`

A
Dissertation
On

"Automatic Evaluating Text Coherence using Discourse Relations"

Submitted in Partial fulfillment of the requirement
For the award of Degree of

MASTER OF TECHNOLOGY
Computer Technology and Application
Delhi Technological University, Delhi

SUBMITTED BY

RAJIV RATN SHAH
University Roll No: 13/CTA/2010

Under the Guidance of:

Mr. MANOJ KUMAR

Associate Professor
Delhi Technological University

DEPARTMENT OF COMPUTER ENGINEERING
DELHI TECHNOLOGICAL UNIVERSITY
2010-2012

`

CERTIFICATE

This is to certify that the work contained in this dissertation entitled “**Automatic Evaluating Text Coherence using Discourse Relations**” submitted in the partial fulfillment, for the award for the degree of M. Tech. in Computer Technology and Applications at **DELHI TECHNOLOGICAL UNIVERSITY** by **RAJIV RATN SHAH, Roll No. 13/CTA/2010** is carried out by him under my supervision. This matter embodied in this project work has not been submitted earlier for the award of any degree or diploma in any university/institution to the best of our knowledge and belief.

(Mr. MANOJ KUMAR)
Project Guide
Associate Professor
Department of Computer Engineering
Delhi Technological University

`

ACKNOWLEDGEMENT

First of all, let me thank the almighty god, my parents and my dear friends who are the most graceful and merciful for their blessing that contributed to the successful completion of this project.

I feel privileged to offer sincere thanks and deep sense of gratitude to **Mr. MANOJ KUMAR,** project guide for expressing his confidence in me by letting me work on a project of this magnitude and using the latest technologies and providing their support, help & encouragement in implementing this project.

I would like to take this opportunity to express the profound sense of gratitude and respect to all those who helped us throughout the duration of this project. **DELHI TECHNOLOGICAL UNIVERSITY**, in particular has been the source of inspiration, I acknowledge the effort of those who have contributed significantly to this project.

RAJIV RATN SHAH
(Roll No.: 13/CTA/2010)

`

ABSTRACT

This thesis considers the problem of automatic evaluation of text coherence. The task of text coherence in linguistics is what makes a text semantically meaningful. Automatic Evaluating Text Coherence, the task of determining which text is more coherent between given pair of text and its sentence ordered permutation. This work has been at the core of the field of Natural Language Processing for the past few years. Natural Language Processing is often described as a discipline that relates to human ability with what computers can do. Study of Natural Language Processing helps us to achieve human level performance through computers. The task of determining which given text is more coherent is very important and challenging problem in Natural Language Processing. One famous application of text coherency can be applied to impose an order on sentences for multi-document summarization. With the tremendous growth of data, users are expecting more relevant and sophisticated information which may be determined by Text Summarization. Natural Language Processing is often described as a discipline to develop applications related to human linguistics. It involves different techniques and algorithms to determine which text is more coherent between given pair of text and its sentence ordered permutation and it can be applicable to NLP application of text summarization. The idea of modeling automatic evaluating text coherence may apply to differentiating a text from its permutation (i.e., the sentence ordering of the text is shuffled) and identifying a more well-written essay from a pair.

In this thesis, we propose a novel approach for determining the automatic evaluating text coherence which is the combination of the new and other related text coherency techniques. Also we prove its effectiveness over various previous techniques such as Entity Grid Relations and Discourse Relations over Entity Grid Model. Entity Grid

`

Relations is the first popular technique for automatic evaluating text coherence but accuracy of this model is quite less than human level performance of task of evaluating text coherence. To improve the accuracy of Entity Grid Model, Discourse Relations imposed over Entity Grid Model have become popular (Lin et. al., 2011). (Lin et. al., 2011) have employed a Discourse Relation Matrix to determine discourse relation transitions of different length. However, the accuracy of Discourse Relations Model is still less than accuracy of human evaluator in evaluating text coherence task.

Our proposed model is able to decide which text is more coherent. We have presented a novel approach to combine few independent semantic features to determine the coherency of text. Our study of linguistics tells us that co-reference plays a vital role in determining the coherency of a text. In particular there exist model of the noun phrase syntax used for distance (named hobb distance) between noun phrase and its co-reference with statistical distribution of the discourse structure and relations. We have considered the text coherency problem as ranking learning problem because for a given pair a text is more coherent than the other. Our system ranks high coherent text with higher score. Our experiments have shown that combining these features together lead to improvement in accuracy of automatic evaluating text coherence. We apply Penn Discourse Treebank (PDTB) discourse relations values (Lin et. al., 2011) and Noun Phrase co-reference over the Entity Grid Model by Barzilay and Lapata (2005; 2008), a popular model of local coherence. Our experiments and results demonstrate that our model achieves higher accuracy than baseline model. The accuracy of our system is closest to the accuracy of human evaluators than other existing model for automatically evaluating text coherence.

`

TABLE OF CONTENTS

`

`

List of Figures

`

List of Tables

`

Chapter 1: Introduction

As discourse is the coherent, structured group of sentences. Models of discourse coherence describe the relationships between neighboring sentences. It helps to determine that which particular ordering of sentences is more coherent. One famous application of text coherency can be applied to impose an order on sentences for multi-document summarization. Coherency of a text is based on its discourse relations, semantic features, co-reference of its noun and pronoun phrases. Barzilay and Lapata (2005; 2008) proposed coherence assessment as a learning task and show that their entity-based representation is well suited for ranking-based generation. Local coherences capture text relatedness at the level of neighboring sentence transitions. As in greedy approach, Local coherence lead to global coherence and has received considerable attention in linguistics. In our approach we have used an entity grid model Barzilay and Lapata (2005; 2008) to capture discourse entity transitions at the sentence-to-sentence level, and used discourse role matrix (Lin et. al., 2011) to consider the discourse role transitions of different length. (Lin et. al., 2011) proposed that overall distribution of discourse role transitions for a coherent text is distinguishable from that for an incoherent text.

We have taken the work of Barzilay and Lapata (2005; 2008) as a baseline and combined this with the extracted sub-sequences with various lengths from the discourse role matrix as features and compute the sub-sequence probabilities as the feature values (Lin et. al., 2011).

We have then combined this model with the NP's Co-reference model to improve the accuracy of the automatic text coherence system. For generating the PDTB discourse relations, we have used the PDTB discourse parser by (Lin et. al., 2009). There are two levels of discourse relations in PDTB. Discourse Role Matrix have level 1 discourse relation as entry. For NP's co-reference resolution we have used "Reconcile" [16].

The explosive growth of many competitive examinations and survey has explored a wide area to interpret and digest this data. There is a need to find multi document Text

`

Summarization and rank the document based on their coherency. The answer to this problem was the evaluating the text coherence, which is the subject of our thesis. Recently, automatic evaluating of text coherence attracted a lot of research attention.

The problem of automatic evaluating of text coherence can be decomposed into two sub problems:

a) Train the system with set of training example pair (source text and its sentence ordered permutation), described in section 3.2.1. During training we have given source text a higher rank than its permutation because it is believed that source text is more coherent than its permutations. Further create the model file (described in section 4.4.3) for good text coherent system based on the support vectors (described in section 4.4.2) generated by training examples.

b) Using the model generated in last step and support vectors for the test text examples, our system predicts the coherency of the test examples in terms of coherency score in prediction file (described in section 4.4.4).

There is a wide agreement among the literature that the first sub-problem is more important of the two. This is because generated model for the text coherence is further used for prediction of text examples. If model would not be good then prediction would be not accurate and lowers the accuracy of the system. That is the reason that researchers paid the great attention to this problem in the recent years.

In this thesis, we expand the horizon of automatically evaluating text coherence by introducing a novel technique for automatically evaluating text coherence. We analyze the performance of our approach on two data sets consist of Associated Press articles about earthquakes from the North American News Corpus, and narratives from the National Transportation Safety Board. Experimental results demonstrate that our approach performs better than previous related works.

1.1 Motivation

A quality of sentences, paragraphs, and essays when all parts are clearly connected, is called coherence. Texts can be coherent mainly at two levels, one is called the 'local level' and the other is 'global level. Local level coherence occurs within small portions of texts, usually within a text no longer than a paragraph. A text is said to have global

`

coherence if the text hangs together as a whole. In our text coherence evaluation problem, we consider global coherence. Applications of text coherence are applied to find well written essay, multi document summarization and prevention of reader misconception (T Donaldson et al., 1996).

One of the biggest challenges that the linguists face today is to determine which text is more coherent. In the past few years many techniques have been made and the motivation for our work comes from the study of these techniques in Natural Language Processing. Each of these approaches has contributed to different enhancements in automatically evaluating text coherence. So we have decided to do our thesis with the aim of giving a better approach for improving the accuracy of the automatically evaluating text coherence.

1.2 Research Objective

This thesis reports on our approach to automatically evaluating text coherence. With respect to this, it explores Natural Language Processing techniques that could be applied to determine which given text is more coherent. The problem statement is: **"To propose a better approach for automatically evaluating text coherence for a given pair (source and its sentence ordered permutation) of text and comparing its accuracy with earlier approaches like Entity Grid Approach and Discourse Relations applied over Entity Grid Model using various real datasets".**

1.3 Related Work

Barzilay and Lapata (2005; 2008) proposed an entity-based model to represent local textual coherence. Their model were motivated by Centering Theory (Grosz et al., 1995), which states that subsequent sentences in a locally coherent text are likely to continue to focus on the same entities as in previous sentences. Barzilay and Lapata operationalised Centering Theory by creating an entity grid model to capture discourse entity transitions at the sentence to sentence level, and demonstrated that how local textual coherence lead to anticipate coherent texts from incoherent ones.

`

Micha Elsner and Eugene Charniak (2008) have proposed two models to improve the text coherency task. Their first model uses the features based on Uryupina (2003) to distinguishing discourse-new from discourse-old noun phrases. Discourse-new NPs are those whose referents have not been previously mentioned in the discourse. So discovering discourse-new NPs gives more information about the coherent text and lead to improve the accuracy of the system.

Their second model based on pronoun co-reference. In coherent text Pronouns must be placed close to appropriate referents with the correct number and gender. Finally they combined these two models with the entity grid described by Lapata and Barzilay (2005) for significant improvement in coherency task.

(Lin et. al., 2011) proposed an Automatically Evaluating Text Coherence using Discourse Relations. A coherent text exhibits measurable preferences for specific intra and inter discourse relation ordering. In their model, they capture the coherence of a text based on the statistical distribution of the discourse structure and relations. They specifically focus on the discourse relation transitions between adjacent sentences and fill them in a Discourse Role Matrix. They have proposed that the overall distribution of discourse role transitions for a coherent text is distinguishable from that for an incoherent text. Their model is able to evaluate the unseen text's coherency based on the distributional differences of such subsequences in coherent and incoherent text in training text.

1.4 Scope of the work

In this thesis, a novel approach for automatically evaluating text coherence is proposed. This approach gives better accuracy as compare to other previous approaches like Entity Grid and Discourse Relation over Entity Grid.

Our proposed approach applies PDTB discourse relations and noun phrase co-reference over the entity grid, a popular model of local coherence. Proposed system improves the accuracy than baseline. Using a Noun Phrase Co-reference technique over Entity Grid and Discourse Relation Model, the accuracy of automatic evaluating text coherence improved. The key benefit of the proposed approach is that its accuracy is close to accuracy of the human evaluators.

`

1.5 Organization of the thesis

In this chapter, we have highlighted the problems faced by users in the evaluating coherency of a text and the uses of Entity Grid, PDTB Discourse Relations and NPs co-reference to automatic evaluation of text coherence serves as the motivation for the work reported in this thesis. Furthermore we have also outlined the specific objective of our research and related research work that has occurred in the past.

The next chapter is literature reviews of related work in the field of automatic evaluation of text coherence. It gives brief introduction about Text Coherence task and how it is related to the field of Natural Language Processing. It also briefly introduces the entity grid and discourse relation. Finally, we present studies that employ the approach to find the coherency of text which help to position our work in its context in the following chapters.

Chapter 3 describes our approach to improve the accuracy of text coherent system. We described the weakness of traditional approaches and further described our approach to overcome these weakness.

Chapter 4 discuss about our Implementation, Experiments and Results. We have discussed the experimental setup, data sets used for training and testing and the performance of human evaluator on this task. We have also briefly discussed about the Support Vector Machine (SVM) RANK, a learning-ranking tool used in our system. Finally we present the comparison of our proposed techniques with the earlier techniques. Chapter 5 discuss about Conclusion and Future Scope of work to improve the accuracy of our system. The next section is the references of our work.

We have concluded this thesis by appendix having some part of our implementation, tools and librarries used in the development of our system to automatically evaluating text coherence.

`

Chapter 2: Literature Review

2.1 Text Coherence Basic Concepts

2.1.1 What is Text Coherence?

Evaluating Text Coherence is an application of Natural Language Processing. Coherence in linguistics is what makes a text semantically meaningful and especially dealt with in text linguistics. Coherence is achieved through syntactical features such as the use of deictic, anaphoric and cataphoric elements or a logical tense structure, as well as presuppositions and implications connected to general world knowledge. The purely linguistic elements that make a text coherent are subsumed under the term cohesion [13]. A quality of sentences, paragraphs, and essays when all parts are clearly connected, is called coherence. In linguistics, cohesion is closely related with coherence. Grammatical and lexical relationship within a text/sentence is called cohesion. It can be can be defined as the links that hold a text together and give it meaning. Coherence is related to the narrower concept of cohesion. Applications of cohesion are segmentation, word sense disambiguation, and extractive summarization, topical and stylistic analysis.

2.1.2 What is Natural Language Processing?

Natural Language Processing (NLP) is the computerized approach to analyzing text that is based on both a set of theories and a set of technologies. It is a very active area of research and development since last few years. Natural Language Processing is a theoretically motivated range of computational techniques for analyzing and representing naturally occurring texts at one or more levels of linguistic analysis for the purpose of achieving human-like language processing for a range of tasks or applications [14].

`

.2.1.3 Natural Language Processing Goal, Research and Evolution

‘Human-like language processing’ reveals that NLP is considered a discipline within Artificial Intelligence (AI). And while the full lineage of NLP does depend on a number of other disciplines, since NLP strives for human-like performance, it is appropriate to consider it an AI discipline [14].

The goal of NLP as stated above is “to accomplish human-like language processing”. The choice of the word ‘processing’ is very deliberate, and should not be replaced with ‘understanding’. For although the field of NLP was originally referred to as Natural Language Understanding (NLU) in the early days of AI, it is well agreed today that while the goal of NLP is true NLU, that goal has not yet been accomplished. A full NLU System would be able to:

1. Paraphrase an input text
2. Translate the text into another language
3. Answer questions about the contents of the text
4. Draw inferences from the text

While NLP has made serious inroads into accomplishing goals 1 to 3, the fact that NLP systems cannot, of themselves, draw inferences from text, NLU still remains the goal of NLP [14].

As most modern disciplines, the lineage of NLP is indeed mixed, and still today has strong emphases by different groups whose backgrounds are more influenced by one or another of the disciplines. Key among the contributors to the discipline and practice of NLP are: Linguistics - focuses on formal, structural models of language and the discovery of language universals - in fact the field of NLP was originally referred to as Computational Linguistics; Computer Science - is concerned with developing internal representations of data and efficient processing of these structures, and; Cognitive Psychology - looks at language usage as a window into human cognitive processes, and has the goal of modeling the use of language in a psychologically plausible way [14].

`

2.2 Text Coherence Techniques

The most important Text Coherence techniques are as follows:

2.2.1 Entity Grid Relations

Lapata and Barzilay (2005) have introduced a method for coherence assessment that is based on grid representation. They have shown with their experiment that the distribution of entities in coherent texts exhibits certain regularities reflected in grid topology. Grids of coherent texts are likely to have some dense columns and many sparse columns which will consist mostly of gaps. They have observed that entities corresponding to dense columns are more often subjects or objects, which is not case with low-coherence texts. Entity Grid Model is motivated by Centering Theory. So patterns of local entity transitions can be used as a feature for learning/ranking algorithm.

We have used four symbols: S (subject), O (object), X (neither subject nor object) and – (entity is not present). A local entity transition is a regular expression $\{S,O,X,–\}^n$ that represents entity occurrences and their syntactic roles in n adjacent sentences. Local transitions can be easily obtained from a grid as continuous subsequences of each column. Each transition will have a certain probability in a given grid. Each text can thus be viewed as a distribution defined over transition types. We believe that considering all entity transitions may uncover new patterns relevant for coherence assessment.

We have further taken the salience of discourse entities into account.

For given example below in figure-2.2.1, we have shown the local entity transition of GROUND, GAUGE, INJURIES and IT entities.

"BC-Israel-Earthquake|Minor Tremor In Northern Israel
JERUSALEM (AP) A minor earthquake struck northern Israel
Thursday but caused no damage or injuries, an official said.
Ami Shapira of the Israel Seismological Institute said the
quake's epicenter was 20 kilometers (12 miles) east of Haifa

```
`
and was felt in some of the city's suburbs at 15:32 (13:32
GMT).
He said it measured 3 on the Richter scale, which is a gauge
of the energy released by an earthquake as measured by ground
motion recorded on a seismograph.
Generally a quake must measure 6 to do severe damage.
``It was quite small and it is quite surprising that people
felt it,'' Shapira said.
"
GROUND - - X - -
GAUGE - - O - -
INJURIES O - - - -
IT - - S - S
```

Figure-2.2.1 (Entiity Grid Example)

As there are five sentences, so for each entity there would be a regular expression of length five having entries of any of the four possible options. Suppose If we consider only above four entities then local entity transition {X,-} and {-,O} has only occurred only once and {-,S} and {O,-} have occurred twice. We will consider the probabilities of these two length subsequences as a feature value.

2.2.2 Discourse Relations

(Lin et al., 2011) have introduce the concept of a discourse role matrix which aims to capture an expanded set of transition patterns of its gold standard PDTB discourse relations. Rows of the discourse role matrix are the sentences and the columns are the stemmed form of open words (Noun, Verb, Adjective and Adverb). POS tagger is used to find these open words in given text. The entry of the discourse role matrix is the level 1 PDTB discourse role of the stemmed word in the sentences. To fill the entries of the Discourse Role matrix we have employed the output of a PDTB-Styled End-to-End Discourse Parser by (Lin et al., 2009).

The PDTB is a large scale corpus annotated with information related to discourse structure and discourse semantics. Though there are many aspects of discourse that

`

are crucial to a complete understanding of natural language, the PDTB focuses on encoding discourse relations. There are three levels of Discourse Relation in gold standard PDTB. For our evaluation of text coherence, we have considered level 1 PDTB discourse relations. In figure-2.2.2, we have shown the hierarchy of sense tags upto two levels. We have taken transitions of Temporal, Contingency, Comparison and Expansion PDTB discourse relation between adjacent sentences and fill them in a Discourse Role Matrix, which we will explain in next section. These entries help us to evaluate discourse relation transition probabilities of different length. Evaluated discourse relation transition probabilities used as a feature for support vectors for learning and predicting.

