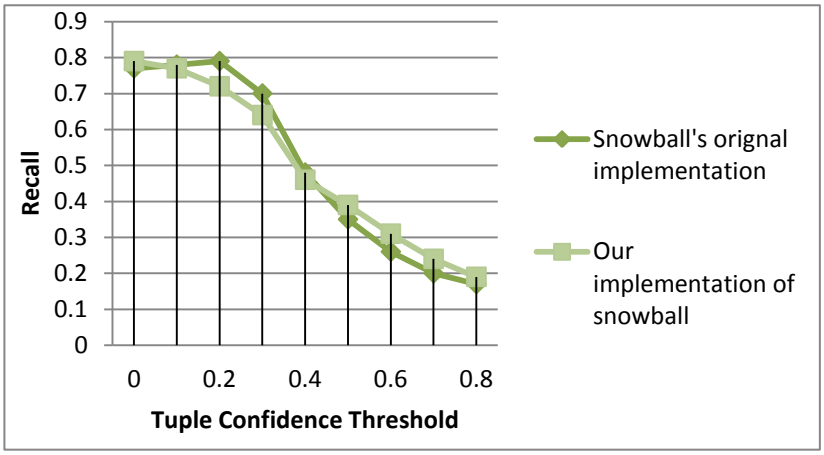

Figure 5.4: comparative analysis of recall at different confidence levels

5.4.4 Experiment 2

5.4.4.1 Method

As the system is expected to extract relations from web document collection, it is also tested on a collection of online documents to find out whether Snowball can be adapted to work on an online corpus or not. If so, it is also of interest to us to determine its performance while coping with unstructured text of web documents.

Seed Tuple		URL
PERSON	**LOCATION**	
Benito Mussolini	Italy	http://www.spartacus.schoolnet.co.uk/PRchamberlain.htm
Tony Tan	Singapore	http://rulers.org/2011-09.html
Julia Gillard	Australia	http://www.squidoo.com/women-presidents-women-prime-ministers
Hasina	Bangladesh	http://www.pmo.gov.bd
Golda Meir	Israel	http://womenshistory.about.com/od/rulers20th/a/women_heads.htm

Table 5.4: User input for Experiment 2

The input used as training collection to carry out this experiment is shown in table 5.4, which shows each seed tuple is an instance of relation type <PERSON, LOCATION> ,or more specifically can be written as < LEADER, COUNTRY>.

5.4.4.2 Results

To verify the correctness of each tuple extracted by the system, we consulted a website maintained by the central intelligence agency ('Central intelligence agency', n.d.). It is an online database of all world leaders. Including the URLs inputted by the user, the system navigated 132 web documents. Other than these five URLs inputted by the user, all other URLs obtained by searching seed tuples using standard Google search, are added to URL list. The system extracted 1020 tuples in all three iterations. Many of the tuples were extracted more than once.

Tuple Conf	Correct Relations	Errors		Precision	
		Incorrect Relations	Incorrect Entities	Actual	Ideal
≥ 0.0	54	4	22	0.67	0.95
≥ 0.8	74	1	5	0.92	0.98

Table 5.5: precision of 80 randomly selected tuples in Experiment 2

In order to compute precision of the system (Table 5.5) while working over a structured text of web documents, we sampled 80 unique tuples from the list of extracted tuples. Surprisingly, the number of errors caused by the entity tagger while working on online web documents is less than those for offline documents which may be because of acquiring valid list of URLs using standard Google search. Therefore actual as well as ideal precision results are higher than those of experiment 1.

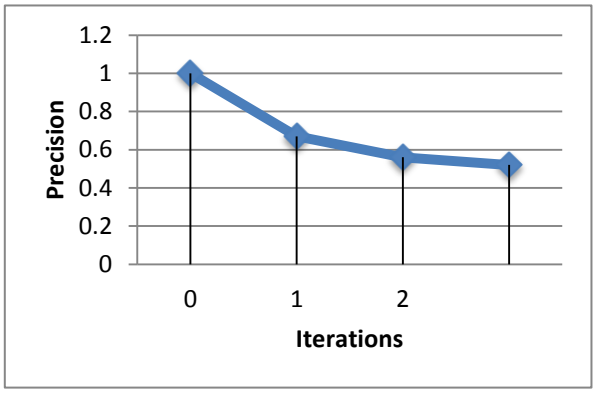

Figure 5.5: Analysis of the system's precision over web documents at different iterations

Table 5.5 presents the overall system's precision and samples are taken from the list of tuples extracted during all three iterations. The precision calculated at all three iterations separately is shown in figure 5.5. It shows that the same trend of decrease in precision is followed by the system irrespective of whether it is working on offline or online documents. The precision of the system at all three iterations here is higher than that recorded in experiment 1, which shows the system's eligibility to work on any unstructured text.

In order to find out the effect of changing tuple confidence threshold over precision, values for the precision are recorded at the different confidence thresholds of tuples, as shown in figure 5.6, which shows that the precision values for higher confidence levels is higher than those recorded in experiment 1, whereas at low confidence levels, the precision values are lower than recorded earlier in experiment 1.This change may be caused by the use of different test collections in both the experiments.