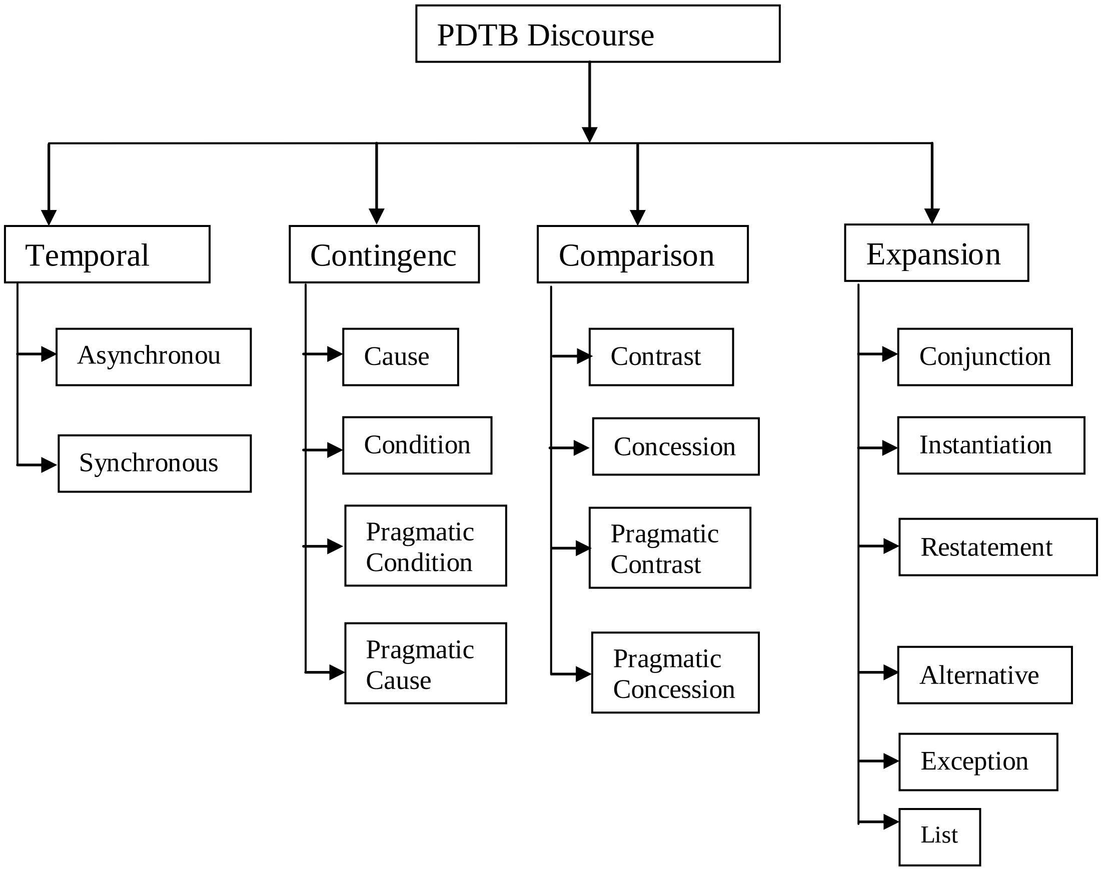

Figure-2.2.2 Hierarchy of Sense Tags

`

For e.g. for given text **“There, he fueled the *aircraft* with 58.15 gallons of fuel at 1503. The *aircraft* was taxied back to the pilot's hangar area Sometime thereafter”**

Discourse Parser Output

Here is the sample output of text given in above example by PDTB discourse parser (Lin et al., 2009). It’s shown the level-2 discourse relations in its output and it is further converted to level-1 discourse relation to find the discourse relation transition pattern. These level-1 discourse relations are filled in Discourse Role Matrix defined by (Lin et al., 2011).

S1- {**NonExp_0_Arg1** There, he fueled the *aircraft* with 58.15 gallons of fuel at 1503 **NonExp_0_Arg1**} S2- {**NonExp_1_Arg1** {**NonExp_0_Arg2_EntRel** The *aircraft* was taxied back to the pilot’s hangar area **NonExp_0_Arg2}** **NonExp_1_Arg1**}
S3- {**NonExp_2_Arg1** {**NonExp_1_Arg2_Asynchronous** {**Exp_0_Arg1** Sometime {**Exp_0_conn_Asynchronous** thereafter **Exp_0_conn**} , the pilot had used a jumper cable to connect the battery {**Exp_0_Arg2** of his car **Exp_0_Arg2**} to the *aircraft* 's electrical system at the power receptacle located on the left side of the nose . **Exp_0_Arg1**} **NonExp_1_Arg2**} **NonExp_2_Arg1**}

Discourse Role Matrix Entry

As discussed above that Discourse Role Matrix having level-1 PDTB discourse relation as its entry. Rows of the matrix represents the sentences of the given text corpus and Columns represents the stemmed version of the tokens present in given text. Table-2.2.2 represents the Discourse Role Matrix for example given above. Though we have only consider the discourse relation for *"aircraft"* token.

`

.

Sentence No.	**Discourse Relation of Token “*aircraft*”**
S1	EntRel.Arg1
S2	Temporal.Arg1, EntRel.Arg2
S3	Temporal.Arg2, Temporal.Arg1, xxx.Arg1

Table-2.2.2 (Discourse Relation of Token “*aircraft*” in examples given above)

Where, xxx is discourse relation, which should be obtained by **S4**, which is not shown here. The level-2 Asynchronous is mapped to level-1 temporal discourse relation.

2.3 Summary

In this chapter, we have surveyed the broad area of Text Coherence with a specific focus on its relation with Natural Language Processing. We have also introduced various prior techniques of automatically evaluating Text Coherence. The problem definition of automatically evaluating Text Coherence and its benefits and application are explored in detail. We have also surveyed many of the earlier techniques for automatically evaluating Text Coherence like Entity Grid and Discourse Relations and also explain them in detail using examples. We found that all the earlier algorithms have not considered the coreference in automatically evaluating Text Coherence, which is a very important aspect of Text Coherence. So we have proposed a new Noun Phrase coreference based approach with combination of Entity Grid Model and Discourse Relation Model in the next chapter.

`

Chapter 3: Proposed Technique for Automatically Evaluating Text Coherence

The objective of this chapter is to introduce our approach to automatically evaluating text coherence and to provide details of the work carried out in automatically evaluating text coherence from the data sets on earthquake and accidents [18]. In particular, we will suggest our approach to consider the normalized hamming distance between a Noun Phrase and all its co-reference as a feature with combination with Entity Relation Transition feature Discourse Relation Transition feature.

The chapter is organized as follows. Section 3.1 gives the brief description of weaknesses of traditional approaches. Section 3.2 provides the problem formulation, algorithm of the proposed technique and how it is combined with prior related work from (Lin et al., 2011) and Lapata and Barzilay (2005) for best results. Finally section 3.3 summarizes the chapter.

3.1 Traditional Approach Weakness

Co-reference in a sentence is a good measure to decide whether a given text is coherent or not. Most of the traditional approaches have not given significant attention to co-reference in evaluation of text coherence. Reason behind that co-reference has not given much attention is that there is not much work has been done in the field of automatically evaluating Text Coherence. To overcome the drawbacks of earlier techniques, we need to consider the co-reference as well with other semantic features in automatically evaluating text coherence. The processes of applying PDTB discourse relations and noun phrase co-reference over the entity grid make useful improvement in corresponding field.

`

It was found that all earlier algorithms are mainly focused on entity grid and discourse relations. They have not considered the co-reference in their work. We thought that considering co-reference together with Entity Grid and PDTB discourse relation model will improve the accuracy of the system. Our experimental results have shown that our approach improve the accuracy of system than that of baseline for our work. Accuracy of our system is close to the accuracy of the human evaluators, which is the aim of most of Natural Language Processing applications.

This way we reach a point where many researchers are trying to give a more efficient approach for finding the automatic evaluating text coherence. Barzilay and Lapata (2005; 2008) gives Entity Grid Model which is the first major work done in the field of automatic evaluating text coherence. Entity Grid Model was the first famous model in this area which was achieving better accuracy. Micha Elsner and Eugene Charniak (2008) have introduced discourse new in the evaluation of text coherence. Discourse-new NPs are those whose referents have not been previously mentioned in the discourse. They have applied their discourse new technique over Entity Grid Model. Using discourse new approach with Entity Grid leads to improvement in accuracy in text coherent task. But still improvement in accuracy is not high. (Lin et. al., 2011) have introduced the concept of discourse relation matrix which is having discourse relation entries of all open words in every sentences in source and permuted text. They have proposed a very good technique whose accuracy is much higher than the baseline. Their proposed technique is the combination of entity grid model and discourse relation. But still there is a scope to improve the accuracy as their accuracy is not equal to the accuracy of the human evaluators.

We propose a novel approach that improves the accuracy of our system and close to the accuracy of human evaluator and also support the incremental and interactive nature of data sets.

3.2 Proposed Technique

We have reviewed many techniques like Entity Grid Model, Discourse Relation Model and combination of these two techniques for evaluating text coherence in last

`

chapter. In this section, a new NPs co-reference based approach is proposed for automatically evaluating text coherence. It is an extended version of the earlier techniques and their combinations. Our experimental results have demonstrated that using Noun Phrase Co-reference based technique together with Entity Grid and Discourse Relation model improve the accuracy than baseline.

As discussed in chapter 2 that a pair of source text and its sentence ordered permutations is used for automatic evaluation of text coherence task. Section 3.2.1 briefly discussed about source text and its permutation. Section 3.2.2 shows the parsed text for source text. Parsed text are required to find the open words (noun, verb, adverb and adjective) from source text to fill the entries of columns of Discourse Role Matrix defined (shown in table-2.2.2) by (Lin et. al., 2011). Section 3.3.3 represents the entity grid representation of source text in figure-3.2.1(a). Figure-3.2.4 in section 3.2.4 is the output of the PDTB discourse parser for the source text in figure-3.2.1(a). In section 3.2.5, we have used Reconcile to generate the Noun Phrase Co-reference (given in figure-3.2.5) for the source text in section 3.2.1. Entity Relation transitions of N-gram length can be determined by entity grid output (figure-3.2.3) and N-gram Discourse Relation transition can be find out using parsed text (figure-3.2.2) and PDTB discourse parser output (figure-3.2.4). We have given the formulation to find these transition probabilities in section 3.2.6. We have used these transition probabilities as features to evaluate the support vector (figure-4.4.2) for source text and its permutations. In next chapter we have described how we have used this support vector for all the training pairs (source and its permutation) to create the model (figure-4.4.3) for best automatically evaluating text coherence system. We have used Support Vector Machine (SVM) discussed in section 4.4for this purpose. Support vectors for all the test pairs are also generated using features discussed above. Using support vector for test pairs and model, SVM give its prediction in terms of prediction score (given in figure-4.4.4) for each and every test pair.

3.2.1 Source Text and Its Permutations

Input to our system is the source text whose coherency has to be determined. In order to determine how coherent a text is? We have randomly swapped sentences of source text to determine the sentence ordered permutations for source text because Barzilay and Lapata (2005; 2008) have classified evaluating text coherence task as learning-ranking problem. As an example for automatically evaluating text coherence we have taken text corpus given in figure-3.2.1(a) as original/source text.

BC-Philippines-Quake, 0194| Strong Quake Rocks Philippines Island.

MANILA (AP) A strong earthquake rocked the Philippines island of Mindoro early Tuesday, killing at least two people and causing some damage, authorities said.

The 3:15 am quake had a preliminary magnitude of 6.7 and was centered near Baco on northern Mindoro Island, about 75 miles south of Manila, according to the Philippine Institute of Vulcanology and Seismology.

The U.S. Geological Survey in Menlo Park, Calif., put the quake's preliminary magnitude at 7.1.

Gov. Rodolfo Valencia of the island's Oriental Mindoro province said two people reportedly were killed and that several buildings and bridges were damaged by the quake.

Several homes near the shore reportedly were washed away by large waves, Valencia told Manila radio station DZBB.

Telephone service was cut, he said.

The quake swayed tall buildings in Manila.

Institute spokesman Aris Jimenez said the quake occurred on the Lubang fault, one of the area's most active.

A magnitude 6 quake can cause severe damage if centered under a populated area, while amgnitude 7 quake indicates a major quake capable of widespread, heavy damage.

Figure-3.2.1(a) (Source Text Corpus from [18])

`

Figure-3.2.1(a) is an example of source text whose coherency has to be determined. In order to determine the coherency of a text corpus we have compared its coherency with the coherency of its sentence ordered permutation. By sentence ordered permutation we mean that we have swapped the few sentences of source text randomly. Figure-3.2.1(b) is an example of sentence ordered permutation of source text given in figure-3.2.1(a).

Telephone service was cut, he said.

The U.S. Geological Survey in Menlo Park, Calif., put the quake's preliminary magnitude at 7.1.

Several homes near the shore reportedly were washed away by large waves, Valencia told Manila radio station DZBB.

Gov. Rodolfo Valencia of the island's Oriental Mindoro province said two people reportedly were killed and that several buildings and bridges were damaged by the quake.

The 3:15 am quake had a preliminary magnitude of 6.7 and was centered near Baco on northern Mindoro Island, about 75 miles south of Manila, according to the Philippine Institute of Vulcanology and Seismology.

A magnitude 6 quake can cause severe damage if centered under a populated area, while amgnitude 7 quake indicates a major quake capable of widespread, heavy damage.

Institute spokesman Aris Jimenez said the quake occurred on the Lubang fault, one of the area's most active.

MANILA (AP) A strong earthquake rocked the Philippines island of Mindoro early Tuesday, killing at least two people and causing some damage, authorities said.

BC-Philippines-Quake,0194| Strong Quake Rocks Philippines Island.

The quake swayed tall buildings in Manila.

Figure-3.2.1(b) (Permuted Text for Source Text Corpus)

`

We make pair of source text with all such permutations. Our system is then trained with all such training pairs and then support vectors for all such pairs are generated in format given in figure- 4.4.1.

3.2.2 POS Parsed Text

As we have discussed in section 1.3 that (Lin et. al., 2011) have introduced the concept of discourse role matrix which is having discourse relation entries of stemmed form of all open words in every sentences in source and permuted text. Open words are the Noun, Verb, Adverb and Adjectives used in the text corpus. Using Stanford POS tagger gives us the parsed text for given text corpus. These open words and their Term Frequencies are important aspect in Text Coherency. Using the output of the Stanford Parser we have determined the open words used in the text corpus to find the columns for discourse role matrix. Further discourse relation transition probabilities can be determined using discourse role matrix. Figure-3.2.2 is the parsed text for source text given in figure-3.2.1(a).

```
((NPB 5 (1 JJ "BC-PHILIPPINES-QUAKE,0194|") (2 JJ
"STRONG") (3 NN "QUAKE") (4 NNS "ROCKS") (5 NNP
"PHILIPPINES") (6 NNP "ISLAND"))
(SVP 2 (VP 2 (NPB 0 (1 NNP "MANILA") (PRN 0 (2 -LRB- "-
LRB-") (NPB 0 (3 NNP "AP")) (4 -RRB- "-RRB-"))) (NPB 2 (5
DT "A") (6 JJ "STRONG") (7 NN "EARTHQUAKE")) (8 VBD
"ROCKED") (NPB 2 (9 DT "THE") (10 NNPS "PHILIPPINES") (11
NN "ISLAND") (PP 0 (12 IN "OF") (NPB 0 (13 NNP
"MINDORO")))) (NPB 1 (14 JJ "EARLY") (15 NNP "TUESDAY"))
(CONJ 1 (VP 0 (16 VBG "KILLING") (NPB 1 (QP 0 (17 IN
"AT") (18 JJS "LEAST") (19 CD "TWO")) (20 NNS "PEOPLE")))
(21 CC "AND") (VP 0 (22 VBG "CAUSING") (NPB 1 (23 DT
"SOME") (24 NN "DAMAGE"))))) (NPB 0 (25 NNS
"AUTHORITIES")) (26 VBD "SAID"))
(SVP 1 (NPB 2 (1 DT "THE") (2 CD "3") (3 CD "15")) (4 VBP
"AM") (CONJ 2 (NPB 0 (5 NN "QUAKE")) (VP 0 (6 VBD "HAD")
(NPB 2 (7 DT "A") (8 JJ "PRELIMINARY") (9 NN "MAGNITUDE")
(PP 0 (10 IN "OF") (NPB 0 (11 CD "6.7"))))) (12 CC "AND")
(VP 0 (13 VBD "WAS") (ADJP 0 (14 VBN "CENTERED") (PP 0
(15 IN "NEAR") (NPB 0 (16 NN "BACO") (PP 0 (17 IN "ON")
(NPB 2 (18 JJ "NORTHERN") (19 NNP "MINDORO") (20 NNP
`
"ISLAND") (ADVP 1 (NPB 1 (QP 0 (21 RB "ABOUT") (22 CD
"75")) (23 NNS "MILES")) (24 RB "SOUTH") (PP 0 (25 IN
"OF") (NPB 0 (26 NNP "MANILA")))))))))) (PP 0 (27 VBG
"ACCORDING") (PP-A 0 (28 TO "TO") (CONJ 1 (NPB 2 (29 DT
"THE") (30 NNP "PHILIPPINE") (31 NNP "INSTITUTE") (PP 0
(32 IN "OF") (NPB 0 (33 NNP "VULCANOLOGY")))) (34 CC
"AND") (NPB 0 (35 NNP "SEISMOLOGY")))))))
(SVP 1 (NPB 3 (1 DT "THE") (2 NNP "U.S.") (3 JJ
"GEOLOGICAL") (4 NN "SURVEY") (PP 0 (5 IN "IN") (NPB 1 (6
NNP "MENLO") (7 NNP "PARK") (NPB 0 (8 NNP "CALIF.")))))
(9 VBD "PUT") (NPB 2 (NPB 2 (10 DT "THE") (11 NN "QUAKE")
(12 POS "'S")) (13 JJ "PRELIMINARY") (14 NN "MAGNITUDE"))
(PP 0 (15 IN "AT") (NPB 0 (16 CD "7.1"))))
(SVP 1 (NPB 2 (1 NNP "GOVERNOR") (2 NNP "RODOLFO") (3 NNP
"VALENCIA") (PP 0 (4 IN "OF") (NPB 3 (NPB 2 (5 DT "THE")
(6 NN "ISLAND") (7 POS "'S")) (8 JJ "ORIENTAL") (9 NNP
"MINDORO") (10 NN "PROVINCE")))) (11 VBD "SAID") (CONJ 1
(VP 2 (NPB 1 (12 CD "TWO") (13 NNS "PEOPLE")) (ADVP 0 (14
RB "REPORTEDLY")) (15 VBD "WERE") (VP-A 0 (16 VBN
"KILLED"))) (17 CC "AND") (SBAR 0 (18 IN "THAT") (VP 1
(NPB 3 (19 JJ "SEVERAL") (20 NNS "BUILDINGS") (21 CC
"AND") (22 NNS "BRIDGES")) (23 VBD "WERE") (VP-A 0 (24
VBN "DAMAGED") (PP 0 (25 IN "BY") (NPB 1 (26 DT "THE")
(27 NN "QUAKE"))))))))
(SVP 2 (VP 2 (NPB 1 (1 JJ "SEVERAL") (2 NNS "HOMES") (PP
0 (3 IN "NEAR") (NPB 1 (4 DT "THE") (5 NN "SHORE"))))
(ADVP 0 (6 RB "REPORTEDLY")) (7 VBD "WERE") (VP-A 0 (8
VBN "WASHED") (PRT 0 (9 RB "AWAY")) (PP 0 (10 IN "BY")
(NPB 1 (11 JJ "LARGE") (12 NNS "WAVES"))))) (NPB 0 (13
NNP "VALENCIA")) (14 VBD "TOLD") (NPB 3 (15 NNP "MANILA")
(16 NN "RADIO") (17 NN "STATION") (18 NNP "DZBB")))
(SVP 2 (VP 1 (NPB 1 (1 NNP "TELEPHONE") (2 NN "SERVICE"))
(3 VBD "WAS") (VP-A 0 (4 VBN "CUT"))) (NPB 0 (5 PRP
"HE")) (6 VBD "SAID"))
(SVP 1 (NPB 1 (1 DT "THE") (2 NN "QUAKE")) (3 VBD
"SWAYED") (NPB 1 (4 JJ "TALL") (5 NNS "BUILDINGS") (PP 0
(6 IN "IN") (NPB 0 (7 NNP "MANILA")))))
(SVP 1 (NPB 3 (1 NNP "INSTITUTE") (2 NN "SPOKESMAN") (3
NNP "ARIS") (4 NNP "JIMENEZ")) (5 VBD "SAID") (VP 1 (NPB
1 (6 DT "THE") (7 NN "QUAKE")) (8 VBD "OCCURRED") (PP 0
(9 IN "ON") (NPB 2 (10 DT "THE") (11 NNP "LUBANG") (12 NN
"FAULT") (NPB 0 (13 CD "ONE") (PP 0 (14 IN "OF") (NPB 2
(NPB 2 (15 DT "THE") (16 NN "AREA") (17 POS "'S")) (18
RBS "MOST") (19 JJ "ACTIVE"))))))))
(SVP 1 (NPB 1 (1 DT "A") (2 NN "MAGNITUDE") (ADJP 1 (3 CD
"6") (4 NN "QUAKE"))) (5 MD "CAN") (VP-A 0 (6 VB "CAUSE")
(NPB 1 (7 JJ "SEVERE") (8 NN "DAMAGE")) (SBAR 0 (9 IN
"IF") (VP 0 (10 VBN "CENTERED") (PP 0 (11 IN "UNDER")
`
(NPB 2 (12 DT "A") (13 JJ "POPULATED") (14 NN "AREA")))
(SBAR 0 (15 IN "WHILE") (VP 1 (NPB 2 (16 NN "AMGNITUDE")
(17 CD "7") (18 NN "QUAKE")) (19 VBZ "INDICATES") (NPB 2
(20 DT "A") (21 JJ "MAJOR") (22 NN "QUAKE") (ADJP 0 (23
JJ "CAPABLE") (PP 0 (24 IN "OF") (NPB 2 (25 JJ
"WIDESPREAD") (26 JJ "HEAVY") (27 NN "DAMAGE"))))))))))
)
```