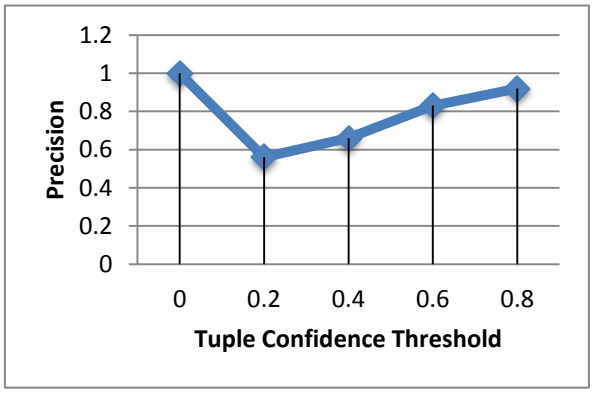

Figure 5.6: analysis of the system's precision over web documents at varying confidence levels

The recall is also recorded at each iteration is shown in figure 5.7. It shows that the system's recall at all three iterations is slightly lower than the recall values recorded earlier for the iterations in experiment 1. The may be because of the use of a wide variety of contexts for representation of the same relation type in web documents. This also confirms that selectivity increases precision and decreases coverage which results in decrease in recall.

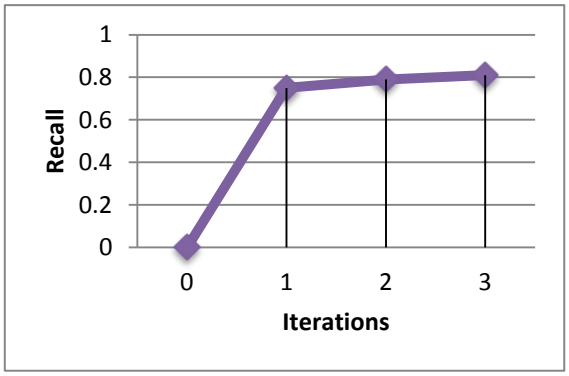

Figure 5.7: The system's recall over web documents at different iterations

Finally, to get complete comparable results from both the experiments; recall is calculated against different confidence threshold settings (figure 5.8). The values obtained at different confidence levels in both the experiments have shown slightly increased recall on test collection of this experiment as compared to the collection used in experiment.

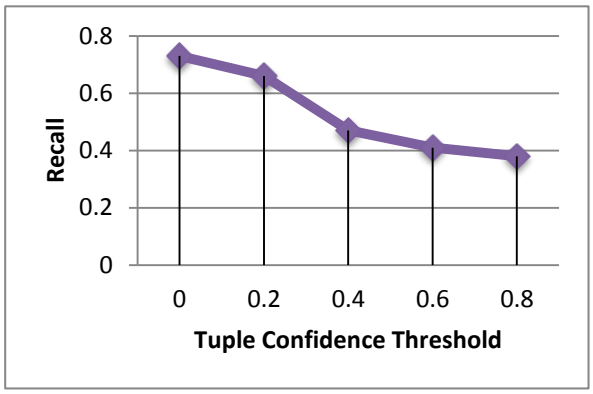

Figure 5.8: The system's recall over web documents at varying confidence levels

5.4 Discussion

The sampled results obtained from both the experiments have shown that most of the errors while extracting relations are caused by improper entity tagging. However, our system showed higher precision than Snowball's original implementation. The reason for the increase in precision may be the use of a different entity tagger, term weighting scheme, context matching and pattern scoring strategies, but as high precision is obtained by high selectivity of patterns, it lowers their coverage. Hence, many of the relations are left unrecognized by the system, which decreases recall of the system. Therefore our system has lower recall than that of its original implementation by Agichtein and Gravano (2000).

Some incorrect tuples were generated by the system in spite of proper entity tagging. This may be caused by generation of some overly general patterns which match the context of entities not bounded by the required relation type. If these tuples occurs more than once in similar contexts, the same pattern can redundantly generate incorrect relations.

The experimental results also showed that the system performed well on highly unstructured web contents. The precision level at most calculating points seemed to increase because of the improved performance of the entity tagger and the use of different test document collection, but for the same reason that has been discussed earlier in this section, the recall of the system for online document collection has also been observed to increase as the precision decreases.

5.5 Summary

This chapter presents the overall testing and evaluation carried out on the system. It started with an explicit description of the evaluation method used to analyze the performance of the system. We also put forward optimal internal parameter settings. The system has experimented with two different test collections, *i.e.* an offline as well as online corpus to analyze the performance of both.

The collected results have uncovered the fact that the technique can be adapted to work on an online corpus. The comparative analysis of the original Snowball implementation against our implementation has shown that variations made to the original technique at different stages worked well and increased overall system precision, but at the same time reduced its recall. Finally, we presented a brief discussion on overall results obtained by the experiments. The next chapter will present the conclusion drawn from the study, an analysis of our project up until now, and future recommendations to extend our work.

Chapter 6

Conclusion and Future Work

6.1 Introduction

Relation Extraction is the most important area of information extraction because of its capability to convert unstructured text into structured informational field values. However, adapting existing systems to new domains and relation types requires considerable human effort and expertise to train these systems to work on new domains. This project solves these problems by developing a system based on a weakly-supervised approach which requires minimal human supervision and effort. This chapter presents the concluding discussion of the project and directions for future work.

6.2 Conclusion

The project is basically aimed at providing automatic relations extraction solution to recognize semantic relations between any entity pairs from unstructured text in web documents. In chapter 3 we first reviewed a number of existing attempts proposing different solution to what was aimed to be done in the project. The study showed that most of the existing approaches require a significant amount of manually annotated text to train the system to perform relation extraction tasks on an intended domain. It was also noticed that some techniques cannot be trained to work on any other domain, whereas others show poor performance on some or all domains.

We took Snowball (Agichtein & Gravano, 2000) as the basic framework for our system as it is a weakly-supervised approach and requires few relation examples from the user. However, Snowball was actually designed to work on a corpus consisting of offline documents, it was not guaranteed to work on web documents, but the experimental results depicted that it can be adapted to work on web documents while preserving its original level of precision and recall.