Figure-3.2.2 (Parsed Text for Source Text Corpus)

3.2.3 Entity Grid

Barzilay and Lapata (2005) have demonstrated that Entity Grid is a good assessment of local coherence. They have also proposed that Text Coherency problem is learning-ranking problem. They have represented the text with Entity Grid. They have employed four symbols S, X, O, and – to represent text. Entity can be present in Text as Subject (S), Object (O) and neither as subject nor object (X). If Entity is not present at all then (–) symbol is used. Figure-3.2.3 is the Entity Grid representation for the source text 3.2.1(a). Entity Grid Relation transition probabilities (given in figure-3.2.6) of different length were used as feature for support vector. Few possible Entity Relation transition of length two are {X,O}, {O,--} and {--, S} and that of length three are {X,O,S}, {O,--,X} and {S,O,S}. The values of these features are in the interval [0,1], as they are the probabilities.

```
PROVINCE - - - - X - - - - -
WAVES - - - - - X - - - -
RADIO - - - - - O - - - -
SEISMOLOGY - - X - - - - - - -
GOVERNOR - - - - S - - - - -
SHORE - - - - - X - - - -
SERVICE - - - - - - S - - -
INSTITUTE - - X - - - - - S -
FAULT - - - - - - - - X -
TELEPHONE - - - - - - S - - -
AP - X - - - - - - - -
BUILDINGS - - - - S - - O - -
MANILA - S X - - O - X - -
RODOLFO - - - - S - - - - -
AREA - - - - - - - - X X
PHILIPPINE - - X - - - - - - -
```

```
`
DZBB - - - - - O - - - -
PHILIPPINES X O - - - - - - - -
ARIS - - - - - - - - S -
STATION - - - - - O - - - -
ISLAND X O X - X - - - - -
EARTHQUAKE - X - - - - - - - -
MINDORO - X X - X - - - - -
MILES - - X - - - - - - -
JIMENEZ - - - - - - - - S -
LUBANG - - - - - - - - X -
VULCANOLOGY - - X - - - - - - -
QUAKE X - X X X - - S S S
HE - - - - - - S - - -
CALIF. - - - X - - - - - -
MENLO - - - X - - - - - -
VALENCIA - - - - S S - - - -
HOMES - - - - - S - - - -
ROCKS X - - - - - - - - -
U.S. - - - S - - - - - -
MAGNITUDE - - O O - - - - - S
BACO - - X - - - - - - -
TUESDAY - X - - - - - - - -
PARK - - - X - - - - - -
PEOPLE - O - - S - - - - -
AUTHORITIES - S - - - - - - - -
SPOKESMAN - - - - - - - - S -
DAMAGE - O - - - - - - - X
AMGNITUDE - - - - - - - - - S
BRIDGES - - - - S - - - - -
SURVEY - - - S - - - - - -
```

Figure-3.2.3 (Entity Grid for Source Text Corpus)

3.2.4 PDTB Discourse Relation

Discourse is a coherent, structured group of sentences. Discourse must be coherent because a randomly ordered sequence of sentences does not form a discourse. A discourse relation is defined between two text spans. It can be realized through discourse connectives such as “however”, “because”, “As a result” etc. PDTB is a discourse level annotation over One million word Wall Street Journals. A discourse relation holds between two arguments, Arg1 and Arg2. Few possible discourse relation transitions of length two are {Temp.Arg1→ Exp.Arg2}, {Exp.Arg1→

`

Exp.Arg2} and {Comp.Arg2→ Exp.Arg2} and that of length three are {Temp.Arg1→ Comp.Arg2 → Exp.Arg2} and {Exp.Arg1→ Comp.Arg2 → Temp.Arg2}. Figure-3.2.4 is the output of source text in figure-3.2.1(a) by PDTB discourse parser by (Lin et al., 2009). Though, figure-3.2.4 having level 2 PDTB discourse relations, which is first converted to level 1 PDTB discourse relations before determining discourse relation transitions.

```
{NonExp_0_Arg1 BC-Philippines-Quake ,0194 Strong Quake
Rocks Philippines Island . NonExp_0_Arg1}
{NonExp_1_Arg1 {NonExp_0_Arg2_EntRel MANILA -LRB- AP -
RRB- A strong earthquake rocked the Philippines island of
Mindoro early Tuesday , killing at least two people and
causing some damage , authorities said . NonExp_0_Arg2}
NonExp_1_Arg1}
{NonExp_2_Arg1 {NonExp_1_Arg2_EntRel The 3:15 am quake
had a preliminary magnitude of 6.7 and was centered near
Baco on northern Mindoro Island , about 75 miles south of
Manila , according to the Philippine Institute of
Vulcanology and Seismology . NonExp_1_Arg2}
NonExp_2_Arg1}
{NonExp_3_Arg1 {NonExp_2_Arg2_EntRel The U.S. Geological
Survey in Menlo Park , Calif. , put the quake 's
preliminary magnitude at 7.1 . NonExp_2_Arg2}
NonExp_3_Arg1}
{Attr_0 {NonExp_4_Arg1 {NonExp_3_Arg2_EntRel Gov. Rodolfo
Valencia of the island 's Oriental Mindoro province said
Attr_0} {Exp_0_Arg1 two people reportedly were killed
Exp_0_Arg1} {Exp_0_conn_Conjunction and Exp_0_conn}
{Exp_0_Arg2 that several buildings and bridges were
damaged by the quake Exp_0_Arg2} . NonExp_3_Arg2}
NonExp_4_Arg1}
{NonExp_5_Arg1 {NonExp_4_Arg2_EntRel Several homes near
the shore reportedly were washed away by large waves ,
Valencia told Manila radio station DZBB . NonExp_4_Arg2}
NonExp_5_Arg1}
{NonExp_6_Arg1 {NonExp_5_Arg2_EntRel Telephone service
was cut , he said . NonExp_5_Arg2} NonExp_6_Arg1}
{NonExp_7_Arg1 {NonExp_6_Arg2_EntRel The quake swayed
tall buildings in Manila . NonExp_6_Arg2} NonExp_7_Arg1}
{Attr_1 {NonExp_8_Arg1 {NonExp_7_Arg2_EntRel Institute
spokesman Aris Jimenez said Attr_1} the quake occurred on
the Lubang fault , one of the area 's most active .
NonExp_7_Arg2} NonExp_8_Arg1}
```

```
`
{NonExp_8_Arg2_EntRel {Exp_1_Arg1 A magnitude 6 quake can
cause severe damage Exp_1_Arg1} {Exp_2_Arg1
{Exp_1_conn_Condition if Exp_1_conn} {Exp_1_Arg2 centered
under a populated area , Exp_2_Arg1} {Exp_2_conn_Contrast
while Exp_2_conn} {Exp_2_Arg2 amgnitude 7 quake indicates
a major quake capable of widespread , heavy damage
Exp_2_Arg2} . Exp_1_Arg2} NonExp_8_Arg2}
```

Figure-3.2.4 (Discourse Parsed Text for Source Text Corpus)

3.2.5 Noun Phrase Co-reference

Reconcile is an Noun Phrase Co-reference resolution system that was developed to provide a stable test-bed for researchers to implement reliable and quick novel ideas. It achieves roughly state-of-the-art performance on many of the most common co-reference resolution test sets, such as ACE and MUC-6, MUC-7. Reconcile comes ready out of the box to train and test on these common data sets as well as the ability to run on unlabeled texts (unseen text). Reconcile utilizes supervised machine learning classifiers from the Weka toolkit. It also used other language processing tools such as the Berkeley Parser and Stanford Named Entity Recognition System. The source language is Java, and it is freely available under the GPL.

Figure-3.2.5 is parsed Noun Phrase Co-reference text for source text (figure-3.2.1(a)) by Reconcile. We have used normalized hobb distance as a feature for support vector for training and testing pairs. We call hobb distance is the distance between noun phrase and its co-reference. For example, figure-3.2.5 have `CorefID="10"` at six places. It means that 10th Noun Phrase is co-referenced at six places. We find the distance of all co-reference from 10th Noun Phrase. We call these distances as hobb distance and maximum of those is called max hobb distnace. We divide the hobb distance by max hobb distance to find the normalized distance and use that as a feature of support vector.

`

```
<NP NO="0" CorefID="10">BC-Philippines-Quake,0194| Strong
Quake</NP> Rocks <NP NO="1" CorefID="4">Philippines
Island</NP>.
<NP NO="2" CorefID="2">MANILA (AP</NP>) <NP NO="3"
CorefID="3">A strong earthquake</NP> rocked <NP NO="4"
CorefID="4">the Philippines island of <NP NO="5"
CorefID="5">Mindoro</NP></NP> <NP NO="6"
CorefID="6">early Tuesday</NP>, killing <NP NO="7"
CorefID="7">at least two people</NP> and causing <NP
NO="8" CorefID="8">some damage</NP>, <NP NO="9"
CorefID="9">authorities</NP> said.
<NP NO="10" CorefID="10">The 3:15 am quake</NP> had <NP
NO="11" CorefID="23">a preliminary magnitude of <NP
NO="12" CorefID="12">6.7</NP></NP> and was centered near
<NP NO="13" CorefID="13">Baco on <NP NO="14"
CorefID="4">northern Mindoro Island</NP></NP>, <NP
NO="15" CorefID="15">about 75 miles</NP> south of <NP
NO="16" CorefID="40">Manila</NP>, according to <NP
NO="17" CorefID="41">the Philippine Institute of <NP
NO="18" CorefID="18">Vulcanology and
Seismology</NP></NP>.
<NP NO="19" CorefID="19">The U.S. Geological Survey in
<NP NO="20" CorefID="20">Menlo Park</NP></NP>, <NP
NO="21" CorefID="4">Calif.</NP>, put <NP NO="23"
CorefID="23"><NP NO="22" CorefID="10">the quake</NP>'s
preliminary magnitude</NP> at <NP NO="24"
CorefID="24">7.1</NP>.
<NP NO="25" CorefID="34">Gov. Rodolfo Valencia of <NP
NO="27" CorefID="27"><NP NO="26" CorefID="4">the
island</NP>'s Oriental Mindoro province</NP></NP> said
<NP NO="28" CorefID="28">two people</NP> reportedly were
killed and that <NP NO="29" CorefID="29">several
buildings and bridges</NP> were damaged by <NP NO="30"
CorefID="10">the quake</NP>.
<NP NO="31" CorefID="31">Several homes near <NP NO="32"
CorefID="32">the shore</NP></NP> reportedly were washed
away by <NP NO="33" CorefID="33">large waves</NP>, <NP
NO="34" CorefID="34">Valencia</NP> told <NP NO="35"
CorefID="35">Manila radio station DZBB</NP>.
<NP NO="36" CorefID="36">Telephone service</NP> was cut,
<NP NO="37" CorefID="34">he</NP> said.
<NP NO="38" CorefID="10">The quake</NP> swayed <NP
NO="39" CorefID="39">tall buildings</NP> in <NP NO="40"
CorefID="40">Manila</NP>.
<NP NO="42" CorefID="42"><NP NO="41"
CorefID="41">Institute</NP> spokesman Aris Jimenez</NP>
said <NP NO="43" CorefID="10">the quake</NP> occurred on
<NP NO="44" CorefID="44">the Lubang fault</NP>, <NP
```

```
`
NO="45" CorefID="45">one of <NP NO="46" CorefID="40">the
area</NP>'s most active</NP>.
<NP NO="47" CorefID="47">A magnitude 6 quake</NP> can
cause <NP NO="48" CorefID="48">severe damage</NP> if
centered under <NP NO="49" CorefID="49">a populated
area</NP>, while <NP NO="50" CorefID="50">amgnitude 7
quake</NP> indicates <NP NO="51" CorefID="51">a major
quake capable of <NP NO="52" CorefID="52">widespread,
heavy damage</NP></NP>.
```

Figure-3.2.5 (NP Co-reference Text for Source Text Corpus)

3.2.6 Problem Formulation

As discussed in Chapter 2 that Entity Grid Model by Barzilay and Lapata (2005; 2008) is based on the Entity Relation Transition of different length and Discourse Relation Model by (Lin et. al., 2011) is based on Discourse Relation Transition Pattern of different length. For given source text (given in figure-3.2.1(a)), Entity Relation Transition of different length can be determined by Entity Grid output (given in figure-3.2.3) and Discourse Relation Transition of different length can be determined by the PDTB Discourse Parser's output (given in figure-3.2.4).

$$\text{Entity Relation Transition Probability} = \frac{Count\ of\ specific\ N{-}gram\ Discourse\ Entity\ Transition}{Count\ of\ total\ N{-}gram\ Discourse\ Entity\ Transition}$$

$$\text{Discourse Relation Transition Probability} = \frac{Count\ of\ specific\ N{-}gram\ Discourse\ Entity\ Transition}{Count\ of\ total\ N{-}gram\ Discourse\ Entity\ Transition}$$

$$\text{Normalized Hobb Distance} = \frac{Hobb\_Distance}{Max\_Hobb\_Distance}$$

Where Hobb_Distance is absolute difference between noun phrase and its co-reference and Max_Hobb_Distance is maximum among all Hobb_Distance for a particular co-reference.

Figure-3.2.6 (Problem Formulation)

`

3.2.7 Algorithm to calculate Automatically Evaluating Text Coherence

We have seen in this section that many tools and software's were used for generating different features for support vectors for automatically evaluating text coherence task. So our system requires some preprocessing on training and testing dataset (e.g. source text in figure-3.2.1(a)) to automatically evaluating text coherence. We have divided the working of our system in three steps.

Step 1 is the preprocessing step, which create sentence ordered permutations (e.g. figure-3.2.1(b)) for each training and testing datasets. This step also create the POS parsed text (e.g. figure-3.2.2), entity grid representation of text (e.g. figure-3.2.3), discourse parsed text (e.g. figure-3.2.4) and noun phrase co-reference parsed text (e.g. figure-3.2.6). Step 2 involves finding the open words and performs stemming on these words in both source text and its permutations for training datasets. Now features are generated using entity relation transitions and discourse relation transition of different lengths. Finally we have determined the hobb distances to generate features based on Noun Phrase co-reference. Using all these calculated feature values, we create the support vector for each and every training dataset pairs. We write all these support vectors (figure-4.4.2) for each pair of training datasets and use this as a input for SVM to generate a best model (figure-4.4.3) for best automatically evaluating text coherence. In step 3 we do the features generation for test data pair as in step 2 and then write all generated feature vectors to a file and use this file together with model file as input to SVM for generating prediction file (figure-4.4.4) with ranking score for each test pair datasets.

`

Step 1: Preprocessing for each text corpus (training/testing)

- Create permutation by shuffling the sentence ordering of the text
- Create Parsed file of each text corpus including all permutations
- Create Grid file of each text corpus including all permutations
- Create Discoursed Parsed file of each text corpus including all permutations
- Create Noun Phrase Co-reference file of each text corpus including all permutations

Step 2: Train the System (Create the features for each text corpus and its permutations)

- For each pair of source text corpus and its permutations
    - Perform Stemming of Open Words
    - Calculate the entity relation transitions at the sentence-to-sentence level
    - Calculate the discourse relation transition of different length
    - Calculate the normalized distance of NPs with its co-reference
- Use above calculated feature values to create support vectors (input for SVM)
- Create model for best text coherence system using SVM and support vectors of training data

Step 3: Test the System (Create the features for each text corpus and its permutations)

- For each pair of source text corpus and its permutations
    - Perform Stemming of Open Words
    - Calculate the entity relation transitions at the sentence-to-sentence level
    - Calculate the discourse relation transition of different length
    - Calculate the normalized distance of NPs with its Co-reference
- Use above calculated feature values to create support vectors (input for SVM)
- Create prediction for test corpus system using SVM and support vectors of test data

Figure-3.2.7 (Algorithm to calculate Automatically Evaluating Text Coherence)

3.3 Summary

In this chapter, we have identified the drawbacks in the existing techniques used to solve the automatically evaluating Text Coherence problem. With respect to this, we have introduced our approach to automatically evaluating Text Coherence by proposing an efficient technique and explain it in detail with examples. We have also given the problem formulation and described our algorithms step by step in detail. We have explained how our approach is combined with the earlier work in this task.

`

Chapter 4: Implementation and Experimental Results

4.1 Environmental Setup

We have used the following configuration while finding the experimental results

4.1.1 Hardware Configuration

Processor	:	Intel Core 2 Duo - P8600
Processor Speed	:	2.40GHz
Main Storage	:	4GB RAM
Hard Disk Capacity	:	360GB
Monitor	:	Dell 15”5’ Color

4.1.2 Software Configuration

Operating System	:	Ubuntu 11.10
Programming Language	:	C
Libraries	:	RubyGems 1.9
Softwares	:	Stanford POS Tagger, PDTB Discourse Parser, Reconcile

4.2 Datasets

In our proposed system, we had assumed two real world datasets on Earthquake and Accidents provided by We the North American News Corpus and the National Transportation Safety Board. Dataset includes the articles of earthquake and

`

narratives of accidents which are used to find the experimental results of the comparison of our proposed technique with the other earlier techniques. These two datasets are further elaborated as given below in the table 4.2.

		Earthquakes	Accidents
Train	Number of Articles	97	100
	Number of Pairs	1856	1978
Test	Number of Articles	97	98
	Number of Pairs	1906	1946

Table-4.2 (Number of Pairs for Training and Testing)

For the training and testing datasets we have used the pair of source text and its sentence ordered permutation for learning ranking problem of text coherence. Different source text has different number of permutations depending on the number of statements in source text. We have taken at most 20 permutation for each source text. Few source texts have less than 20 permutations due to less number of statements in it.

4.3 Human Evaluation

As the aim of most of the Natural Language Processing task is to achieve the human level performance. Therefore in order to measure the performance of our Automatically Evaluating Text Coherence we compare the performance of our system with that of human evaluator. For the fair comparison, human evaluator has been given the same set of data who are not the authors of the paper. As we have discussed

`

in earlier sections that automatically evaluating text coherence problem is same as the text ordering ranking task. When we discuss any application of Natural Language Processing then two key questions about that task need to be addresses.

(1) Up to what extent the source text is more coherent than its permutation? And

(2) What is the accuracy of the human evaluator on the same dataset?

First question validate the correctness of this synthetic task, while the second question obtain the accuracy for upper bound for evaluation. Therefore human evaluators are required to answer these questions.

(Lin et. al., 2011) have assigned randomly 50 source text/permutation pairs from each of the Earthquakes, and Accidents training sets to human evaluators, those are not the authors of their paper. It has been observed that some of the original/source texts have formulaic structures in their initial sentences that help the evaluator to determine the correct ordering. For example, sources from the Earthquakes data always begin with a headline sentence and a location newswire sentence.