Certain changes as well as additional processing steps were intuitively required to adapt the technique to work on web documents such as plain text files, web documents containing plain text embedded with visual formatting tags and also, plain scrapped natural language text can contain boilerplate, such as advertisements and navigational options, which obviously needs an additional processing step. To prevent the impact of such additional processing over the system's performance, the strategies requiring less computational resources were used at different steps of the overall process. These strategies include a different weighting scheme (section 4.5.2), a string similarity approach (section 4.5.3), and constraint-based pattern scoring strategy (section 4.7) for pattern evaluation.

To test the changes made to originally proposed Snowball technique, we implemented the overall system along with all design decisions made during the design phase of the system (discussed in detail in chapter 4). To carry out the system's implementation, we chose the java 6 platform because of its traditional overwhelming features and the advanced data structures introduced in its latest version, this made our system platform independent and also reduced implementation efforts and time. Rather than re-inventing the wheels, existing libraries including Stanford named entity taggers and boilerpipe were put into place to carry out the specific tasks of entity tagging and text scrapping respectively. During the implementation, each module was individually tested to ensure that it performed what was intended for it and that it interacted correctly with rest of the implementation.

The system was tested on two different test collections, *i.e.* online as well as hand-crafted corpuses. The performance of the system was measured by two different metrics, *i.e.* precision and recall, which were

calculated by random sampling (sections 2.9 and 5.2). Each randomly taken tuple was then verified for its correctness from reliable web sources. Such manual verification was the most labour-rigorous and time-consuming task.

The system showed higher precision (chapter 5) than Snowball's original implementation when tested on the New York Times News Corpus (1996), the part of the same corpus on which Snowball was originally tested. This may be because of the use of different entity tagger and design decisions. However, it showed a lower recall value than the original implementation because of high selectively and low coverage of newly generated patterns at each iteration. The system could not achieve 100% precision and recall because of incorrect tuple generation. It was observed that most of the incorrect relations are due to improper entity tagging.

The system was also experimented over an online corpus consisting of web document collection. The results showed approximately the same precision, and even higher than that of the offline corpus at some points (section 5.3.2), and almost the same recall level as recorded for offline test collection.

In conclusion, the system has proven successful in performing its basic functionality of extracting semantic relations from any collection of web documents by implementing the Snowball technique with a number of variations. It is observed to have high precision and recall values. Our solution to relation extraction problem is domain-independent, automatic, requires minimal human supervision and is suitable for large text collection.

6.3 Future Work

Although the project implemented the system with all the basic functionality required to extract semantic relations from web documents with a significant level of performance parameters, further contributions can be made to extend it to perform well in new directions. Some potential future extensions listed with solutions proposed by the author can enhance the capability and performance of our work. However, the solutions provided here are not tested by the author because of time constraint.

6.3.1 N-ary relations

As discussed in section 4.5.1, the patterns used to generate new tuples are represented by 5-tuples. Each pattern representation contains exactly one entity pair and the surrounding context. A pattern is represented by 5-tuple, which can only be used to represent binary relations. It implies that the generated relations will also be binary. But free text of natural language does not only contain binary relations but rather more than an entity pair can also be associated by a relation. Such n-ary relations cannot be extracted by our current implementation, but can be extended to capture the relations involving any number of entities.

A solution to implement this extension may be to eliminate the constraint of converting a candidate sentence into an exact 5-tuple representation but rather the system should be able to generate any N-tuple representation. **N** should be calculated as:

$$N = E * (E +1) \text{……..} (6.1)$$

Here, E represents total number of entities in a sentence or text segment, and E+1 is the number of contexts surrounding all entities.

6.3.2 Non-local relations

As discussed in section 4.4.2, during preprocessing; our currently implemented text chunker takes sentence as a unit for chunking while performing its task on the text contents of web documents. This is carried out on the basis of the assumption that if any two entities are bounded by a relation, then they occur within the same sentence. It depicts the fact that all the relations to be extracted by the system are local, *i.e.* within same sentence, but unstructured text has evidenced to contain non-local relations. These non-local relations may be paragraph-level, document-level or even may be multi-document. This means that even if the entities occur far apart from each other within the text, they can still be candidates for relations.

In order to extend the system to work on non-local relations, a solution may be to store information about not just the local context but also about the context of entities in neighboring sentences. Although the proposed solution seems suitable for paragraph-level relations, it seems infeasible to capture multi-document relations and even document-level relations in long text documents.

6.3.3 Pronoun mention resolution

It has been observed that if natural language text is a continuous description about the same entity; then after the first sentence it is mentioned by a pronoun. These kinds of examples can be seen in the biography of a person where, after introducing a person, references are used to describe his/her relation with other entities, *e.g.*

"Bill gates is one of the richest personalities in the world. He is a current CEO of Microsoft…"

In the above example, a pronoun 'he' is used to refer to the person 'Bill gates', and shows his relation with an organization, 'Microsoft'. Such kinds of relations containing pronoun mentions of entities are omitted by our currently implemented system, but a module can be added to the pre-processing step that should resolve all the co-references before performing sentence validation and entity tagging tasks.

6.4 Summary

This chapter concluded the project with a discussion on important findings of the project. It also described some future directions in which to extend the project's capability to extract all kinds of non-local and n-ary relations. The coverage of the currently implemented system can be improved by adding a new feature of pronoun mention resolution. We also proposed solutions for all extensions which could not be tested because of limited time constraint.

References

ABNEY, S. 1991. Parsing by chunks, Dordrecht., Kluwer Academic Publishers.

AGENCY, C. I. https://www.cia.gov [Online]. [Accessed September 2011].

AGICHTEIN, E. & GRAVANO, L. 2000. Snowball: extracting relations from large plain-text collections. Proceedings of the fifth ACM conference on Digital libraries. San Antonio, Texas, United States: ACM.