Therefore for correct accuracy determination (Lin et. al., 2011) have removed these sentences from the source and permuted texts to avoid the human evaluator judging based on these clues instead of textual coherence. They have employed two human evaluators for ranking task. When both human evaluators rank a source text higher than its permutation, it can be interpreted that both human evaluators are agreeing that the source text is more coherent than the permutation. Table 4.3 shows the inter-human evaluator agreements.

Earthquakes	Accidents	Overall
90.0	94.0	92.0

Table-4.3 (Accuracy by Human Evaluator [4])

Though human evaluator's performance is not perfect, still it is suggesting fair upper bound limits on system performance. It has been observed that the Accidents data set is relatively easier to rank, as it has a higher upper bound than the Earthquake datasets.

`

4.4 Support Vector Machine (SVM)

SVM is an implementation of Vapnik's Support Vector Machine for the problem of pattern recognition, regression and learning a ranking function. The algorithm for SVM has scalable memory requirements and can handle problems with many thousands of support vectors efficiently.

For learning ranking problem, the goal of SVM is to learn a function from preference examples, so that it predicts a new set of objects with their ranking score as accurately as possible. Such ranking problems naturally occur in applications like search engines, recommender systems and to find well written essay.

SVM has been used on a large range of problems, including text classification, pattern recognition, image recognition tasks, bioinformatics and medical applications. The main features of the SVM are the following [15]:

fast optimization algorithm

- Solves classification and regression problems.
- Solves ranking problems
- efficiently computes Leave-One-Out estimates of the error rate, the precision, and the recall
- allows restarts from specified vector of dual variables
- can train SVMs with cost models and example dependent costs
- handles many thousands of support vectors
- computes XiAlpha-estimates of the error rate, the precision, and the recall
- handles several hundred-thousands of training examples
- uses sparse vector representation

4.4.1 SVM RANK

SVM RANK is an instance of SVM for efficiently training Ranking and learning.

`

SVM RANK learns an unbiased linear classification rule. The file format of the training and test files is the same as for SVM, with the exception that the lines in the input files have to be sorted by increasing qid. Lines started with # can be ignored as comment. Each lines of input represent one training example and are of the following format [15]:

```
<line> .=. <target> qid:<qid> <feature>:<value>
<feature>:<value> ... <feature>:<value> # <info>
<target> .=. <float>
<qid> .=. <positive integer>
<feature> .=. <positive integer>
<value> .=. <float>
<info> .=. <string>
```

Figure-4.4.1 (Training Example format for SVM)

The target value and each of the (feature, value) pairs are separated by a space. (Feature, value) pairs MUST be ordered by increasing feature number and qid. Features with value zero can be ignored. The target value defines the rank of the examples for each query and used to generate pairwise preference constraints. A preference constraint is included for all pairs of examples given below, for which the target value differs. Generation of constraints can be restricted by using the special feature ‘qid’. Two examples are considered for a pairwise preference constraint only if the value of ‘qid’ is the same for both. For example,

```
1 qid:1 2:0.5 4:0.2 5:0.2 # 1a
2 qid:1 1:0.3 3:0.4 4:0.1 # 1b
1 qid:1 1:0.2 2:0.1 4:0.4 5:0.3 # 1c
3 qid:1 1:0.6 3:1 4:0.3 # 1d
1 qid:2 2:0.2 3:0.1 4:0.5 # 2a
2 qid:2 1:1 2:0 3:1 4:0.4 5:0 # 2b
3 qid:2 2:0.6 4:0.3 5:0.2 # 2c
1 qid:2 1:0 3:0.3 4:0.2 5:0.1 # 2d
```

Figure-4.4.2 (set of Training example for pairwise constraints)

`

The following set of pair-wise constraints is generated

```
1d>1a, 1d>1b, 1d>1c, 1b>1a, 1b>1c, 2c>2a, 2c>2b, 2c>2d, 2b>2a, 2b>2d
```

The result of SVM RANK learning is the model that is learned from the training example data. The model is written to model.dat file. For predictions on test examples, SVM RANK classifies these examples using model.dat file.

For each line in test file, the predicted ranking score is written to the file predictions. For each line in test examples, there is a line in prediction file with its ranking score.

4.4.2 Support Vector for SVM

Input file for SVM contains the support vector as given in figure-4.4.1 of the training and test datasets examples. Each of the following lines represents one training/test example and is of the following format:

```
<line> .=. <target> <feature>:<value> <feature>:<value> ...
<feature>:<value> # <info>
<target> .=. +1 | -1 | 0 | <float>
<feature> .=. <integer> | "qid"
<value> .=. <float>
<info> .=. <string>
```

The string `<info>` can be used to pass additional information to the kernel to identify source text and permutation. Zero target values are ignored. Figure-4.4.2 is the sample support vectors for few test pairs (source text and its sentence ordered permutations) for example given in figure-3.2.1(a). We have used three kinds of `<feature>` in figure-4.4.2. First is N-gram Entity Relation Transitions and second is the N-gram Discourse Relation Transitions. We have found best accuracy for N <= 3. Value associated with these features is the transition probabilities and they were the features with the existing approaches. Third type of feature is the Noun Phrase Co-reference and its value is the normalized hobb distance between noun phrase and its co-reference.

```
`
2 qid:1890 1:0.052174 2:0.023913 3:0.073913 4:0.850000
5:0.007246 7:0.002415 8:0.041063 9:0.002415 10:0.002415
12:0.004831 13:0.062802 14:0.050725 15:0.019324
16:0.717391 17:0.002717 20:0.002717 25:0.002717
28:0.035326 35:0.002717 37:0.002717 39:0.002717
40:0.057065 41:0.043478 43:0.019022 44:0.043478
45:0.600543 # apwsE941114.0478-2-2-0.perm-1
1 qid:1890 1:0.052174 2:0.023913 3:0.073913 4:0.850000
5:0.004831 8:0.050725 11:0.002415 12:0.024155 13:0.060386
14:0.707729 18:0.005435 22:0.002717 24:0.054348
34:0.002717 36:0.024457 38:0.581522 # apwsE941114.0478-2-
2-0.perm-10
2 qid:1891 1:0.052174 2:0.023913 3:0.073913 4:0.850000
5:0.007246 7:0.002415 8:0.041063 9:0.002415 10:0.002415
12:0.004831 13:0.062802 14:0.050725 15:0.019324
16:0.717391 17:0.002717 20:0.002717 25:0.002717
28:0.035326 35:0.002717 37:0.002717 39:0.002717
40:0.057065 41:0.043478 43:0.019022 44:0.043478
45:0.600543 # apwsE941114.0478-2-2-0.perm-1
1 qid:1891 1:0.052174 2:0.023913 3:0.073913 4:0.850000
7:0.004831 8:0.050725 9:0.002415 12:0.070048 13:0.053140
14:0.014493 22:0.002717 23:0.002717 24:0.005435
31:0.002717 37:0.070652 38:0.051630 40:0.010870 #
apwsE941114.0478-2-2-0.perm-11
2 qid:1892 1:0.052174 2:0.023913 3:0.073913 4:0.850000
5:0.007246 7:0.002415 8:0.041063 9:0.002415 10:0.002415
12:0.004831 13:0.062802 14:0.050725 15:0.019324
16:0.717391 17:0.002717 20:0.002717 25:0.002717
28:0.035326 35:0.002717 37:0.002717 39:0.002717
40:0.057065 41:0.043478 43:0.019022 44:0.043478
45:0.600543 # apwsE941114.0478-2-2-0.perm-1
1 qid:1892 1:0.052174 2:0.023913 3:0.073913 4:0.850000
7:0.002415 8:0.048309 10:0.004831 12:0.007246 13:0.045894
14:0.016908 15:0.062802 23:0.002717 26:0.002717
30:0.002717 34:0.002717 37:0.008152 39:0.046196
40:0.002717 41:0.013587 42:0.002717 # apwsE941114.0478-2-
2-0.perm-12
2 qid:1893 1:0.052174 2:0.023913 3:0.073913 4:0.850000
5:0.007246 7:0.002415 8:0.041063 9:0.002415 10:0.002415
12:0.004831 13:0.062802 14:0.050725 15:0.019324
16:0.717391 17:0.002717 20:0.002717 25:0.002717
28:0.035326 35:0.002717 37:0.002717 39:0.002717
40:0.057065 41:0.043478 43:0.019022 44:0.043478
45:0.600543 # apwsE941114.0478-2-2-0.perm-1
1 qid:1893 1:0.052174 2:0.023913 3:0.073913 4:0.850000
6:0.002415 7:0.004831 8:0.043478 10:0.026570 11:0.002415
13:0.021739 14:0.062802 20:0.002717 23:0.002717
```

`

```
32:0.027174 34:0.002717 38:0.024457 39:0.065217 #
apwsE941114.0478-2-2-0.perm-13
2 qid:1894 1:0.052174 2:0.023913 3:0.073913 4:0.850000
5:0.007246 7:0.002415 8:0.041063 9:0.002415 10:0.002415
12:0.004831 13:0.062802 14:0.050725 15:0.019324
16:0.717391 17:0.002717 20:0.002717 25:0.002717
28:0.035326 35:0.002717 37:0.002717 39:0.002717
40:0.057065 41:0.043478 43:0.019022 44:0.043478
45:0.600543 # apwsE941114.0478-2-2-0.perm-1
1 qid:1894 1:0.052174 2:0.023913 3:0.073913 4:0.850000
8:0.057971 10:0.004831 11:0.050725 12:0.019324
13:0.055556 14:0.734300 21:0.005435 23:0.002717
24:0.040761 34:0.005435 36:0.057065 38:0.013587
39:0.032609 40:0.644022 # apwsE941114.0478-2-2-0.perm-14
```

Figure-4.4.2 (Sample Support Vectors for test/train Text Corpus Pair)

4.4.3 SVM model for best Text Coherence

As described above that SVM RANK is the learning-ranking tool. It based on supervised learning. It learns from the training examples and the rank test examples based on what it learns from the training examples. Input for the SVM RANK is the set of support vectors for the example pairs in training dataset. In each pair of training dataset one text (more coherent text) is given higher rank than other (low coherent text) to allow the SVM to learn the model for the system. Figure-4.4.3 is the model learned from the SVM RANK after learning examples in the training datasets.

```
SVM-light Version V6.20
0 # kernel type
3 # kernel parameter -d
1 # kernel parameter -g
1 # kernel parameter -s
1 # kernel parameter -r
empty# kernel parameter -u
85 # highest feature index
2 # number of training documents
2 # number of support vectors plus 1
0 # threshold b, each following line is a SV (starting with
alpha*y)
1 1:-0.037122004 2:-0.014853534 3:-0.081966519 4:0.13394354
5:-0.013915306 6:-0.00012894755 7:-0.01013522 8:0.057118915
```

`

```
9:0.00034828149 10:0.0019695167 11:0.015615575 12:0.020479314
13:-0.0017661332 14:-0.0051014926 15:0.010021333
16:0.064363495 17:-0.090607196 18:-0.051454607 19:-0.13367696
20:0.13687003 21:-0.00034530606 22:-0.00044533834 23:-
0.0017635875 24:-0.0090493234 25:-0.00030129967 26:-
1.5102041e-05 27:-0.00097856065 28:0.0031480347
29:0.0051352307 30:-0.0010360903 31:0.0029490869 32:-
0.016961375 33:0.0052496884 34:-0.0066009881 35:0.015035317
36:0.13161498 37:-0.00028037594 38:-6.2352308e-05
39:0.0054422878 40:-0.005839549 41:-5.3748659e-05
42:0.00055013964 43:0.001015392 44:0.0014893878 45:-
0.0012609345 46:8.383461e-05 47:0.00079791615 48:0.027306035
49:0.0043179914 50:-0.0057275081 51:-0.0020905905
52:0.048808761 53:-0.0020996241 54:-0.0013595703 55:-
0.0026530074 56:0.0088017825 57:-0.0011006874 58:-0.0015550376
59:-0.0012608164 60:0.0038498067 61:-0.003528507 62:-
0.00036784104 63:0.0028321587 64:0.023262899 65:-0.013070022
66:-0.012713782 67:-0.013320044 68:0.18480353 69:-0.021136424
70:0.00021674552 71:-0.017631836 72:0.0045656594
73:0.00059419964 74:-0.00018457578 75:-0.0060584638
76:0.0056261434 77:-0.014377604 78:-0.012397411 79:-
0.0087651554 80:0.064472772 81:-0.1728847 82:-0.058815531 83:-
0.21099305 84:0.077121839 #
```

Figure-4.4.3 (Sample Model file for Coherent Text)

4.4.4 SVM prediction for test data

As described above that after learning model for the system, SVM RANK predicts the ranking on the test examples. Similarly for our automatic evaluating text coherence task, SVM RANK learn model for the best Text Coherence evaluation and apply this model to support vectors of test dataset to predict ranking based on text coherence. Figure-4.4.4 is the sample prediction for some test example and its different permutations. It can be seen easily that every alternate prediction is the ranking score for source text and next is the ranking for its permutation. It is also clear from this example that in most of the cases, source text having higher ranking score than its permutations.

```
0.27488188
0.27417934
0.27488188
0.27224022
```

`

```
0.27488188
0.27287580
0.27488188
0.27393578
0.27488188
0.26937219
0.27488188
0.26915587
0.27488188
0.27102609
0.27488188
0.27415887
0.27488188
0.27103075
0.27488188
0.27435507
0.27488188
0.27191210
0.27488188
0.27339202
0.27488188
0.27453685
0.27488188
0.27603847
0.27488188
0.27045820
0.27488188
0.27269032
0.27488188
0.27343367
0.27488188
0.27396089
0.27488188
0.27243140
0.27543198
0.27497060
0.27543198
```

Figure-4.4.4 (Sample Prediction for few test Text Corpus Pair)

4.5 Analysis and Results

In this section, we present the experimental analysis and results on the performance of our proposed tree algorithm. We compare the performance of our proposed approach with that of existing approaches for this task of automatically evaluating text

`

coherence. The accuracy of our approach is better than that of baseline and its close to the accuracy of human evaluators. The experiments are pursued on several datasets on earthquake and accidents provided by [18].

4.5.1 Analysis and Discussion

We have considered at most 20 permutations for each source text and some source text have less than 20 permutations due to less number of sentences in training/testing datasets. We have observed that the accuracy of the system is related with the value of N in N-gram Entity Relation Transition and in N-gram Discourse Relation Transition. It has also been observed that the accuracy of the system is also closely related to the word's Term Frequency (TF) in the training and testing datasets. It has been found that best accuracy is noted on optimal setting with N <= 3 and TF >= 2. We have carried out our experiment with N = 3 and TF > 2. In case of Noun Phrase Co-reference, we have only considered that Noun Phrase which is having more than four Co-reference in testing datasets. Table-4.5.2 shows our experimental results of our approach together with prior approaches for this task. Best achievable accuracy of automatic evaluating text coherent task is also based on the type of dataset we are dealing with. For e.g. Best accuracy in case of Accidents is higher than the accuracy of Earthquake.

4.5.2 Accuracy Comparison

When we compare the accuracy of our full system with that of earlier approaches we have found that accuracy of our system is better than that of baseline of our work. Our system performs well in both datasets [18]. It is clear from the results in Table-4.5.2 that combined model "Entity Grid + Discourse Relation + NPs Co-reference Model" performs better than accuracy when only using "Entity Grid Model" and using "Entity Grid Model" with "Discourse Relation Model".

`

	Earthquake	**Accident**	**Overall**
Entity Grid Model	82.42	85.78	84.10
Entity Grid + Discourse Relation Model	87.20	89.83	88.51
Entity Grid + Discourse Relation + NPs Co-reference Model	88.74	90.86	89.80

Table-4.5.2 (Experimental Results)

It is also clear from the figure-4.5.2 that accuracy of our system is closest to the accuracy of human evaluators, which is the goal of most of the Natural Language Processing applications. Overall accuracy of Accident dataset is higher than the overall accuracy of Earthquake datasets. This indicates that the same results different accuracy on different datasets based on its complexity. As on the same datasets, accuracy of human evaluators is still the highest, which indicates that still there is some scope of work to further improve the accuracy of the system. In next chapter we have suggested few more enhancements to our proposed approach to further improve the accuracy of automatically evaluating text coherence.

`

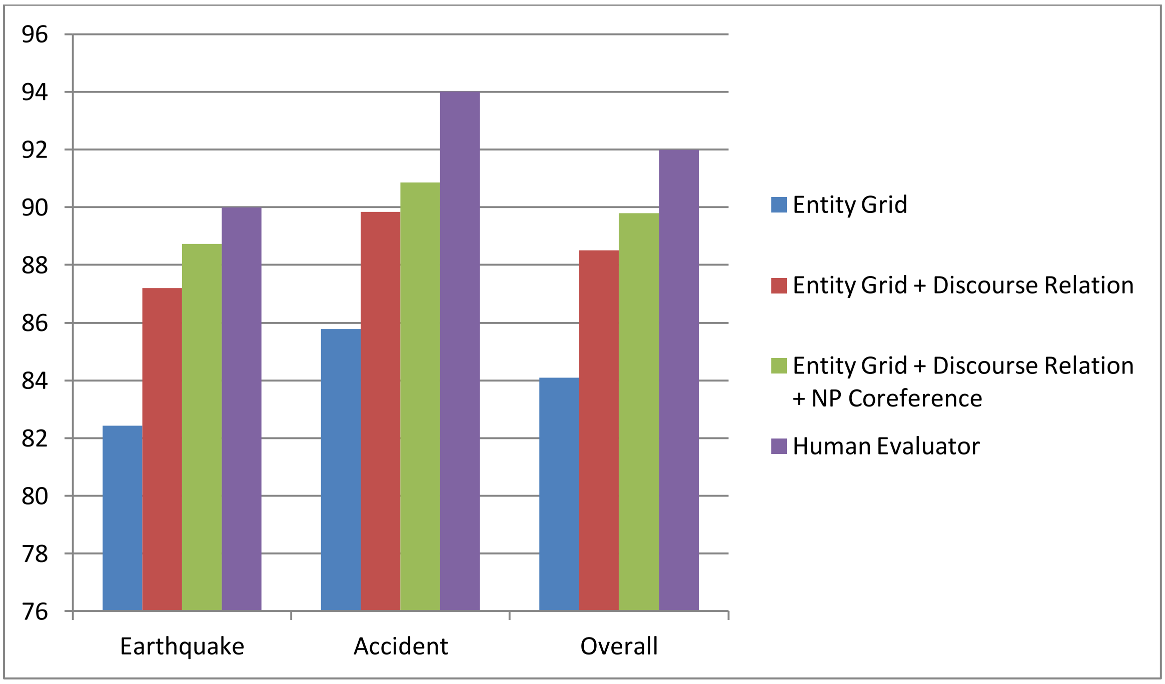

Figure-4.5.2 Comparison of accuracy of different technique and human evaluator

4.6Summary

In this chapter we have introduced the environmental setup which we have used while making the experimental results. In addition to this I have also explain the types of dataset which I have used and put some light on their statistics in our experiment. Finally we have mentioned all the analysis and experimental results that we have got and found that our proposed approach is more accurate in comparison to earlier approaches to this task.

`

Chapter 5: Conclusion & Future Scope

5.1 Conclusion

In this thesis, a new technique for automatic evaluating text coherence is proposed. We have proposed a new model for discourse coherence motivated from Centering Theory (Grosz et al., 1995) that NPs should not be too far from its co-reference. We have taken the normalized distance of NPs with its co-reference, which is referring to more than two NPs. We believe that coherent text preferentially follow certain discourse structures, which can be captured in and represented by the patterns of discourse relation transitions. We first combine the sequence of discourse relation transitions with entity based model, which results in improvement in accuracy of coherent system. We combine this model with our Noun Phrase Co-reference model to improve accuracy further. In our approach we have taken n-gram subsequences [n = 3] of transitions per term in the discourse role matrix to distinguish coherence from incoherence. When applied to distinguish a source text from a sentence-reordered permutation, though our model achieve the higher accuracy than current state-of-the-art system. Experiments validate our claim, with a combined model outperforming both single models.

The idea of modeling coherence with discourse relations transitions, we may apply this model for multi document summarization, differentiating a text from its permutation (i.e., the sentence ordering of the text is shuffled) and identifying a more well-written essay from a pair.