ATZENI, P., MENDELZON, A., MECCA, G. & BRIN, S. 1999. Extracting Patterns and Relations from the World Wide Web. *The World Wide Web and Databases.* Springer Berlin / Heidelberg.

BAEZA-YATES, R. & B. RIBEIRO-NETO, B. 1999. Modern Information Retrieval, New York, ACM Press.

BAEZA-YATES, W. B. F. R. 1992. Information Retrieval: Data Structures and Algorithms., Prentice-Hall.

BERNERS-LEE T., H. J., LASSILA O 2001. The Semantic Web. Scientific American, 284, 10.

BIKEL D., SCHWARTZ R. & WEISCHEDEL R. 1999. An algorithm that learns what's in a name. *Machine Learning Journal Special Issue on Natural Language Learning.*

BLEI, D. M., NG, A. Y. & JORDAN, M. I. 2003. Latent dirichlet allocation. J. Mach. Learn. Res., 3, 993-1022.

BOBROVNIKO, D. 2000. Experiments in Semantic Bootstrapping with a Cluster-Based Extension to DIPRE. *Stanford University* (2000).

CHIANG, J.-H., YU, H.-C. & HSU, H.-J. 2004. GIS: a biomedical text mining system for gene information discovery. Bioinformatics Applications Note, 20, 2.

CONFERENCE, M. U. http://www.itl.nist.gov/iaui/894.02/related_projects/muc/ [Online]. [Accessed September 2011].

CONFERENCE, T. R. http://trec.nist.gov/ [Online]. [Accessed September 2011].

CONSORTIUM, L. D. http://www.ldc.upenn.edu [Online]. [Accessed September 2011].

COULET, A., SHAH, N., HUNTER, L., BARREL, C. & ALTMAN, R. B. 2010. Extraction of Genotype-Phenotype-Drug Relationship from Text:. *Pacific Symposium on Biocomputing,* 15**,** 477-480.

DEFAYS, D. 1977. An efficient algorithm for complete link method. The Computer Journal, 20, 2.

DODDINGTON, G., MITCHELL, A., PRZYBOCKI, M., RAMSHAW, L., STRASSEL, S. & WEISCHEDEL, R. 2004. The Automatic Content Extraction (ACE) Program Tasks , Data , and Evaluation. 4**,** 4.

ELLEN M. VOORHEES & DAWN M.TICE. 2000. The TREC-8 question answering track evaluation. *Proceedings of the Eighth Text REtrieval Conference (TREC-8)*, pp. 83–105, 2000. NIST Special Publication 500-246.
EXTRACTION, A. C. http://www.itl.nist.gov/iad/mig/tests/ace/ [Online]. [Accessed September 2011].

FINKEL, J. R., GRENAGER, T. & MANNING, C. 2005. Incorporating non-local information into information extraction systems by Gibbs sampling. Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics. Ann Arbor, Michigan: Association for Computational Linguistics.

GIULIANO, CLAUDIO., LAVELLI, ABLERTO., ROMANO LORENZA. Exploiting Shallow Linguistic Information for Relation Extraction from Biomedical Literature. In Proceedings of the *11th Conference of the European Chapter of the Association for Computational Linguistics (EACL 2006)*, Trento, Italy, 3-7 April 2006.

GOOGLE FINANCE. http://finance.google.com/finance [Online]. [Accessed September 2011].

GORDON, R. K. L. A. M. D. 1999. Literature-based discovery by lexical statistics. Journal of the American Society for Information Extraction, 13.

GRISHMAN, R. & SUNDHEIM, B. 1996. Message understanding conference - 6: A brief history. *Proceedings of the International Conference on Computational Linguistics.*

GUODONG, Z., JIAN, S., JIE, Z. & MIN, Z. 2005. Exploring various knowledge in relation extraction. Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics. Ann Arbor, Michigan: Association for Computational Linguistics.

HAKENBERG, J., LEAMAN, R., NGUYEN HA, V., JONNALAGADDA, S., SULLIVAN, R., MILLER, C., TARI, L., BARAL, C. & GONZALEZ, G. 2010. Efficient Extraction of Protein-Protein Interactions from Full-Text Articles. *Computational Biology and Bioinformatics, IEEE/ACM Transactions on,* 7**,** 481-494.

HASEGAWA, T., SEKINE, S. & GRISHMAN, R. 2004. Discovering relations among named entities from large corpora. *Proceedings of the 42nd Annual Meeting on Association for Computational Linguistics.* Stroudsburg, PA, USA, , Article 415: Association for Computational Linguistics.

HIRSCHMAN, L. & GAIZAUSKAS, R. 2001. Natural language question answering: the view from here. Natural Language Engineering, 7, 26.

HRISTOVSKI D., P. B., MITCHELL J.A., HUMPHREY S.M., 2005. Using literature-based discovery to identify disease candi. Int J Med Inform, 74, 10.

HU, Y., LU, R., CHEN, Y. & PEI, B. 2007. Text Retrieval Oriented Auto-construction of Conceptual Relationship. Proceedings of the 7th international conference on Computational Science, Part II. Beijing, China: Springer-Verlag.

JAVA SE 6. http://www.oracle.com [Online]. [Accessed September 2011].

JONNALAGADDA, S. & GONZALEZ, G. 2010a. BioSimplify: an open source sentence simplification engine to improve recall in automatic biomedical information extraction. *AMIA Annu Symp Proc,* 2010**,** 351-5.

JONNALAGADDA, S. & GONZALEZ, G. 2010b. Sentence Simplification Aids Protein-Protein Interaction Extraction. *The 3rd International Symposiumon Languages in Biology and Medicine,* abs/1001**,** 6.