5.2 Future Scope

We believe that as Noun Phrase Co-reference contributes to improve the accuracy, taking Pronoun Phrase Co-reference will also lead to improvement in accuracy.

`

We can also use "Semantic Vectors" (http://code.google.com/p/semanticvectors/) as it can be used for many semantic (concept-aware) matching tasks such as automatic thesaurus generation, knowledge representation, and concept matching. We believe that uses of "Semantic Vectors" will lead to more improvement in accuracy.

We can also use the features based on Uryupina (2003) to distinguishing discourse-new from discourse-old noun phrases.

In general, we feel that as a young research field in Text Coherence, automatic evaluating text coherence using NPs Co-reference has achieved improvement in accuracy of the system and claimed a wide range of applications in Natural Language Processing. However, in-depth research is still needed on several critical issues so that the field may have its long lasting and deep impact in Natural Language Processing applications.

`

REFERNCES

1. Manoj Kumar and Rajiv Ratn Shah. 2012. A Comparative study of Automatically Evaluating Text Coherence. In Proceeding of International Conference on Computer Science & Engineering (ICCSE-2012) Nainital (India), ISBN:978-93-81693-96-4, 19 May 2012.

2. Regina Barzilay and Mirella Lapata. 2005. Modeling local coherence: an entity-based approach. In Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL 2005), pages 141–148, Morristown, NJ, USA. Association for Computational Linguistics.

3. Regina Barzilay and Mirella Lapata. 2008. Modeling local coherence: An entity-based approach. Computational Linguistics, 34:1–34, March.

4. Ziheng Lin, Hwee Tou Ng and Min-Yen Kan, Automatically Evaluating Text Coherence Using Discourse Relations. In Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics, pages 997–1006, Portland, Oregon, June 19-24, 2011. c 2011 Association for Computational Linguistics .

5. Ziheng Lin, Min-Yen Kan and Hwee Tou Ng. Recognizing Implicit Discourse Relations in the Penn Discourse Treebank. In Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing, pages 343–351, Singapore, 6-7 August 2009. c 2009 ACL and AFNLP .

6. Micha Elsner and Eugene Charniak . Coreference-inspired Coherence Modeling. In Proceedings of ACL-08: HLT, Short Papers (Companion Volume), pages 41–44, Columbus, Ohio, USA, June 2008. c 2008 Association for Computational Linguistics.

7. Shasha Liao and Ralph Grishman. Large Corpus-based Semantic Feature Extraction for Pronoun Coreference .

8. Barbara J. Grosz, Aravind K. Joshi, and Scott Weinstein. 1995. Centering: A framework for modeling the local coherence of discourse. Computational Linguistics, 21(2):203–225

`

9. Olga Uryupina. 2003. High-precision identification of discourse new and unique noun phrases. In Proceedings of the ACL Student Workshop, Sapporo.

10. Ziheng Lin, Hwee Tou Ng, and Min-Yen Kan. 2010. A PDTB-Styled End-to-End Discourse Parser.

11. Ziheng Lin, Min-Yen Kan and Hwee Tou Ng. Recognizing Implicit Discourse Relations in the Penn Discourse Treebank. In Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing, pages 343–351, Singapore, 6-7 August 2009. c2009 ACL and AFNLP.

12. T Donaldson, M Makuta, R Cohen. An Integrated Approach to Evaluating Text Coherence and its Application to the Prevention of Reader Misconceptions.

13. http://www.wikipedia.org/

14. Liddy, E. D. In Encyclopedia of Library and Information Science, 2nd Ed. Marcel Decker, Inc.

15. http://svmlight.joachims.org/

16. http://www.cs.utah.edu/nlp/reconcile/

17. Rashmi Prasad, et al., The Penn Discourse Treebank 2.0, LREC 2008

18. Press articles about earthquakes from the North American News Corpus, and narratives from the National Transportation Safety Board.

`

APPENDIX A: CODING

```
/*          Automatic Evaluating the Text Coherence        */
/*  This file generates the input file for SVM   */
/*  This generates the vectors for each test examples   */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include <ctype.h>      /* for isupper, islower, tolower */

/* Used the Porter stemming algorithm, coded up in ANSI C*/

#define TRUE 1
#define FALSE 0
#define NOT_FOUND -1

#define MAX_COREF_COUNT 20    /* Update these #DEFINE according to needs*/
#define MAX_LINE_LEN 1024
#define MAX_FILE_NAME_LEN 100
#define MAX_DRM_ENTRY_LEN 1024
#define MAX_NUM_TERM 1024
#define MAX_WORD_LEN 1024
#define MAX_DISC_REL 13
#define MATRIX_SIL_VAL 2
#define MAX_ERM_ENTRY 2
#define MAX_ENT_REL 4
#define MAX_COREF_REL 500


/* The main part of the stemming algorithm starts here. b is a buffer
   holding a word to be stemmed. The letters are in b[k0], b[k0+1] ...
   ending at b[k]. k is readjusted downwards as the stemming progresses.
   Zero termination is not in fact used in the algorithm.

   Note that only lower case sequences are stemmed. Forcing to lower case
   should be done before stem(...) is called.
*/

static char * b;       /* buffer for word to be stemmed */
static int k,k0,j;     /* j is a general offset into the string */

/* cons(i) is TRUE <=> b[i] is a consonant. */

static int cons(int i)
{  switch (b[i])
   {  case 'a': case 'e': case 'i': case 'o': case 'u': return FALSE;
      case 'y': return (i==k0) ? TRUE : !cons(i-1);
      default: return TRUE;
   }
}

/* m() measures the number of consonant sequences between k0 and j. if c is
   a consonant sequence and v a vowel sequence, and <..> indicates arbitrary
```

```
`
  presence,

    <c><v>       gives 0
    <c>vc<v>     gives 1
    <c>vcvc<v>   gives 2
    <c>vcvcvc<v> gives 3
    ....
*/

static int m()
{  int n = 0;
  int i = k0;
  while(TRUE)
  {  if (i > j) return n;
    if (! cons(i)) break; i++;
  }
  i++;
  while(TRUE)
  {  while(TRUE)
    {  if (i > j) return n;
        if (cons(i)) break;
        i++;
    }
    i++;
    n++;
    while(TRUE)
    {  if (i > j) return n;
      if (! cons(i)) break;
      i++;
    }
    i++;
  }
}

/* vowelinstem() is TRUE <=> k0,...j contains a vowel */

static int vowelinstem()
{  int i; for (i = k0; i <= j; i++) if (! cons(i)) return TRUE;
  return FALSE;
}

/* doublec(j) is TRUE <=> j,(j-1) contain a double consonant. */

static int doublec(int j)
{  if (j < k0+1) return FALSE;
  if (b[j] != b[j-1]) return FALSE;
  return cons(j);
}

/* cvc(i) is TRUE <=> i-2,i-1,i has the form consonant - vowel - consonant
  and also if the second c is not w,x or y. this is used when trying to
  restore an e at the end of a short word. e.g.

    cav(e), lov(e), hop(e), crim(e), but
    snow, box, tray.

*/
```

`

```
static int cvc(int i)
{  if (i < k0+2 || !cons(i) || cons(i-1) || !cons(i-2)) return FALSE;
   {  int ch = b[i];
      if (ch == 'w' || ch == 'x' || ch == 'y') return FALSE;
   }
   return TRUE;
}

/* ends(s) is TRUE <=> k0,...k ends with the string s. */

static int ends(char * s)
{  int length = s[0];
   if (s[length] != b[k]) return FALSE; /* tiny speed-up */
   if (length > k-k0+1) return FALSE;
   if (memcmp(b+k-length+1,s+1,length) != 0) return FALSE;
   j = k-length;
   return TRUE;
}

/* setto(s) sets (j+1),...k to the characters in the string s, readjusting
   k. */

static void setto(char * s)
{  int length = s[0];
   memmove(b+j+1,s+1,length);
   k = j+length;
}

/* r(s) is used further down. */

static void r(char * s) { if (m() > 0) setto(s); }

/* step1ab() gets rid of plurals and -ed or -ing. e.g.

       caresses  ->  caress
       ponies    ->  poni
       ties      ->  ti
       caress    ->  caress
       cats      ->  cat

       feed      ->  feed
       agreed    ->  agree
       disabled  ->  disable

       matting   ->  mat
       mating    ->  mate
       meeting   ->  meet
       milling   ->  mill
       messing   ->  mess

       meetings  ->  meet

*/

static void step1ab()
{  if (b[k] == 's')
```

```
`
  {  if (ends("\04" "sses")) k -= 2; else
     if (ends("\03" "ies")) setto("\01" "i"); else
     if (b[k-1] != 's') k--;
  }
  if (ends("\03" "eed")) { if (m() > 0) k--; } else
  if ((ends("\02" "ed") || ends("\03" "ing")) && vowelinstem())
  {  k = j;
     if (ends("\02" "at")) setto("\03" "ate"); else
     if (ends("\02" "bl")) setto("\03" "ble"); else
     if (ends("\02" "iz")) setto("\03" "ize"); else
     if (doublec(k))
     {  k--;
        {  int ch = b[k];
           if (ch == 'l' || ch == 's' || ch == 'z') k++;
        }
     }
     else if (m() == 1 && cvc(k)) setto("\01" "e");
  }
}

/* step1c() turns terminal y to i when there is another vowel in the stem. */

static void step1c() { if (ends("\01" "y") && vowelinstem()) b[k] = 'i'; }


/* step2() maps double suffices to single ones. so -ization ( = -ize plus
   -ation) maps to -ize etc. note that the string before the suffix must give
   m() > 0. */

static void step2() { switch (b[k-1])
{
    case 'a': if (ends("\07" "ational")) { r("\03" "ate"); break; }
              if (ends("\06" "tional")) { r("\04" "tion"); break; }
              break;
    case 'c': if (ends("\04" "enci")) { r("\04" "ence"); break; }
              if (ends("\04" "anci")) { r("\04" "ance"); break; }
              break;
    case 'e': if (ends("\04" "izer")) { r("\03" "ize"); break; }
              break;
    case 'l': if (ends("\03" "bli")) { r("\03" "ble"); break; } /*-DEPARTURE-*/

 /* To match the published algorithm, replace this line with
    case 'l': if (ends("\04" "abli")) { r("\04" "able"); break; } */

              if (ends("\04" "alli")) { r("\02" "al"); break; }
              if (ends("\05" "entli")) { r("\03" "ent"); break; }
              if (ends("\03" "eli")) { r("\01" "e"); break; }
              if (ends("\05" "ousli")) { r("\03" "ous"); break; }
              break;
    case 'o': if (ends("\07" "ization")) { r("\03" "ize"); break; }
              if (ends("\05" "ation")) { r("\03" "ate"); break; }
              if (ends("\04" "ator")) { r("\03" "ate"); break; }
              break;
    case 's': if (ends("\05" "alism")) { r("\02" "al"); break; }
              if (ends("\07" "iveness")) { r("\03" "ive"); break; }
              if (ends("\07" "fulness")) { r("\03" "ful"); break; }
              if (ends("\07" "ousness")) { r("\03" "ous"); break; }
`
          break;
   case 't': if (ends("\05" "aliti")) { r("\02" "al"); break; }
          if (ends("\05" "iviti")) { r("\03" "ive"); break; }
          if (ends("\06" "biliti")) { r("\03" "ble"); break; }
          break;
   case 'g': if (ends("\04" "logi")) { r("\03" "log"); break; } /*-DEPARTURE-*/

 /* To match the published algorithm, delete this line */

} }

/* step3() deals with -ic-, -full, -ness etc. similar strategy to step2. */

static void step3() { switch (b[k])
{
   case 'e': if (ends("\05" "icate")) { r("\02" "ic"); break; }
          if (ends("\05" "ative")) { r("\00" ""); break; }
          if (ends("\05" "alize")) { r("\02" "al"); break; }
          break;
   case 'i': if (ends("\05" "iciti")) { r("\02" "ic"); break; }
          break;
   case 'l': if (ends("\04" "ical")) { r("\02" "ic"); break; }
          if (ends("\03" "ful")) { r("\00" ""); break; }
          break;
   case 's': if (ends("\04" "ness")) { r("\00" ""); break; }
          break;
} }

/* step4() takes off -ant, -ence etc., in context <c>vcvc<v>. */

static void step4()
{  switch (b[k-1])
   {  case 'a': if (ends("\02" "al")) break; return;
     case 'c': if (ends("\04" "ance")) break;
            if (ends("\04" "ence")) break; return;
     case 'e': if (ends("\02" "er")) break; return;
     case 'i': if (ends("\02" "ic")) break; return;
     case 'l': if (ends("\04" "able")) break;
            if (ends("\04" "ible")) break; return;
     case 'n': if (ends("\03" "ant")) break;
            if (ends("\05" "ement")) break;
            if (ends("\04" "ment")) break;
            if (ends("\03" "ent")) break; return;
     case 'o': if (ends("\03" "ion") && (b[j] == 's' || b[j] == 't')) break;
            if (ends("\02" "ou")) break; return;
            /* takes care of -ous */
     case 's': if (ends("\03" "ism")) break; return;
     case 't': if (ends("\03" "ate")) break;
            if (ends("\03" "iti")) break; return;
     case 'u': if (ends("\03" "ous")) break; return;
     case 'v': if (ends("\03" "ive")) break; return;
     case 'z': if (ends("\03" "ize")) break; return;
     default: return;
   }
   if (m() > 1) k = j;
}
```

`

```
/* step5() removes a final -e if m() > 1, and changes -ll to -l if
   m() > 1. */

static void step5()
{  j = k;
   if (b[k] == 'e')
   {  int a = m();
      if (a > 1 || a == 1 && !cvc(k-1)) k--;
   }
   if (b[k] == 'l' && doublec(k) && m() > 1) k--;
}

/* In stem(p,i,j), p is a char pointer, and the string to be stemmed is from
   p[i] to p[j] inclusive. Typically i is zero and j is the offset to the last
   character of a string, (p[j+1] == '\0'). The stemmer adjusts the
   characters p[i] ... p[j] and returns the new end-point of the string, k.
   Stemming never increases word length, so i <= k <= j. To turn the stemmer
   into a module, declare 'stem' as extern, and delete the remainder of this
   file.
*/

int stem(char * p, int i, int j)
{  b = p; k = j; k0 = i; /* copy the parameters into statics */
   if (k <= k0+1) return k; /*-DEPARTURE-*/

   /* With this line, strings of length 1 or 2 don't go through the
      stemming process, although no mention is made of this in the
      published algorithm. Remove the line to match the published
      algorithm. */

   step1ab(); step1c(); step2(); step3(); step4(); step5();
   return k;
}

/*--------------------stemmer definition ends here------------------------*/


static char * s;         /* a char * (=string) pointer; passed into b above */

#define INC 50           /* size units in which s is increased */
static int i_max = INC;  /* maximum offset in s */

void increase_s()
{  i_max += INC;
   {  char * new_s = (char *) malloc(i_max+1);
      { int i; for (i = 0; i < i_max; i++) new_s[i] = s[i]; } /* copy across */
      free(s); s = new_s;
   }
}

#define LETTER(ch) (isupper(ch) || islower(ch))

static void stemfile(FILE * f)
{  while(TRUE)
   {  int ch = getc(f);
      if (ch == EOF) return;
      if (LETTER(ch))
`
      {  int i = 0;
         while(TRUE)
         {  if (i == i_max) increase_s();

            ch = tolower(ch); /* forces lower case */

            s[i] = ch; i++;
            ch = getc(f);
            if (!LETTER(ch)) { ungetc(ch,f); break; }
         }
              /*          printf("\n***********************************\n");
                          printf("Word = %s, and i = %d",s,i);
                          printf("\n***********************************\n");*/
         s[stem(s,0,i-1)+1] = 0;
         /* the previous line calls the stemmer and uses its result to
            zero-terminate the string in s */
         printf("%s",s);
      }
      else putchar(ch);
   }
}

              /*          Stemmed tokens are stored seperately in char [][]
                          First Create the Array of Stemmed thoken used in DRM          */

char term_buffer[MAX_NUM_TERM][MAX_WORD_LEN];
char entity_buffer[MAX_NUM_TERM][MAX_WORD_LEN];
char temp_term_buffer[MAX_NUM_TERM][MAX_WORD_LEN];
int line_number=0, token_number=0, temp_token_number=0;
int entity_number=0, total_statement=0;

int          num_dis_rel = 0;
char    *dis_rel_list[MAX_DISC_REL];

              /*          Used to trim the char " and " from start and last of the word */
void trim_word(char **word){
              int i=0;
              ++*word;
              while(*(*word+i) != '"')
                          i++;
              *(*word+i) = '\0';
}

    /*  Fill all types of Entity   Relations     */
void       fill_all_ent_rel_type(char (*ent_rel_type)[MAX_WORD_LEN])
{
              strcpy(*(ent_rel_type+0), "S");
              strcpy(*(ent_rel_type+1), "O");
              strcpy(*(ent_rel_type+2), "X");
              strcpy(*(ent_rel_type+3), "-");
}

   /*   Count the number of Entity Present      */
void       count_num_entity(FILE *fp)
{
              char buffer[MAX_LINE_LEN];
              if ( fp != NULL )
```

```
`
        {
                while ( fgets ( buffer, sizeof(buffer), fp ) != NULL ) /* read a line */
                {
                        entity_number++;
                }
                rewind ( fp );
        }
        else
        {
                puts("Cant Open File.");
        }
}


 /*   Add new term to the Term Buffer        */
int add_term(char *buffer)
{
        int i =0;
        for (i = 0; i < temp_token_number; i++)
        {
                if(strcmp(temp_term_buffer[i], buffer) == 0)
                        return 0;
        }
        return 1;

}

        /*      Find the index of the stemmed word in Dusxourse Role Matrix*/
int     find_index(char   *buffer)
{
        int     i =0;
        for(i = 0; i< token_number; i++ )
        {
                if(strcmp(term_buffer[i],buffer) == 0)
                        return i;
        }
        return NOT_FOUND;
}

  /*   Extract number of Discourse Relations */
int     extract_num_discourse_rel(char ***DRM, int i, int j)
{
        char *result, *temp;
        int count = 0;
        if (( temp = ( char* )malloc(MAX_DRM_ENTRY_LEN )) == NULL )
        {
                printf("\nMemory Allocation failed\n");
                return ;
        }

        strcpy(temp, DRM[i][j]);

        if(strcmp(DRM[i][j], "NIL") == 0)
                return 1;
        else
        {
                result = strtok( temp, "#" );
```

```
`
                        while( result != NULL ) {
                                ++count;
                                result = strtok( NULL, "#" );
                        }
                        return count;
                }
}

   /*   All discourse Relations of Length 1*/
int             find_all_1_len_sub_seq(char ***DRM)
{
                int total_count = 0, count = 1;
                int i = 0, j = 0;

                for(i = 0; i < token_number; i++)
                {
                        for(j = 0; j < line_number; j++)
                        {
                                count = extract_num_discourse_rel(DRM, j, i);
                                total_count += count;
                        }
                }
                return total_count;
}

/*   All discourse Relations of Length 2*/
int             find_all_2_len_sub_seq(char ***DRM)
{
                int total_count = 0, prev_count = 1, cur_count = 1,count = 0;
                int i = 0, j = 0;

                for(i = 0; i < token_number; i++)
                {
                        prev_count = extract_num_discourse_rel(DRM, 0, i);
                //      total_count += prev_count;

                        for(j = 1; j < line_number; j++)
                        {
                                cur_count = extract_num_discourse_rel(DRM, j, i);
                                total_count += (prev_count * cur_count);
                                prev_count = cur_count;
                        }
//                      printf("\n\nTerm - %d :    Total len_2_sub_seq_rel ---->  %d  \n",i, total_count);
                }
                return total_count;
}

/*   All discourse Relations of Length 3*/
int             find_all_3_len_sub_seq(char ***DRM)
{
                int total_count = 0, prev_count = 1, sec_prev_count = 1, cur_count = 1,count = 0;
                int i = 0, j = 0, k = 0;

                for(i = 0; i < token_number; i++)
                {
                        sec_prev_count = extract_num_discourse_rel(DRM, 0, i);
                //      total_count += sec_prev_count;
```