JONNALAGADDA, S., LEAMAN, R., COHEN, T. & GONZALEZ, G. 2010a. *A Distributional Semantics Approach to Simultaneous Recognition of Multiple Classes of Named Entities,* Romania, Springer Berlin / Heidelberg.

JONNALAGADDA, S., TARI, L., HAKENBERG, J. & BARAL, C. G., GRACIELA 2009a. Towards effective sentence simplification for automatic processing of biomedical text. *Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics, Companion Volume: Short Papers.* Boulder, Colorado: Association for Computational Linguistics.

JONNALAGADDA, S., TOPHAM, P. & GONZALEZ, G. 2010b. ONER: Tool for Organization Named Entity Recognition from Affiliation Strings in. *The 3rd International Symposiumon Languages in Biology and Medicine.* Jeju Island, South Korea: eprint ARXIV.

JONNALAGADDA, S., TOPHAM, P. & GONZALEZ, G. Towards automatic extraction of social networks of organizations in PubMed abstracts. Bioinformatics and Biomedicine Workshop, 2009. BIBMW 2009. IEEE International Conference on, 1-4 Nov. 2009 2009b. 279-286.

JAVA SCRIPT OBJECT NOTATION. http://www.json.org [Online]. [Accessed September 2011].

KAMBHATLA, N. 2004. Combining lexical, syntactic, and semantic features with maximum entropy models for extracting relations. Proceedings of the ACL 2004 on Interactive poster and demonstration sessions. Barcelona, Spain: Association for Computational Linguistics.

KOHLSCHÜTTER, C., FANKHAUSER, P. & NEJDL, W. 2003. Boiler plate detection using Shallow Parsing. The Third ACM International Conference on Web Search and Data Mining. New York City, NY USA.

LEI SHI & RADA MIHALCEA. 2005. Putting pieces together: Combining framenet, verbnet and wordnet for robust semantic parsing. *The 6th International Conference on Computational Linguistics and Intelligent Text Processing (CICLing-05),* Mexico City, Mexico,May 2 - 7.

LIN, D. & PANTEL, P. 2001. Dirt-discovery of inference rules from text. *Proceedings of ACM Conference on Knowledge Discovery and Data Mining(KDD-01).* San Francisco, CA.

M. BIKEL, D., MILLER, S., SCHWARTZ, R. M. & WEISCHEDEL, R. M. 1997. Nymble: a High-Performance Learning Name-finder. *Proceedings of the fifth Applied Natural Language Processing Conference, Association of Computational Linguistics.*

MAKHOUL, J., KUBALA, F., SCHWARTZ, R. & WEISCHEDEL, R. 1999. *Performance Measures For Information Extraction,* San Francisco, DARPA publications.

MARCUS, M. P., MARCINKIEWICZ, M. A. & SANTORINI, B. 1993. Building a large annotated corpus of English: the penn treebank. Comput. Linguist., 19, 313-330.

MARSH, E. 1998. TIPSTER information extraction evaluation: the MUC-7 workshop. *Proceedings of a workshop on held at Baltimore, Maryland: October 13-15, 1998.* Baltimore, Maryland: Association for Computational Linguistics.

MCDONALD, R. 2004. Extracting Relations from Unstructured Text. Technical Report(MS-CIS-05-06).
MICHAEL S., MORGENSTERN B., STOYE, J. 2003. Divide-and-Conquer multiple alignment with segment-based constraints. Bioinformatics. 19(2): 189-195.

MILLER, G. A. 1995. WordNet: a lexical database for English. Commun. ACM, 38, 39-41.

MILLER, S., CRYSTAL, M., FOX, H., RAMSHAW, L., SCHWARTZ, R., STONE, R., WEISCHEDEL, R. & GROUP, T. A. 1998. Algorithms that learn to extract information–BBN: Description of the SIFT system as used for MUC. *In Proceedings of the Seventh Message Understanding Conference (MUC-7)*.

MILLER, S., FOX, H., RAMSHAW, L. & WEISCHEDEL, R. 1998. A novel use of statistical parsing to extract information from text.

MOLINA, A. & PLA, F. 2002. Shallow parsing using specialized hmms. J. Mach. Learn. Res., 2, 595-613.

NEEDLEMAN, S. & WUNSCH, C. 1970. A general method applicable to the search for similarities in the amino acid sequences of two proteins. Journal of Molecular Biology, 48, 10.

PYYSALO, S., GINTER, F., HEIMONEN, J., BJORNE, J., BOBERG, J., JARVINEN, J. & SALAKOSKI, T. 2007. Bioinfer: a corpus for information extraction in the biomedical domain. BMC Bioinformatics, 8, 24.

ROSENFELD, B. & FELDMAN, R. 2007. Clustering for unsupervised relation identification. Proceedings of the sixteenth ACM conference on Conference on information and knowledge management. Lisbon, Portugal: ACM.

ROTH, D. & YIH, W.-T. 2002. Probabilistic reasoning for entity \& relation recognition. Proceedings of the 19th international conference on Computational linguistics - Volume 1. Taipei, Taiwan: Association for Computational Linguistics.

ROZENFELD, B. & FELDMAN, R. 2006. High-Performance Unsupervised Relation Extraction from Large Corpora. Data Mining, 2006. ICDM '06. Sixth International Conference on, 1032-1037.

SALTON, G. & MCGILL, M. 1983. Introduction to modern information retrieval, McGraw-Hill.

SAPATINAS, T. 2004. The Elements of Statistical Learning. Journal of the Royal Statistical Society: Series A (Statistics in Society), 167, 192-192.