`

```
                prev_count = extract_num_discourse_rel(DRM, 1, i);
        //      total_count += prev_count;

                for(k = 2; k < line_number; k++)
                {
                        cur_count = extract_num_discourse_rel(DRM, j, i);
                        total_count += (sec_prev_count * prev_count * cur_count);
                        sec_prev_count = prev_count;
                        prev_count = cur_count;
                }
                //printf("\n\nTerm - %d :    Total len_2_sub_seq_rel ---->  %d  \n",i, total_count);
        }
        return total_count;
}

/*   discourse Relations Present*/
void    add_to_rel_list(char *rel_type)
{
        if(strcmp(rel_type, "") == 0)
                return;
        char *rel;
        if (( rel = ( char* )malloc(MAX_WORD_LEN)) == NULL )
        {
                printf("\nMemory Allocation failed\n");
                return;
        }

        strcpy(rel, rel_type);

        int flag = 1, i = 0;
        *(rel+strlen(rel)-1) = '\0';
        for(i = 0; i < num_dis_rel ; i++)
        {
                if(strcmp(dis_rel_list[i], rel) == 0)
                {
                        flag = 0;
                        break;
                }
        }
        if(flag == 1)
        {
                strcpy(dis_rel_list[num_dis_rel++], rel);
        }
        //strcat(rel, "#");
}

/*   All discourse Relations Present*/
int find_rel(char *rel, char ***DRM, int i, int j)
{
        char *result, *temp;
        if (( temp = ( char* )malloc(MAX_DRM_ENTRY_LEN )) == NULL )
        {
                printf("\nMemory Allocation failed\n");
                return ;
        }
```

```
`
        strcpy(temp, DRM[i][j]);

        result = strtok( temp, "#" );
        if(result != NULL)
                if(strcmp(result, rel) == 0)
                        return TRUE;
        while( result != NULL ) {
                result = strtok( NULL, "#" );
                if(result != NULL)
                        if(strcmp(result, rel) == 0)
                                return TRUE;
        }
        free(temp);
        return FALSE;
}

/*   Find level 1 discoutse relation corresponds to level 2 discourse relation*/
void    find_level_1_dis_rel(char **drm_entry, char *dis_rel1,
                   char **prev_stmnt_nonexp_rel, char position)
{
        char *temp_rel,*temp_rel1,*temp_rel2,*dis_rel, *rel, *argument,*pos=NULL;
        int flag_right = 0;
        temp_rel = (char*) malloc(MAX_WORD_LEN);
        temp_rel1 = (char*) malloc(MAX_WORD_LEN);
        temp_rel2 = (char*) malloc(MAX_WORD_LEN);
        dis_rel = (char*) malloc(MAX_WORD_LEN);
        rel = (char*) malloc(MAX_WORD_LEN);
        argument = (char*) malloc(MAX_WORD_LEN);

        strcpy(argument, "");
        strcpy(dis_rel, dis_rel1);
        if(dis_rel == NULL)
        {
                printf("\ndis_rel is NULL\n");
                return;
        }

        while(strlen(dis_rel) != 0)
        {
                *(dis_rel+(strlen(dis_rel) - 1)) = '\0';
                pos = strrchr(dis_rel, '+');
                if(pos == NULL)
                {
                        strcpy(temp_rel, dis_rel);
                        *dis_rel = '\0';
                        temp_rel1 = strtok( temp_rel, "_" );
                }
                else
                {
                        strcpy(temp_rel, pos+1);
                        *(dis_rel+(pos-dis_rel)) = '\0';
                        strcat(dis_rel,"+");
                        temp_rel1 = strtok(temp_rel, "_" );
                }

            // checking for non explicit realtions nonexp_2_arg2_entrel
                if(strcmp(temp_rel1, "nonexp") == 0)
```

`

```
{
        temp_rel1 = strtok( NULL, "_" );    //0
        temp_rel1 = strtok( NULL, "_" );    //arg1
        temp_rel2 = strtok( NULL, "_" );
        if(temp_rel2 != NULL)
        {
                if((strcmp(temp_rel2, "synchrony") == 0) ||
            (strcmp(temp_rel2, "asynchronous") == 0))
                {
                        strcat(strcat(strcpy(rel,"temp."),temp_rel1), "#");
                        strcat(*drm_entry, rel);

                        if(strcmp(temp_rel1, "arg2") == 0)
                        {
                                //strcat(*prev_drm_entry, "temp.arg1#");
                                strcpy(*prev_stmnt_nonexp_rel, "temp.arg1#");
                                add_to_rel_list("temp.arg1#");
                        }
                }
                else if((strcmp(temp_rel2, "cause") == 0) ||
       (strcmp(temp_rel2, "condition") == 0))
                {
                        strcat(strcat(strcpy(rel, "cont."), temp_rel1), "#");
                        strcat(*drm_entry, rel);

                        if(strcmp(temp_rel1, "arg2") == 0)
                        {
                                //strcat(*prev_drm_entry, "cont.arg1#");

                        strcpy(*prev_stmnt_nonexp_rel, "cont.arg1#");
                                add_to_rel_list("cont.arg1#");
                        }
                }
                else if(strcmp(temp_rel2, "pragmatic") == 0)
                {
                        temp_rel2 = strtok( NULL, "_" );
                        if((strcmp(temp_rel2, "cause") == 0) ||
       (strcmp(temp_rel2, "condition") == 0))
                        {
                        strcat(strcat(strcpy(rel,"cont."), temp_rel1), "#");
                                strcat(*drm_entry, rel);

                                if(strcmp(temp_rel1, "arg2") == 0)
                                {

                      strcpy(*prev_stmnt_nonexp_rel, "cont.arg1#");
                                        add_to_rel_list("cont.arg1#");
                                }
                        }
                        if((strcmp(temp_rel2, "contrast") == 0) ||
       (strcmp(temp_rel2, "concession") == 0))
                        {
                      strcat(strcat(strcpy(rel,"comp."),temp_rel1), "#");
                                strcat(*drm_entry, rel);

                                if(strcmp(temp_rel1, "arg2") == 0)
                                {
```

`

```
              strcpy(*prev_stmnt_nonexp_rel, "comp.arg1#");
                                add_to_rel_list("comp.arg1#");
                        }
                }
        }
        else if((strcmp(temp_rel2, "contrast") == 0) ||
(strcmp(temp_rel2, "concession") == 0))
        {
            strcat(strcat(strcpy(rel, "comp."), temp_rel1), "#");
                strcat(*drm_entry, rel);

                if(strcmp(temp_rel1, "arg2") == 0)
                {
              strcpy(*prev_stmnt_nonexp_rel, "comp.arg1#");
                        add_to_rel_list("comp.arg1#");
                }
        }
        else if((strcmp(temp_rel2, "conjunction") == 0) ||
           (strcmp(temp_rel2, "instantiation") == 0) ||
          (strcmp(temp_rel2, "restatement") == 0) ||
                 (strcmp(temp_rel2, "alternative") == 0) ||
           (strcmp(temp_rel2, "exception") == 0) ||
            (strcmp(temp_rel2, "list") == 0))
        {
                strcat(strcat(strcpy(rel, "exp."), temp_rel1), "#");
                strcat(*drm_entry, rel);

                if(strcmp(temp_rel1, "arg2") == 0)
                {
                strcpy(*prev_stmnt_nonexp_rel, "exp.arg1#");
                        add_to_rel_list("exp.arg1#");
                }
        }
        else if(strcmp(temp_rel2, "entrel") == 0)
        {
            strcat(strcat(strcpy(rel, "entrel."), temp_rel1), "#");
                strcat(*drm_entry, rel);

                if(strcmp(temp_rel1, "arg2") == 0)
                {
                strcpy(*prev_stmnt_nonexp_rel, "entrel.arg1#");
                        add_to_rel_list("entrel.arg1#");
                }
        }
        else if(strcmp(temp_rel2, "norel") == 0)
        {
             strcat(strcat(strcpy(rel, "norel."), temp_rel1), "#");
                strcat(*drm_entry, rel);

                if(strcmp(temp_rel1, "arg2") == 0)
                {
              strcpy(*prev_stmnt_nonexp_rel, "entrel.arg1#");

                add_to_rel_list("norel.arg1#");
                }
        }
        add_to_rel_list(rel);
```

`

```
                    }
                    else //nonexp_0_arg1
                    {
                            //strcpy(argument, temp_rel1);
                            continue;
                    }
            }
            else if(strcmp(temp_rel1, "exp") == 0)
            {
                    temp_rel1 = strtok( NULL, "_" );  //2
                    temp_rel1 = strtok( NULL, "_" );  //arg2 or conn
                    temp_rel2 = strtok( NULL, "_" );  //NULL or Contrast etc.
                    if(temp_rel2 != NULL)
                    {
                            if((strcmp(temp_rel2, "synchrony") == 0) ||
                 (strcmp(temp_rel2, "asynchronous") == 0))
                            {
                                    // checking prev relation for argument
                                    if(strcmp(dis_rel, "") == 0)
                                    {
                                            if(strcmp(argument, "") == 0)
                                                    return;
                                            else
                                            {
                                  strcat(strcat(strcpy(rel, "temp."), argument), "#");
                                                    strcat(*drm_entry, rel);



                                            add_to_rel_list("temp.arg1#");

                                            break;
                                            }
                                    }
                                    *(dis_rel+(strlen(dis_rel) - 1)) = '\0';
                                    strcpy(temp_rel2, dis_rel);
                                    pos = strrchr(dis_rel, '+');
                                    if(pos != NULL)
                                    {
                                            strcpy(temp_rel, pos+1);
                                            *(dis_rel+(pos-dis_rel)) = '\0';
                                    }
                                    else
                                    {
                                            strcpy(temp_rel, temp_rel2);
                                    }

                                    strcat(dis_rel,"+");

                                    temp_rel1 = strtok( temp_rel, "_" );
                                    if(strcmp(temp_rel1, "attr") == 0)
                                    {
                                            strcpy(temp_rel1, argument);

                                    }
                                    else
                                    {
```

```
`
                temp_rel1 = strtok( NULL, "_" );
                temp_rel1 = strtok( NULL, "_" );
                temp_rel2 = strtok( NULL, "_" );
                if(temp_rel2 == NULL);
                else
                {
                        strcpy(temp_rel1, argument);

                }
        }

        strcpy(temp_rel1, argument);
        if(flag_right)
        {
                strcpy(argument, "arg2");
                flag_right = 0;
        }
     strcat(strcat(strcpy(rel, "temp."), temp_rel1), "#");
        strcat(*drm_entry, rel);


        if(strcmp(temp_rel1, "arg2") == 0)
        {
                add_to_rel_list("temp.arg1#");
        }
}
else if((strcmp(temp_rel2, "cause") == 0) ||
 (strcmp(temp_rel2, "condition") == 0))
{
        if(strcmp(dis_rel, "") == 0)
        {
                if(strcmp(argument, "") == 0)
                        return;
                else
                {
        strcat(strcat(strcpy(rel, "cont."), argument), "#");
                        strcat(*drm_entry, rel);



                        add_to_rel_list("cont.arg1#");

                        break;
                }
        }
        *(dis_rel+(strlen(dis_rel) - 1)) = '\0';
        strcpy(temp_rel2, dis_rel);
        pos = strrchr(dis_rel, '+');
        if(pos != NULL)
        {
                strcpy(temp_rel, pos+1);
                *(dis_rel+(pos-dis_rel)) = '\0';
        }
        else
        {
                strcpy(temp_rel, temp_rel2);
        }
```

`

```
    strcat(dis_rel,"+");


    temp_rel1 = strtok( temp_rel, "_" );
    if(strcmp(temp_rel1, "attr") == 0)
    {
        strcpy(temp_rel1, argument);

    }
    else
    {
        temp_rel1 = strtok( NULL, "_" );
        temp_rel1 = strtok( NULL, "_" );
        temp_rel2 = strtok( NULL, "_" );
        if(temp_rel2 == NULL);
        else
        {
            strcpy(temp_rel1, argument);

        }
    }

    strcpy(temp_rel1, argument);
    if(flag_right)
    {
        strcpy(argument, "arg2");
        flag_right = 0;
    }
   strcat(strcat(strcpy(rel, "cont."), temp_rel1), "#");
    strcat(*drm_entry, rel);

    //if(strcmp(temp_rel1, "arg2") == 0)
    add_to_rel_list("cont.arg1#");

}
else if(strcmp(temp_rel2, "pragmatic") == 0)
{
    temp_rel2 = strtok( NULL, "_" );
    if((strcmp(temp_rel2, "cause") == 0) ||
   (strcmp(temp_rel2, "condition") == 0))
    {
        if(strcmp(dis_rel, "") == 0)
        {
            if(strcmp(argument, "") == 0)
                return;
            else
            {
   strcat(strcat(strcpy(rel, "cont."), argument), "#");
                strcat(*drm_entry, rel);


            add_to_rel_list("cont.arg1#");

                break;
            }
        }
        *(dis_rel+(strlen(dis_rel) - 1)) = '\0';
```

```
            strcpy(temp_rel2, dis_rel);
            pos = strrchr(dis_rel, '+');
            if(pos != NULL)
            {
                    strcpy(temp_rel, pos+1);
                    *(dis_rel+(pos-dis_rel)) = '\0';
            }
            else
            {
                    strcpy(temp_rel, temp_rel2);
            }

            strcat(dis_rel,"+");

            temp_rel1 = strtok( temp_rel, "_" );
            if(strcmp(temp_rel1, "attr") == 0)
            {
                    strcpy(temp_rel1, argument);

            }
            else
            {
                    temp_rel1 = strtok( NULL, "_" );
                    temp_rel1 = strtok( NULL, "_" );
                    temp_rel2 = strtok( NULL, "_" );
                    if(temp_rel2 == NULL);
                    else
                    {
                        strcpy(temp_rel1, argument);

                    }
            }

            strcpy(temp_rel1, argument);

            if(flag_right)
            {
                    strcpy(argument, "arg2");
                    flag_right = 0;
            }
 strcat(strcat(strcpy(rel, "cont."), temp_rel1), "#");
            strcat(*drm_entry, rel);



            add_to_rel_list("cont.arg1#");

    }
    if((strcmp(temp_rel2, "contrast") == 0) ||
(strcmp(temp_rel2, "concession") == 0))
    {
            if(strcmp(dis_rel, "") == 0)
            {
                    if(strcmp(argument, "") == 0)
                            return;
                    else
                    {
```

```
`
strcat(strcat(strcpy(rel, "comp."), argument), "#");
                    strcat(*drm_entry, rel);



          add_to_rel_list("comp.arg1#");
                break;
              }
        }
        *(dis_rel+(strlen(dis_rel) - 1)) = '\0';
        strcpy(temp_rel2, dis_rel);

        pos = strrchr(dis_rel, '+');
        if(pos != NULL)
        {
              strcpy(temp_rel, pos+1);
              *(dis_rel+(pos-dis_rel)) = '\0';
        }
        else
        {
              strcpy(temp_rel, temp_rel2);
        }

        strcat(dis_rel,"+");

        temp_rel1 = strtok( temp_rel, "_" );
        if(strcmp(temp_rel1, "attr") == 0)
        {
              strcpy(temp_rel1, argument);

        }
        else
        {
              temp_rel1 = strtok( NULL, "_" );
              temp_rel1 = strtok( NULL, "_" );
              temp_rel2 = strtok( NULL, "_" );
              if(temp_rel2 == NULL);
              else
              {
              strcpy(temp_rel1, argument);

              }
        }

        strcpy(temp_rel1, argument);

        if(flag_right)
        {
              strcpy(argument, "arg2");
              flag_right = 0;
        }
strcat(strcat(strcpy(rel, "comp."), temp_rel1), "#");
        strcat(*drm_entry, rel);



        add_to_rel_list("comp.arg1#");
```

```
            }
}
else if((strcmp(temp_rel2, "contrast") == 0) ||
   (strcmp(temp_rel2, "concession") == 0))
{
            if(strcmp(dis_rel, "") == 0)
            {
                        if(strcmp(argument, "") == 0)
                                    return;
                        else
                        {
         strcat(strcat(strcpy(rel, "comp."), argument), "#");
                                    strcat(*drm_entry, rel);



                                    add_to_rel_list("comp.arg1#");

                                    break;
                        }
            }

            *(dis_rel+(strlen(dis_rel) - 1)) = '\0';
            strcpy(temp_rel2, dis_rel);
            pos = strrchr(dis_rel, '+');
            if(pos != NULL)
            {
                        strcpy(temp_rel, pos+1);
                        *(dis_rel+(pos-dis_rel)) = '\0';
            }
            else
            {
                        strcpy(temp_rel, temp_rel2);
            }
            strcat(dis_rel,"+");

            temp_rel1 = strtok( temp_rel, "_" );
            if(strcmp(temp_rel1, "attr") == 0)
            {
                        strcpy(temp_rel1, argument);

            }
            else
            {
                        temp_rel1 = strtok( NULL, "_" );
                        temp_rel1 = strtok( NULL, "_" );
                        temp_rel2 = strtok( NULL, "_" );
                        if(temp_rel2 == NULL);
                        else
                        {
                                    strcpy(temp_rel1, argument);

                        }
            }
            strcpy(temp_rel1, argument);
```

`

```
        if(flag_right)
        {
                strcpy(argument, "arg2");
                flag_right = 0;
        }
     strcat(strcat(strcpy(rel, "comp."), temp_rel1), "#");
        strcat(*drm_entry, rel);


        add_to_rel_list("comp.arg1#");

}
else if((strcmp(temp_rel2, "conjunction") == 0) ||
 (strcmp(temp_rel2, "instantiation") == 0) ||
(strcmp(temp_rel2, "restatement") == 0) ||
(strcmp(temp_rel2, "alternative") == 0) ||
 (strcmp(temp_rel2, "exception") == 0) ||
(strcmp(temp_rel2, "list") == 0))
{
        if(strcmp(dis_rel, "") == 0)
        {
                if(strcmp(argument, "") == 0)
                        return;
                else
                {
         strcat(strcat(strcpy(rel, "exp."), argument), "#");
                        strcat(*drm_entry, rel);



                        add_to_rel_list("exp.arg1#");

                        break;
                }
        }
        *(dis_rel+(strlen(dis_rel) - 1)) = '\0';
        strcpy(temp_rel2, dis_rel);
        pos = strrchr(dis_rel, '+');
        if(pos != NULL)
        {
                strcpy(temp_rel, pos+1);
                *(dis_rel+(pos-dis_rel)) = '\0';
        }
        else
        {
                strcpy(temp_rel, temp_rel2);
        }

        strcat(dis_rel,"+");

        temp_rel1 = strtok( temp_rel, "_" );
        if(strcmp(temp_rel1, "attr") == 0)
        {
                strcpy(temp_rel1, argument);

        }
        else
```