SARIC, J., JUHL JENSEN, L., OUZOUNOVA, R., ROJAS, I. & BORK, P. 2006. Extraction of regulatory gene/protein networks from Medline. Bioinformatics, 22, 6.

SENSEVAL. http://www.senseval.org [Online]. [Accessed September 2011].

SHARMA, A., SWAMINATHAN, R. & HUI, Y. A Verb-Centric Approach for Relationship Extraction in Biomedical Text. Semantic Computing (ICSC), 2010 IEEE Fourth International Conference on, 22-24 Sept. 2010 2010. 377-385.

STOYANOV, V., CARDIE, C., GILBERT, N., RILOFF, E., BUTTLER, D. & HYSOM, D. 2010. Coreference resolution with reconcile. Proceedings of the ACL 2010 Conference Short Papers. Uppsala, Sweden: Association for Computational Linguistics.

SUTTON, C. & MCCALLUM, A. 2002. An introduction to conditional random fields for relational learning. Computer and information science, 93.

TATA, S. & PATEL, J. M. 2007. Estimating the selectivity of tf-idf based cosine similarity predicates. SIGMOD Rec., 36, 75-80.

TATAR, S. & CICEKLI, I. 2009. Two learning approaches for protein name extraction. *Journal of Biomedical Informatics,* 42**,** 1046-1055.

TICE, E. M. V. A. D. M. 1999. The TREC-8 Question Answering Track Evaluation, In Text Retrieval Conference TREC-8.

TONY , M. & WILSON, A. 1996. Corpus Linguistics: An Introduction, Edinburgh, Edinburgh University Press.

TSURUOKA, Y. & TSUJII, J. I. 2003. Boosting precision and recall of dictionary-based protein name recognition. *Proceedings of the ACL 2003 workshop on Natural language processing in biomedicine.* Sapporo, Japan: Association for Computational Linguistics.

YAKUSHIJI, A., TATEISI, Y., MIYAO, Y. & TSUJII, J. 2001. Event Extraction from Biomedical papers using a full parser. 6, 12.

YAROWSKY, D. 1995. Unsupervised word sense disambiguation rivaling supervised methods. *Proceedings of the 33rd annual meeting on Association for Computational Linguistics*. Cambridge, Massachusetts: Association for Computational Linguistics.

Appendix 1

Part of Source Code

This appendix shows source code of some of the important classes our system. Here, just important methods are included in the class representation. Other commonly used and less important such as set and get methods are omitted. We have also omitted most of the internal as well as external comments, fields of the classes and import lines which are used to import external libraries.

SentencesExtractor.java

```
01 /**
02  *  scarps text from web pages at given URL while throwing away html tags and boilerplate
03  * The text is then chunked into sentences
04  */
05 public class SentencesExtractor {
06
07    public void process(String url) {
08
09       extractContents(url);
10    }
11    //extracts text from web page
12
13    private void extractContents(String link) {
14
15
16       String filename = "out2.txt";
17       try {
18          URL url = new URL(link);
19
20          String output = KeepEverythingExtractor.INSTANCE.getText(url);
21          //create a stream to file and write output
22          FileWriter filestr = new FileWriter(filename);
23          BufferedWriter out = new BufferedWriter(filestr);
24          out.write(output);
25          out.close();
26          System.out.println(link);
27          contents = processContents(filename);
28
29
30
```

```
31        } catch (Exception e) {
32            System.out.println("An error occured during opening the link" + e);
33
34        }
35    }
36
37    //convert text into valid sentence list
38    private List<String> processContents(String filename) throws IOException {
39        File file = new File(filename);
40        SentenceValidator validator = new SentenceValidator();
41        String input = Files.readFromFile(file, "ISO-8859-1");
42        List<String> sentence_list = new ArrayList<String>();
43
44        Chunking chunks = chunker.chunk(input.toCharArray(), 0, input.length());
45        Set<Chunk> sentences = chunks.chunkSet();
46        if (sentences.size() < 1) {
47            System.out.println("No string to chunk");
48            return sentence_list;
49        }
50        String sentenc = chunks.charSequence().toString();
51       for (Iterator<Chunk> iter = sentences.iterator(); iter.hasNext();) {
52          Chunk sentence = iter.next();
53          int sentence_str = sentence.start();
54          int sentence_end = sentence.end();
55          String Sent = sentenc.substring(sentence_str, sentence_end);
56          //System.out.println(Sent);
57          Sent = Sent.replaceAll("\\s+", " ");
58
59          if (param1.equals("") && param1.equals("")) {
60             contents.add(Sent);
61          } else if (Pattern.compile(Pattern.quote(param1), Pattern.CASE_INSENSITIVE).matcher(Sent).find()) {
62             if (Pattern.compile(Pattern.quote(param2), Pattern.CASE_INSENSITIVE).matcher(Sent).find()) {
63                contents.add(Sent);
64             }
65          }
66       }
67
68       return contents;
69    }
70
71    }
72 }
```

CPattern.java

```
01 public class CPattern {
02
03    public boolean createTuple(String sentence, String param1, String param2) {
04
05        try {
06           //tagging named entites in the sentence
07           EntityTagger tagger = new EntityTagger();
08           sentence = tagger.tag(sentence);
```

```
09          SentenceValidator validator = new SentenceValidator();
10
11
12
13          if (validator.Validate(sentence, param1, param2)) {
14
15              System.out.println(sentence);
16
17              //assigning paramter entities to class object variables.
18              entity1 = param1;
19              entity2 = param2;
20
21
22
23
24              String pattern = "";
25
26              //setting the order of entities
27              sentence = sentence.toLowerCase();
28              param1 = param1.toLowerCase();
29              param2 = param2.toLowerCase();
30              if (sentence.indexOf(param1) < sentence.indexOf(param2)) {
31                  order = 0;
32                  pattern = "(.*)(<([A-Z]*)>(" + param1 + ")</\\3?>?)(.*)(<([A-Z]*)>(" + param2 + ")</\\7?>?)(.*)";
33              } else {
34                  order = 1;
35                  pattern = "(.*)(<([A-Z]*)>(" + param2 + ")</\\3?>?)(.*)(<([A-Z]*)>(" + param1 + ")</\\7?>?)(.*)";
36                  entity2 = param1;
37                  entity1 = param2;
38
39              }
40
41              Pattern p = Pattern.compile(pattern, Pattern.CASE_INSENSITIVE);
42              Matcher m = p.matcher(sentence);
43              if (m.find()) {
44                  //converting sentence to tuple
45
46                  left_context = m.group(1);
47                  entity1_type = m.group(3);
48
49                  middle_context = m.group(5);
50                  entity2_type = m.group(7);
51
52                  right_context = m.group(9);
53                int Index1 = left_context.lastIndexOf(">");
54                int Index2 = right_context.lastIndexOf(">");
55                if (Index1 != -1) {
56                    left_context = left_context.substring(Index1 + 1);
57                }
58
59                if (Index2 != -1) {
60                    right_context = right_context.substring(Index2 + 1);
61                }
62                //chunking the contexts to proper size
63                ContextChunker Cwindow = new ContextChunker();
64                left_context = Cwindow.chunkTOSize(left_context, "left");
```

```
65                  right_context = Cwindow.chunkTOSize(right_context, "right");
66              }
67
68              if (!entity1_type.equals("") && !entity2_type.equals("")) {
69
70                  //creating vectors from chunked context Strings
71                  TermVector vector = new TermVector();
72
73                  //considering empty context as a token so that weights can be assigned
74                  if (left_context.equals("")) {
75                      left_context = "*";
76                  }
77                  if (right_context.equals("")) {
78                      right_context = "*";
79                  }
80                  if (middle_context.equals("")) {
81                      middle_context = "*";
82                  }
83                  left_terms = vector.countFreq(left_context, "left");
84                  middle_terms = vector.countFreq(middle_context, "middle");
85                  right_terms = vector.countFreq(right_context, "right");
86                  return true;
87              }
88          }
89      } catch (Exception e) {
90      }
91      return false;
92  }
93
94  public boolean equals(CPattern p) {
95      if (p == this) {
96          return true;
97      } else {
98          return false;
99      }
100   }
101 }
```

Cluster.java

```
1
2 public class Cluster
3 {
4
5   //This method list of term lists into hashset to remove duplicated terms
6   private List<HashSet<String>> getTermSet(List<List<Term>> Vectors)
7   {
8     List<HashSet<String>> VectSets= new ArrayList<HashSet<String>>();
9     Iterator mover=Vectors.iterator();
10      while(mover.hasNext())
11      {
12       List<Term> lst=(List<Term>)mover.next();
13      HashSet<String> terms = new HashSet<String>();
14      Iterator it=lst.iterator();
```

```
15        while(it.hasNext())
16        {
17           Term trm=(Term)it.next();
18           terms.add(trm.getWord());
19        }
20
21       VectSets.add( terms);
22     }
23
24
25     return VectSets;
26
27 }
28
29 // This method will find out intersection of terms in all the contexts
30 private List<String> findIntersection(List<HashSet<String>> VectSets)
31 {
32
33   HashSet<String> str= new HashSet();
34   HashSet<String> str2= new HashSet();
35   Iterator it = VectSets.iterator();
36   if(it.hasNext())
37    str=(HashSet)it.next();
38   while(it.hasNext())
39   {
40      str2=(HashSet) it.next();
41      str.retainAll( str2);
42    }
43
44
45   return new ArrayList<String>(str);
46
47 }
48
49 //This method will remove all the terms not common in pattern's context
50 private List<Term> removeUniintersectedTerms(List<String> strings, List<Term> terms)
51 {
52
53 List<Term> res=new ArrayList<Term>();
54   Iterator it= strings.iterator();
55   while(it.hasNext())
56   {
57     String tofind= (String)it.next();
58     Iterator iterate= terms.iterator();
59   while(iterate.hasNext())
60   {
61
62      Term term=(Term) iterate.next();
63      String tomatch= term.getWord();
64      if(tofind.equalsIgnoreCase(tomatch))
65       {
66          res.add(term);
67       }
68   }
69
70    }
```

```
71    return res;
72 }
73
74 //This method will calculate centroid of a given context
75 private List<Term> calcCentroid( List<List<Term>> TV,String context)
76 {
77    TermVector v1=new TermVector();
78      List<HashSet<String>> VectSets= new ArrayList<HashSet<String>>();
79      VectSets=getTermSet(TV);
80      List<String> str=findIntersection( VectSets);
81      List<List<Term>> all=new ArrayList<List<Term>>();
82      for(int i=0;i<TV.size();i++)
83      {
84          List<Term> Vect=TV.get(i);
85          Vect=removeUniintersectedTerms(str,Vect);
86           all.add(Vect);
87        }
88
89        Iterator keys=str.iterator();
90        List<Term> resultant=new ArrayList<Term>();
91        while(keys.hasNext())
92        {
93
94            Term t=getMaxValue((String)keys.next(),all,context);
95            if(t!=null)
96            resultant.add(t);
97        }
98
99
100       double wt=0.0;
101       int fr=1;
102       if(resultant.size()<1)
103       {
104           if(context.equalsIgnoreCase("left") || context.equalsIgnoreCase("right"))
105          wt=0.2;
106           else if(context.equalsIgnoreCase("middle"))
107               wt=0.6;
108           resultant.add(new Term("*",context,wt,fr));
109       }
110      return resultant;
111   }
112
113   // This method search for a key term in given list of term and returns it
114   private Term searchTerm(String key, List<Term> terms)
115   {
116
117      Iterator it=terms.iterator();
118     while(it.hasNext())
119     {
120      Term term=(Term)it.next();
121      if(key.equalsIgnoreCase(term.getWord()))
122      return term;
123      }
124     return null;
125   }
126
```

```
127   //this method finds out maximum term value among given set of values for a term
128    private Term getMaxValue(String key,List<List<Term>> term_lists,String context)
129    {
130       int maxfreq=0;
131       double maxweight=0.0;
132
133       Iterator it=term_lists.iterator();
134       while(it.hasNext())
135       {
136          List<Term> lst=(List<Term>)it.next();
137          Term current=searchTerm(key,lst);
138          if(current.getWeight()>maxweight)
139          maxweight=current.getWeight();
140          if(current.getFreq()>maxfreq)
141          maxfreq=current.getFreq();
142       }
143
144
145    Term res= new Term(key,context,maxweight,maxfreq);
146    return res;
147 }
148
149   public void resetCentroid()
150   {
151    List<List<Term>> left=  new ArrayList<List<Term>>();
152    List<List<Term>> middle=  new ArrayList<List<Term>>();
153    List<List<Term>> right=  new ArrayList<List<Term>>();
154    Iterator it=All_Patterns.iterator();
155    while(it.hasNext())
156    {
157       CPattern Ptrn= (CPattern)it.next();
158       left.add(Ptrn.getLeftTerms());
159       middle.add(Ptrn.getMiddleTerms());
160       right.add(Ptrn.getRightTerms());
161    }
162    calcCentroid(left,"left");
163    calcCentroid(middle,"middle");
164    calcCentroid(right,"right");
165
166   }
167 }
```