`

```
        {
                temp_rel1 = strtok( NULL, "_" );
                temp_rel1 = strtok( NULL, "_" );
                temp_rel2 = strtok( NULL, "_" );
                if(temp_rel2 == NULL);
                else
                {
                        strcpy(temp_rel1, argument);

                }
        }

        strcpy(temp_rel1, argument);
        if(flag_right)
        {
                strcpy(argument, "arg2");
                flag_right = 0;
        }
        strcat(strcat(strcpy(rel, "exp."), temp_rel1), "#");
        strcat(*drm_entry, rel);

      add_to_rel_list("exp.arg1#");

}
else if(strcmp(temp_rel2, "entrel") == 0)
{
        if(strcmp(dis_rel, "") == 0)
        {
                if(strcmp(argument, "") == 0)
                        return;
                else
                {
      strcat(strcat(strcpy(rel, "entrel."), argument), "#");
                        strcat(*drm_entry, rel);


                        add_to_rel_list("entrel.arg1#");

                        break;
                }
        }
        *(dis_rel+(strlen(dis_rel) - 1)) = '\0';
        strcpy(temp_rel2, dis_rel);
        pos = strrchr(dis_rel, '+');
        if(pos != NULL)
        {
                strcpy(temp_rel, pos+1);
                *(dis_rel+(pos-dis_rel)) = '\0';
        }
        else
        {
                strcpy(temp_rel, temp_rel2);
        }
        strcat(dis_rel,"+");

        temp_rel1 = strtok( temp_rel, "_" );
        if(strcmp(temp_rel1, "attr") == 0)
```

```
`
                    {
                        strcpy(temp_rel1, argument);

                    }
                    else
                    {
                        temp_rel1 = strtok( NULL, "_" );
                        temp_rel1 = strtok( NULL, "_" );
                        temp_rel2 = strtok( NULL, "_" );
                        if(temp_rel2 == NULL);
                        else
                        {
                            strcpy(temp_rel1, argument);

                        }
                    }

                    strcpy(temp_rel1, argument);
                    if(flag_right)
                    {
                        strcpy(argument, "arg2");
                        flag_right = 0;
                    }
                  strcat(strcat(strcpy(rel, "entrel."), temp_rel1), "#");
                    strcat(*drm_entry, rel);
                add_to_rel_list("entrel.arg1#");

                }
                add_to_rel_list(rel);
            }
            else
            {
                if(position == 'l')
                    strcpy(argument, "arg1");
                else if(position == 'r')
                    strcpy(argument, "arg2");
                else if(position == 'a')
                {
                    strcpy(argument, "arg1");
                    flag_right = 1;
                }
                continue;
            }
        }
        else if (strcmp(temp_rel1, "attr") == 0)
        {
            continue;
        }
    }
}

/*  Remove duplicate relations from the relation Buffer   */
void    remove_duplicate_rel(char   *drm_rel, char **drm_entry)
{
    if(*drm_entry == NULL)
    {
        *drm_entry = drm_rel;
```

```
`
                return;
        }

        int duplicate_flag = 0;
        char *temp_rel1,  *temp_rel2, *temp_rel3, *pos = NULL;
        temp_rel1 = (char*) malloc(MAX_WORD_LEN);
        temp_rel2 = (char*) malloc(MAX_WORD_LEN);
        temp_rel3 = (char*) malloc(MAX_WORD_LEN);

        strcpy(temp_rel1, *drm_entry);
        strcpy(temp_rel3, drm_rel);

        temp_rel1 = strtok( temp_rel1, "#" );
        while (temp_rel1 != NULL)
        {
                duplicate_flag = 0;
                strcpy(temp_rel2, drm_rel);
                temp_rel2 = strtok( temp_rel2, "#" );
                while (temp_rel2 != NULL)
                {
                        if(strcmp(temp_rel1, temp_rel2) == 0)
                        {
                                duplicate_flag = 1;
                        }
                        temp_rel2 = strtok( NULL, "#" );
                }
                if(duplicate_flag == 0)
                {
                        strcat(temp_rel3, temp_rel1);
                }
                temp_rel1 = strtok( NULL, "#" );
        }
        if(*(temp_rel3+strlen(temp_rel3)-1)!= '#')
                strcat(temp_rel3, "#");
        strcpy(*drm_entry, temp_rel3);
}

  /*   Fill all possible level 1 relation to the relation buffer     */
void    fill_all_rel_type(char (*dis_rel_type)[MAX_WORD_LEN])
{
        strcpy(*(dis_rel_type+0), "NIL");
        strcpy(*(dis_rel_type+1), "temp.arg1");
        strcpy(*(dis_rel_type+2), "temp.arg2");
        strcpy(*(dis_rel_type+3), "cont.arg1");
        strcpy(*(dis_rel_type+4), "cont.arg2");
        strcpy(*(dis_rel_type+5), "comp.arg1");
        strcpy(*(dis_rel_type+6), "comp.arg2");
        strcpy(*(dis_rel_type+7), "exp.arg1");
        strcpy(*(dis_rel_type+8), "exp.arg2");
        strcpy(*(dis_rel_type+9), "entrel.arg1");
        strcpy(*(dis_rel_type+10), "entrel.arg2");
        strcpy(*(dis_rel_type+11), "norel.arg1");
        strcpy(*(dis_rel_type+12), "norel.arg2");
}

  /*   Matching with possible length 1 relation to relation buffer*/
int     find_1_len_rel(char  (*dis_rel_all)[MAX_WORD_LEN], char *dis_rel)
```

```
`
{
        int i = 0;
        for(i=0; i< MAX_DISC_REL; i++)
        {
                if(strcmp(*(dis_rel_all+i), dis_rel) == 0)
                        return i;
        }
        return -1;
}

  /*  Matching with possible length 2 relation to relation buffer*/
int     find_2_len_rel(char **dis_rel_all, char *dis_rel)
{
        int i = 0;
        for(i=0; i< MAX_DISC_REL * MAX_DISC_REL; i++)
        {
                if(strcmp(*(dis_rel_all+i), dis_rel) == 0)
                        return i;
        }
        return -1;
}

  /*  Matching with possible length 3 relation to relation buffer*/
int     find_3_len_rel(char **dis_rel_all, char *dis_rel)
{
        int i = 0;
        for(i=0; i< MAX_DISC_REL * MAX_DISC_REL * MAX_DISC_REL; i++)
        {
                if(strcmp(*(dis_rel_all+i), dis_rel) == 0)
                        return i;
        }
        return -1;
}

  /*   Convert  string to integer    */
void   cvtInt( char **str, int num)
{
        sprintf( *str, "%d", num );
}

  /*  Count Number of statement    */
void    count_num_statement(FILE *fp)
{
        char buffer[MAX_LINE_LEN];
        if ( fp != NULL )
        {
                fgets ( buffer, sizeof(buffer), fp );
                FILE *file = fopen("tmp", "w");     //Stemmed version of Discoure Parsed File
                if( file == NULL )
                {
                        puts("Can't open file:    tmp");
                        return ;
                }
                fprintf (file, "%s", buffer); //writting lines to temp file

                fclose(file);
                file =  fopen("tmp", "r");
```

`

```
			while (!feof(file))
			{
				fflush(stdin);
				fscanf(file, "%s", buffer);
				total_statement++;
			}
			total_statement--;
			rewind(fp);

			fclose(file);
			rewind ( fp );
		}
		else
		{
			puts("Cant Open File.");
		}
}


/*    Fill Existing Entity Relationship   */
void    fill_entity_rel(FILE *fp, char ***ERM)
{
		int    i = 0, j = 0;
		char buffer[MAX_WORD_LEN];

		for(i = 0; i < entity_number; i++)
		{
			fflush(stdin);
			fscanf(fp, "%s", buffer);
			for(j = 1; j < total_statement; j++)
			{
				fflush(stdin);
				fscanf(fp, "%s", buffer);
				strcpy(ERM[i][j], buffer);
			}
		}
}


    /*   Fill Length 1 Entity Relationship   */
void	count_len_1_ent_seq(char ***ERM, int *count_1_len_ent_rel)
{
		int    i = 0, j = 0;
		char buffer[MAX_WORD_LEN];

		for(i = 0; i < entity_number; i++)
		{
			for(j = 1; j < total_statement; j++)
			{
				if(strcmp(ERM[i][j], "S") == 0)
				{
					count_1_len_ent_rel[0]++;
				}
				else if(strcmp(ERM[i][j], "O") == 0)
				{
					count_1_len_ent_rel[1]++;
`
                }
                else if(strcmp(ERM[i][j], "X") == 0)
                {
                    count_1_len_ent_rel[2]++;
                }
                else if(strcmp(ERM[i][j], "-") == 0)
                {
                    count_1_len_ent_rel[3]++;
                }
                else
                {
                    printf("\nUnknown etity type.");
                    return;
                }
            }
        }
}

 /*   Fill Length 2 Entity Relationship   */
void    count_len_2_ent_seq(char ***ERM, int *count_2_len_ent_rel,
                       char **ent_rel_2_len_all)
{
        int i=0;
        char *temp, *temp1, *temp2;
        temp = (char*) malloc(MAX_WORD_LEN);
        temp1 = (char*) malloc(MAX_ERM_ENTRY);
        temp2 = (char*) malloc(MAX_ERM_ENTRY);

        for(i = 0; i < MAX_ENT_REL*MAX_ENT_REL; i++)
        {
            for(j = 0; j < entity_number; j++)
            {
                for(k = 1; k < total_statement-1; k++)
                {
                    strcpy(temp1, ERM[j][k]);
                    strcat(temp1, "#");
                    strcpy(temp2, ERM[j][k+1]);
                    strcat(temp2, "#");
                    strcpy(temp, temp1);
                    strcat(temp, temp2);
                    if(strcmp(temp, ent_rel_2_len_all[i]) == 0)
                    {
                        count_2_len_ent_rel[i]++;
                    }
                }
            }
        }

}

 /*   Fill Length 3 Entity Relationship   */
void    count_len_3_ent_seq(char ***ERM, int *count_3_len_ent_rel,
                       char **ent_rel_3_len_all)
{
        int i=0;
        char *temp, *temp1, *temp2, *temp3;
        temp = (char*) malloc(MAX_WORD_LEN);
```

```
`
        temp1 = (char*) malloc(MAX_ERM_ENTRY);
        temp2 = (char*) malloc(MAX_ERM_ENTRY);
        temp3 = (char*) malloc(MAX_ERM_ENTRY);

        for(i = 0; i < MAX_ENT_REL*MAX_ENT_REL*MAX_ENT_REL; i++)
        {
                for(j = 0; j < entity_number; j++)
                {
                        for(k = 1; k < total_statement-2; k++)
                        {
                                strcpy(temp1, ERM[j][k]);
                                strcat(temp1, "#");
                                strcpy(temp2, ERM[j][k+1]);
                                strcat(temp2, "#");
                                strcpy(temp3, ERM[j][k+2]);
                                strcat(temp3, "#");
                                strcpy(temp, temp1);
                                strcat(temp, temp2);
                                strcat(temp, temp3);
                                if(strcmp(temp, ent_rel_3_len_all[i]) == 0)
                                {
                                        count_3_len_ent_rel[i]++;
                                }
                        }
                }
        }

}



    /*   Execution Starts Here-   main()  */
int main(int argc, char *argv[])
{
        /* We need two input files
          1. Parsed File with Tags of original Text
          2. Discourse Parsed File*/

        int count_braces=0, i=0, j=0, k=0, l=0;

        if ( argc != 2 ) /* argc should be 2 for correct execution */
        {
                printf( "\nInvalid number of arguments:  usage: %s filename\n",argv[0]);
                return 1;
        }

        char file_name[MAX_FILE_NAME_LEN];
        strcpy(file_name, "./data1-test/");
        strcat(file_name, argv[1]);
        char filename_parse[MAX_FILE_NAME_LEN];
        strcpy(filename_parse, file_name);
        strcat(filename_parse, "-p.parsed");
        char filename_grid[MAX_FILE_NAME_LEN];
        strcpy(filename_grid, filename_parse);
        strcat(filename_grid, ".grid");
        char filename_coref[MAX_FILE_NAME_LEN];
        strcpy(filename_coref, "./data1-test-coref/");
```

```
`
            strcat(filename_coref, argv[1]);
            strcat(filename_coref, ".coref");

            FILE *fp, *fp1, *fp2, *fp3;

            char *buffer;
            buffer = (char*) malloc(MAX_WORD_LEN);

            fp1 = fopen(filename_parse, "r");
            if( fp1 == NULL )
            {
                        printf("\nCan't open file:  %s\n",filename_parse);
                        return ;
            }
            fflush(stdin);

/*          Extracting all the Verb, Noun, Adjective and Adverb from the Parsed File*/
/*          Add these extracted stemmed terms to term_buffer list              */
            while (!feof(fp1)){
                        fflush(stdin);
                        fscanf(fp1, "%s", buffer);
                        if(strcmp(buffer,"NN")==0 || strcmp(buffer,"NNS")==0 ||
                       strcmp(buffer,"NNP")==0 || strcmp(buffer,"NNPS")==0 ||
                       strcmp(buffer,"JJ")==0 || strcmp(buffer,"JJR")==0 ||
                       strcmp(buffer,"JJS")==0 || strcmp(buffer,"RB")==0 ||
                         strcmp(buffer,"RBR")==0 || strcmp(buffer,"RBS")==0 ||
                       strcmp(buffer,"VB")==0 || strcmp(buffer,"VBD")==0 ||
                           strcmp(buffer,"VBG")==0 || strcmp(buffer,"VBN")==0 ||
                       strcmp(buffer,"VBP")==0 || strcmp(buffer,"VBZ")==0 )
                        {
                                    fflush(stdin);
                                    fscanf(fp1, "%s", buffer);
                                    trim_word(&buffer);
                                    for (i = 0; buffer[i]; i++)
                                                buffer[i] = tolower(buffer[ i ]);
                                    buffer[ stem(buffer,0,strlen(buffer)-1)+1] = 0;
          // finding stemmed word corresponding to the term
                                    if(add_term(buffer) == TRUE)
                                    {
                                                strcpy(temp_term_buffer[temp_token_number],buffer);
                                                temp_token_number++;
                                    }
                                    /*if(add_term(buffer) == TRUE)
                                    {
                                                strcpy(term_buffer[token_number],buffer);
                                                token_number++;
                                    }*/
                        }
                        fflush(stdin);
            }

            rewind(fp1);
            fclose(fp1);

            /*          Reading Discoure Parsed File          */

            char file_name1[MAX_FILE_NAME_LEN], *pos, ch;
```

`

```
strcpy(file_name1, "./discourse_relations/test/");
strcat(file_name1, argv[1]);
strcat(file_name1, "-disc-rel");

fp = fopen(file_name1, "r");
if( fp == NULL )
{
        printf("Can't open file:   %s", file_name1);
        return ;
}

fp1 = fopen("temp1", "w");          //Stemmed version of Discoure Parsed File
if( fp1 == NULL )
{
        printf("Can't open file:   temp1");
        return ;
}


while (!feof(fp)){
        fflush(stdin);
        fscanf(fp, "%s", buffer);
        for (i = 0; buffer[i]; i++)
                buffer[i] = tolower(buffer[ i ]);
        pos = strrchr(buffer, '_');
        if((pos != NULL) && ((strcmp(pos+1, "synchrony") == 0) ||
    (strcmp(pos+1, "asynchronous") == 0) || (strcmp(pos+1, "cause") == 0) ||
    (strcmp(pos+1, "condition") == 0) || (strcmp(pos+1, "contrast") == 0)||
    (strcmp(pos+1, "concession")==0)||(strcmp(pos+1, "conjunction")==0)||
    (strcmp(pos+1, "instantiation")==0)||(strcmp(pos+1, "restatement")==0)||
     (strcmp(pos+1, "alternative") == 0)||(strcmp(pos+1, "exception") == 0) ||
     (strcmp(pos+1, "list") == 0)));
        else
                buffer[ stem(buffer,0,strlen(buffer)-1)+1] = 0;
        fprintf (fp1, "%s", " "); //writting lines to temp file
        fprintf (fp1, "%s", buffer); //writting lines to temp file
        fprintf (fp1, "%s", " "); //writting lines to temp file
}
fseek(fp1,-strlen(buffer)-2,SEEK_CUR);
fprintf (fp1, "%s", "                    ");

fclose(fp1);
fclose(fp);


/* Start of the filling Discourse Relations*/
fp = fopen("temp1", "r");   //Stemmed version of Discoure Parsed File
if( fp == NULL )
{
        puts("Can't open file:    temp1");
        return ;
}
        /*Start Extracting Discourse Relations from Discourse Parsed File*/

int *temp_token_count;
temp_token_count = (int*) malloc(sizeof(int) * temp_token_number);
```

```
`
    for(k=0; k< temp_token_number; k++)
    {
        temp_token_count[k] = 0;
    }

    for(k=0; k< temp_token_number; k++)
    {
        while (!feof(fp))
        {
            fflush(stdin);
            fscanf(fp, "%s", buffer);
            if(strcmp(buffer, temp_term_buffer[k]) == 0)
            {
                ++temp_token_count[k];
            }
        }
        rewind(fp);
    }

    /*for(k=0; k< temp_token_number; k++)
    {
printf("\nCount for token %s is %d", temp_term_buffer[k], temp_token_count[k]);
    }*/

    fclose(fp);

    int *token_count;
    token_count = (int*) malloc(sizeof(int) * temp_token_number);

    for(k = 0; k < temp_token_number; k++)
    {
        if(temp_token_count[k] >= MATRIX_SIL_VAL)
        {
            strcpy(term_buffer[token_number],temp_term_buffer[k]);
            token_count[token_number++] = temp_token_count[k];
        }
    }

    /*printf("\n\nToken Number: %d\n", token_number);
    for(k=0; k< token_number; k++)
    {
printf("\nNew Count %d for token %s is %d", k, term_buffer[k], token_count[k]);
    }*/

    fp = fopen(file_name1, "r");
    if( fp == NULL )
    {
        printf("Can't open file:   %s", file_name1);
        return ;
    }

    do
    {
        ch = (char)fgetc(fp);
        if(ch == '{')
            count_braces++;
        else if(ch == '}')
```

`

```
        {
                count_braces--;
                if(count_braces == 0)
                        ++line_number;
        }
} while(ch != EOF);

//printf("\nNumber of Statements: %d \n", line_number);

rewind(fp);

/*  Start of Extracting LINES from the Parsed File  */

//printf("\nNumber of statements in Article:   %d", line_number);
//printf("\nNumber of Open Words (Terms) in Article:   %d", line_number);


char ***DRM;                    //      line_number = Number of sentences
if (( DRM = ( char*** )malloc( (line_number+1)*sizeof( char** ))) == NULL )
{
        printf("\nMemory Allocation failed\n");
        return ;
}
//      token_number = Number of Stemmed token in DRM

for(i = 0; i< line_number+1; i++)
{
if (( DRM[i] = ( char** )malloc( (token_number+1)*sizeof( char* ))) ==          NULL)
        {
                printf("\nMemory Allocation failed\n");
                return ;
        }
}

for(i = 0; i< line_number+1; i++)
{
        for ( j = 0; j < token_number+1; j++ )
        {
                if (( DRM[i][j] = ( char* )malloc(
                                MAX_DRM_ENTRY_LEN )) == NULL )
                {
                        printf("\nMemory Allocation failed\n");
                        return ;
                }
        }
        //Initialization Here
        for(k=0; k< token_number + 1; k++)
        {
                strcpy(DRM[i][k],"NIL");
        }
}


/* Start of the filling Discourse Relations*/
fp = fopen("temp1", "r");   //Stemmed version of Discoure Parsed File
if( fp == NULL )
{
```