Evaluater.java

```
1 package relationshipextr;
2
3 /**
4  *  evalutes newly generated patterns, assigns score to each one.
5  * It will then use the scored patterns to measure conf score of all new tuples
6  */
7 public class Evaluater {
8
```

```
 9    public void evaluatePatterns(List<CPattern> patterns, List<SeedTuple> seeds, List<CandidateTuple>
Candidates) {
10        Iterator it = patterns.iterator();
11        while (it.hasNext()) {
12            CPattern p = (CPattern) it.next();
13            ScorePattern(p, seeds, Candidates);
14        }
15    }
16
17    public List<SeedTuple> evaluateTuples(List<CandidateTuple> candidates, List<CPattern> patterns) {
18        Iterator it = candidates.iterator();
19        List<SeedTuple> seeds = new ArrayList<SeedTuple>();
20        while (it.hasNext()) {
21            CandidateTuple tuple = (CandidateTuple) it.next();
22            scoreTuple(tuple, patterns);
23            if (tuple.getConf() >= MinTupleConf) {
24                seeds.add(new SeedTuple(tuple.getEntity1(), tuple.getEntity1(), "http://www.google.com"));
25            }
26        }
27        return seeds;
28    }
29
30    //This method is used for scoring individual tuple
31    private void scoreTuple(CandidateTuple tuple, List<CPattern> patterns) {
32        double conf = 0.0;
33        double Prob = 1.0;
34        Iterator it = patterns.iterator();
35        while (it.hasNext()) {
36            CPattern ptrn = (CPattern) it.next();
37            if (ptrn.equals(tuple.getGenPattern())) {
38                Prob *= 1 - ptrn.getScore() * tuple.getScore();
39
40            }
41        }
42
43        conf = 1 - Prob;
44        tuple.setConf(conf);
45    }
46
47    //This method is used for scoring individual pattern
48    private void ScorePattern(CPattern p, List<SeedTuple> seeds, List<CandidateTuple> Candidates) {
49        int positive = 0, negative = 0;
50        Iterator it = Candidates.iterator();
51        while (it.hasNext()) {
52            CandidateTuple tuple = (CandidateTuple) it.next();
53            if (tuple.getGenPattern().equals(p)) {
54                if (isPositive(tuple, seeds)) {
55                    positive++;
56                } else if (isNegative(tuple, seeds)) {
57                    negative++;
58                }
59            }
60        }
61
62
63        if ((positive + negative) == 0) {
```

```
64            p.setScore(0.0);
65        } else {
66            p.setScore((positive / (positive + negative)));
67        }
68
69    }
70
71    //This method will check if tuple is positive
72    private boolean isPositive(CandidateTuple tuple, List<SeedTuple> seeds) {
73        boolean res = false;
74
75        Iterator it = seeds.iterator();
76        while (it.hasNext()) {
77            SeedTuple sd = (SeedTuple) it.next();
78            if (sd.getParam1().equals(tuple.getEntity1()) && sd.getParam2().equals(tuple.getEntity2())) {
79                res = true;
80                break;
81            }
82        }
83
84        return res;
85    }
86    //This method will check if tuple is negative
87
88    private boolean isNegative(CandidateTuple tuple, List<SeedTuple> seeds) {
89        boolean res = false;
90        Iterator it = seeds.iterator();
91        while (it.hasNext()) {
92            SeedTuple sd = (SeedTuple) it.next();
93            if (sd.getParam1().equals(tuple.getEntity1()) && !(sd.getParam2().equals(tuple.getEntity2()))) {
94                res = true;
95                break;
96            } else if (!(sd.getParam1().equals(tuple.getEntity1())) && (sd.getParam2().equals(tuple.getEntity2()))) {
97                res = true;
98                break;
99            }
100       }
101
102       return res;
103   }
104 }
105
```