`

```
        puts("Can't open file:    temp1");
        return ;
}
        /*Start Extracting Discourse Relations from Discourse Parsed File*/

for ( j = 0; j < MAX_DISC_REL; j++ )
{
        if (( dis_rel_list[j] = ( char* )malloc(MAX_DRM_ENTRY_LEN ))
                                                                == NULL)
        {
                printf("\nMemory Allocation failed\n");
                return ;
        }
}

strcpy(dis_rel_list[num_dis_rel++], "NIL");

int paranthesis_count = 0, index_word = -1,statement_num =0, term_num = 0;
char *dis_rel, *drm_entry, *prev_drm_entry, *prev_stmnt_nonexp_rel;
int *prev_statement_num;
prev_statement_num = (int *)malloc(sizeof(int) * (token_number+1));
dis_rel = (char*) malloc(MAX_WORD_LEN);
drm_entry = (char*) malloc(MAX_WORD_LEN);
prev_drm_entry = (char*) malloc(MAX_WORD_LEN);
prev_stmnt_nonexp_rel = (char*) malloc(MAX_WORD_LEN);

for(i = 0; i<= token_number; i++)
{
        prev_statement_num[i] = -1;
}

rewind(fp);

FILE *fp4;
char position;
int exp_flag = 0, flag = 0;

while (!feof(fp))
{
        fflush(stdin);
        fp4 = fopen("temp2", "w");
        if( fp4 == NULL )
        {
                printf("Can't open file:   temp2");
                return ;
        }

        while (!feof(fp))
        {
                fscanf(fp, "%s", buffer);
                fprintf (fp4, "%s", buffer);
                fprintf (fp4, "%s", " ");

                if(*buffer == '{')
                {
                        strcat(dis_rel, ++buffer);
                        strcat(dis_rel, "+");
```

```
`
            paranthesis_count++;
            continue;
        }
        else if(*(buffer+strlen(buffer)-1) == '}')
        {
            paranthesis_count--;
            if(paranthesis_count == 0)
            {
                break;
            }
        }
    }
    fclose(fp4);

    fp4 = fopen("temp2", "r");
    if( fp4 == NULL )
    {
        printf("Can't open file:   temp2");
        return ;
    }

    while (!feof(fp4))
    {
        fscanf(fp4, "%s", buffer);

        if(exp_flag && (strncmp(buffer, "{exp_", 5) == 0))
        {
            flag = 1;
        }
        else if(flag && (strncmp(buffer, "{exp_", 5) == 0)&&
(strncmp(buffer+strlen(buffer) - 4, "arg2", 4) == 0))
        {
            position = 'a';
            flag = 0;
            continue;
        }
        else if((strncmp(buffer, "{exp_", 5) == 0) &&
(strncmp(buffer+strlen(buffer) - 4, "arg1", 4) == 0) )
        {
            position = 'l';
            exp_flag = 1;
            continue;
        }
        else if((strncmp(buffer, "{exp_", 5) == 0) &&
(strncmp(buffer+strlen(buffer) - 4, "arg2", 4) == 0) )
        {
            position = 'r';
            continue;
        }

        exp_flag = 0;

        if((term_num = find_index(buffer)) == NOT_FOUND)
        {
            continue;
        }
        else
```

```
`
            {
                memset(drm_entry,'\0',sizeof(drm_entry));
                memset(prev_drm_entry,'\0',sizeof(prev_drm_entry));

        memset(prev_stmnt_nonexp_rel,'\0',sizeof(prev_stmnt_nonexp_rel));

            if(strcmp(DRM[statement_num][term_num], "NIL") == 0)
                {

                    find_level_1_dis_rel(&drm_entry, dis_rel,
            &prev_stmnt_nonexp_rel, position);
                    if(strcmp(drm_entry, "") != 0)

    strcpy(DRM[statement_num][term_num],drm_entry);

                    if(strlen(prev_stmnt_nonexp_rel) != 0)
                    {
                        if(statement_num > 0)
                        {

       if(strcmp(DRM[statement_num-1][term_num], "NIL") == 0)
         strcpy(DRM[statement_num-1][term_num],
                prev_stmnt_nonexp_rel);
                            else
                            {

    remove_duplicate_rel(DRM[statement_num-1]
            [term_num], &prev_stmnt_nonexp_rel);
                                strcpy(DRM[statement_num-1]
            [term_num],prev_stmnt_nonexp_rel);
                            }
                        }
                    }

                }
                else
 // Need to resolve if same term appears more than once in a sentence
                {
                    find_level_1_dis_rel(&drm_entry, dis_rel,
           &prev_stmnt_nonexp_rel, position);

                    if(strcmp(DRM[statement_num][term_num],drm_entry)
== 0)

                    {
                        if(statement_num > 0)
                        {
                        if(strcmp(DRM[statement_num-1]    [term_num],
                        "NIL") == 0)
                strcpy(DRM[statement_num-1][term_num],
                    prev_stmnt_nonexp_rel);
                        else
                        {
                         remove_duplicate_rel(DRM[statement_num-
1][term_num],
               &prev_stmnt_nonexp_rel);
                  strcpy(DRM[statement_num-1][term_num],
               prev_stmnt_nonexp_rel);
```

`

```
                                                  }
                                                  }
                                          }
                                          else
                                          {

          remove_duplicate_rel(DRM[statement_num][term_num],
                                     &drm_entry);

          strcpy(DRM[statement_num][term_num],drm_entry);
                                                  if(statement_num > 0)
                                                  {
                                                               if(strcmp(DRM[statement_num-
          1][term_num],"NIL") ==0)
                                                                    strcpy(DRM[statement_num-
1][term_num],
                              prev_stmnt_nonexp_rel);
                                                          else
                                                          {

          remove_duplicate_rel(DRM[statement_num-1]
                         [term_num], &prev_stmnt_nonexp_rel);
                                                                    strcpy(DRM[statement_num-
1][term_num],
                              prev_stmnt_nonexp_rel);
                                                          }
                                                  }

                                          }
                              prev_statement_num[term_num] = statement_num;
                              }
                    }
          }
          statement_num++;          //need to handle
          memset(dis_rel,'\0',sizeof(dis_rel)); //need to handle
          fclose(fp4);
     }

     fclose(fp);

          //        Deriving all possible 2 len sub sequences

     char **dis_rel_2_len;
     char **dis_rel_3_len;
     char **dis_rel_2_len_all;
     char **dis_rel_3_len_all;
     int        index_2_len_rel = 0;
     int        index_3_len_rel = 0;

     if (( dis_rel_2_len = ( char** )malloc( MAX_DISC_REL * MAX_DISC_REL *  sizeof( char*
     ))) == NULL )
     {
          printf("\nMemory Allocation failed\n");
          return ;
     }
     if (( dis_rel_2_len_all = ( char** )malloc( MAX_DISC_REL * MAX_DISC_REL *   sizeof(
char* ))) == NULL )
```

```
`
    {
        printf("\nMemory Allocation failed\n");
        return ;
    }
    if (( dis_rel_3_len_all = ( char** )malloc( MAX_DISC_REL * MAX_DISC_REL *
MAX_DISC_REL * sizeof( char* ))) == NULL )
    {
        printf("\nMemory Allocation failed\n");
        return ;
    }
    if (( dis_rel_3_len = ( char** )malloc( MAX_DISC_REL * MAX_DISC_REL *
MAX_DISC_REL * sizeof( char* ))) == NULL )
    {
        printf("\nMemory Allocation failed\n");
        return ;
    }
    for ( j = 0; j < MAX_DISC_REL * MAX_DISC_REL; j++ )
    {
        if ((( dis_rel_2_len[j] = ( char* )malloc(MAX_DRM_ENTRY_LEN )) == NULL)
   ||((dis_rel_2_len_all[j]=(char* malloc(MAX_DRM_ENTRY_LEN))==NULL))
        {
            printf("\nMemory Allocation failed\n");
            return ;
        }
    }
    for ( j = 0; j < MAX_DISC_REL * MAX_DISC_REL * MAX_DISC_REL; j++ )
    {
        if ( ((dis_rel_3_len_all[j] = (char*)malloc(MAX_DRM_ENTRY_LEN))==NULL)||
   ((dis_rel_3_len[j] = ( char* )malloc(MAX_DRM_ENTRY_LEN )) == NULL) )
        {
            printf("\nMemory Allocation failed\n");
            return ;
        }
    }

    /*      Printing DRM     */
    /*for(i = 0; i< line_number; i++)
    {
        printf("\n\n\n Sentence %d:                    ",i);
        for(k=0; k< token_number; k++)
        {
            printf("%s\t",DRM[i][k]);
        }
        printf("\n");
    }
    for(i = 0; i< num_dis_rel; i++)
    {
        printf("\nExisting Discourse Relations   %s",dis_rel_list[i]);
    }*/


    for(i = 0; i< num_dis_rel; i++)
    {
        for(j = 0; j< num_dis_rel; j++)
        {
                strcpy(dis_rel_2_len[index_2_len_rel], dis_rel_list[i]);
                strcat(dis_rel_2_len[index_2_len_rel], "#");
`
                                strcat(dis_rel_2_len[index_2_len_rel], dis_rel_list[j]);
                                strcat(dis_rel_2_len[index_2_len_rel++], "#");

                }
        }

        for(i = 0; i< num_dis_rel; i++)
        {
                for(j = 0; j< num_dis_rel; j++)
                {
                        for(k = 0; k< num_dis_rel; k++)
                        {
                                strcpy(dis_rel_3_len[index_3_len_rel], dis_rel_list[i]);
                                strcat(dis_rel_3_len[index_3_len_rel], "#");
                                strcat(dis_rel_3_len[index_3_len_rel], dis_rel_list[j]);
                                strcat(dis_rel_3_len[index_3_len_rel], "#");
                                strcat(dis_rel_3_len[index_3_len_rel], dis_rel_list[k]);
                                strcat(dis_rel_3_len[index_3_len_rel++], "#");
                        }
                }
        }

        int     *count_1_len_rel, *count_2_len_rel, *count_3_len_rel;

        // index_2_len_rel represents possible 2 len discourse relations

        if (((count_1_len_rel = ( int*)malloc((num_dis_rel)* sizeof(int)))==NULL) ||
   ((count_2_len_rel = (int*)malloc((index_2_len_rel)* sizeof(int)))== NULL)||
   ((count_3_len_rel = (int*)malloc((index_3_len_rel)* sizeof(int))) == NULL))
        {
                printf("\nMemory Allocation failed in Count_*_Len_Rel\n");
                return ;
        }
        for (i = 0 ; i < num_dis_rel; i++)
                count_1_len_rel[i] = 0;

        for (i = 0 ; i < index_2_len_rel; i++)
                count_2_len_rel[i] = 0;
        for (i = 0 ; i < index_3_len_rel; i++)
                count_3_len_rel[i] = 0;

        /*      Calculating Probabilities for 2 length Sub Sequence   */

        int     total_num_1_len_sub_seq = 0,        total_num_2_len_sub_seq = 0;
   int  total_num_3_len_sub_seq = 0;

        total_num_1_len_sub_seq = find_all_1_len_sub_seq(DRM);
        total_num_2_len_sub_seq = find_all_2_len_sub_seq(DRM);
        total_num_3_len_sub_seq = find_all_3_len_sub_seq(DRM);

//      printf("\n\nTotal len_2_sub_seq_rel ----> %d  \n", total_num_2_len_sub_seq);
//      printf("\n\nTotal len_3_sub_seq_rel ----> %d  \n", total_num_3_len_sub_seq);

        char *rel1, *rel2, *rel3, *temp;
        if (( temp = ( char* )malloc(MAX_DRM_ENTRY_LEN )) == NULL )
        {
                printf("\nMemory Allocation failed\n");
```

```
`
                return ;
        }

        char    dis_rel_list_all[MAX_DISC_REL][MAX_WORD_LEN];

        fill_all_rel_type(dis_rel_list_all);

        k = 0;
        for(i = 0; i< MAX_DISC_REL; i++)
        {
                for(j = 0; j< MAX_DISC_REL; j++)
                {
                        strcpy(dis_rel_2_len_all[k], dis_rel_list_all[i]);
                        strcat(dis_rel_2_len_all[k], "#");
                        strcat(dis_rel_2_len_all[k], dis_rel_list_all[j]);
                        strcat(dis_rel_2_len_all[k++], "#");
                }
        }

        /*for(i = 0; i< MAX_DISC_REL *MAX_DISC_REL ; i++)
        {
                printf("\nAll possible 2 len relation%d: %s",i+1, dis_rel_2_len_all[i]);
        }*/

        l = 0;
        for(i = 0; i< MAX_DISC_REL; i++)
        {
                for(j = 0; j< MAX_DISC_REL; j++)
                {
                        for(k = 0; k< MAX_DISC_REL; k++)
                        {
                                strcpy(dis_rel_3_len_all[l], dis_rel_list_all[i]);
                                strcat(dis_rel_3_len_all[l], "#");
                                strcat(dis_rel_3_len_all[l], dis_rel_list_all[j]);
                                strcat(dis_rel_3_len_all[l], "#");
                                strcat(dis_rel_3_len_all[l], dis_rel_list_all[k]);
                                strcat(dis_rel_3_len_all[l++], "#");
                        }
                }
        }

        /*for(i = 0; i< MAX_DISC_REL *MAX_DISC_REL *MAX_DISC_REL ; i++)
        {
                printf("\nAll possible 3 len relation%d: %s",i+1, dis_rel_3_len_all[i]);
        }*/


/*******************************************************************/


        const int max_feature_index = MAX_DISC_REL + (MAX_DISC_REL*MAX_DISC_REL)
+
   (MAX_DISC_REL*MAX_DISC_REL*MAX_DISC_REL) + MAX_ENT_REL +

(MAX_ENT_REL*MAX_ENT_REL)+(MAX_ENT_REL*MAX_ENT_REL*MAX_ENT_REL)+MA
X_COREF_REL;
```

```
`
double feature_value[max_feature_index];

for (i = 0; i< max_feature_index; i++)
{
        feature_value[i] = 0.0;
}

int index = 0, ent_index = 0, disc_index = 0;
ent_index = MAX_DISC_REL + (MAX_DISC_REL*MAX_DISC_REL) +
             (MAX_DISC_REL*MAX_DISC_REL*MAX_DISC_REL);
disc_index = ent_index + MAX_ENT_REL + (MAX_ENT_REL*MAX_ENT_REL) +
               (MAX_ENT_REL*MAX_ENT_REL*MAX_ENT_REL);


fp2 = fopen(filename_grid, "r");
if( fp2 == NULL )
{
        printf("\nCan't open file:  %s\n",filename_grid);
        return ;
}

count_num_entity(fp2);
rewind(fp2);
count_num_statement(fp2);
rewind(fp2);

//printf("\nNumber of Entities: %d\n",entity_number);
//printf("\nNumber of Statement: %d\n",total_statement);

char **ent_rel_2_len;
char **ent_rel_3_len;
char **ent_rel_2_len_all;
char **ent_rel_3_len_all;
int     index_2_len_ent_rel = 0;
int     index_3_len_ent_rel = 0;

if (( ent_rel_2_len = ( char** )malloc( MAX_ENT_REL * MAX_ENT_REL *
                    sizeof( char* ))) == NULL )
{
        printf("\nMemory Allocation failed\n");
        return ;
}
if (( ent_rel_2_len_all = ( char** )malloc( MAX_ENT_REL * MAX_ENT_REL *
                    sizeof( char* ))) == NULL )
{
        printf("\nMemory Allocation failed\n");
        return ;
}
if (( ent_rel_3_len_all = ( char** )malloc( MAX_ENT_REL * MAX_ENT_REL *
            MAX_ENT_REL * sizeof( char* ))) == NULL )
{
        printf("\nMemory Allocation failed\n");
        return ;
}
if (( ent_rel_3_len = ( char** )malloc( MAX_ENT_REL * MAX_ENT_REL *
            MAX_ENT_REL * sizeof( char* ))) == NULL )
{
```

```
`
        printf("\nMemory Allocation failed\n");
        return ;
    }
    for ( j = 0; j < MAX_ENT_REL * MAX_ENT_REL; j++ )
    {
        if ((( ent_rel_2_len[j] = ( char* )malloc(MAX_WORD_LEN )) == NULL) ||  ((
ent_rel_2_len_all[j] = ( char* )malloc(MAX_WORD_LEN)) == NULL))
        {
            printf("\nMemory Allocation failed\n");
            return ;
        }
    }
    for ( j = 0; j < MAX_ENT_REL * MAX_ENT_REL * MAX_ENT_REL; j++ )
    {
        if ( ((ent_rel_3_len_all[j] = ( char* )malloc(MAX_WORD_LEN )) == NULL)||
((ent_rel_3_len[j] = ( char* )malloc(MAX_WORD_LEN)) == NULL) )
        {
            printf("\nMemory Allocation failed\n");
            return ;
        }
    }

    int tot_1_len_seq_ent=0, tot_2_len_seq_ent=0, tot_3_len_seq_ent=0;
    tot_1_len_seq_ent = entity_number * (total_statement - 1);
    tot_2_len_seq_ent = entity_number * (total_statement - 2);
    tot_3_len_seq_ent = entity_number * (total_statement - 3);

    char    ent_rel_list_all[MAX_ENT_REL][MAX_WORD_LEN];

    fill_all_ent_rel_type(ent_rel_list_all);

    /*for(i = 0; i< MAX_ENT_REL ; i++)
    {
        printf("\nAll possible 1 len relation%d: %s",i+1, ent_rel_list_all[i]);
    }*/

    k = 0;
    for(i = 0; i< MAX_ENT_REL; i++)
    {
        for(j = 0; j< MAX_ENT_REL; j++)
        {
            strcpy(ent_rel_2_len_all[k], ent_rel_list_all[i]);
            strcat(ent_rel_2_len_all[k], "#");
            strcat(ent_rel_2_len_all[k], ent_rel_list_all[j]);
            strcat(ent_rel_2_len_all[k++], "#");
        }
    }

    /*for(i = 0; i< MAX_ENT_REL *MAX_ENT_REL ; i++)
    {
        printf("\nAll possible 2 len relation%d: %s",i+1, ent_rel_2_len_all[i]);
    }*/

    l = 0;
    for(i = 0; i< MAX_ENT_REL; i++)
    {
        for(j = 0; j< MAX_ENT_REL; j++)
        {
            for(k = 0; k< MAX_ENT_REL; k++)
            {
                strcpy(ent_rel_3_len_all[l], ent_rel_list_all[i]);
                strcat(ent_rel_3_len_all[l], "#");
                strcat(ent_rel_3_len_all[l], ent_rel_list_all[j]);
                strcat(ent_rel_3_len_all[l], "#");
                strcat(ent_rel_3_len_all[l], ent_rel_list_all[k]);
                strcat(ent_rel_3_len_all[l++], "#");
            }
        }
    }

    /*for(i = 0; i< MAX_ENT_REL *MAX_ENT_REL*MAX_ENT_REL ; i++)
    {
        printf("\nAll possible 3 len relation%d: %s",i+1, ent_rel_3_len_all[i]);
    }*/


    char ***ERM;    //      line_number = Number of sentences
    if (( ERM = ( char*** )malloc( (entity_number)*sizeof( char** ))) == NULL )
    {
        printf("\nMemory Allocation failed\n");
        return ;
    }                       //          token_number = Number of Stemmed token in DRM

    for(i = 0; i< entity_number; i++)
    {
        if (( ERM[i] = (char**)malloc((total_statement)*sizeof( char* )))==NULL)
        {
            printf("\nMemory Allocation failed\n");
            return ;
        }
    }

    for(i = 0; i< entity_number; i++)
    {
        for ( j = 0; j < total_statement; j++ )
        {
            if (( ERM[i][j] = ( char* )malloc(MAX_ERM_ENTRY)) == NULL )
            {
                printf("\nMemory Allocation failed\n");
                return ;
            }
        }
    }

    fill_entity_rel(fp2, ERM);

    /*for(i = 0; i < entity_number; i++)
    {
        printf("\nERM Row:%d     ",i);
        for(j = 1; j < total_statement; j++)
        {
            printf("  %s",ERM[i][j]);
        }
    }*/
```

`

```
        int     *count_1_len_ent_rel, *count_2_len_ent_rel, *count_3_len_ent_rel;

        // index_2_len_rel represents possible 2 len discourse relations

        if (((count_1_len_ent_rel=(int*)malloc((MAX_ENT_REL)*sizeof(int)))== NULL)||
      ((count_2_len_ent_rel = ( int* )malloc(MAX_ENT_REL*MAX_ENT_REL*
      sizeof(int))) == NULL) || ((count_3_len_ent_rel = ( int* )malloc(
      MAX_ENT_REL*MAX_ENT_REL*MAX_ENT_REL* sizeof(int))) == NULL))
        {
                printf("\nMemory Allocation failed in Count_*_Len_Rel\n");
                return ;
        }
        for (i = 0 ; i < MAX_ENT_REL; i++)
                count_1_len_ent_rel[i] = 0;

        for (i = 0 ; i < MAX_ENT_REL*MAX_ENT_REL; i++)
                count_2_len_ent_rel[i] = 0;

        for (i = 0 ; i < MAX_ENT_REL*MAX_ENT_REL*MAX_ENT_REL; i++)
                count_3_len_ent_rel[i] = 0;

        count_len_1_ent_seq(ERM, count_1_len_ent_rel);
        /*for(i = 0; i < MAX_ENT_REL; i++)
        {
                printf("\nCount for 1 len Entity Seq:   %d",count_1_len_ent_rel[i]);
        }*/
        count_len_2_ent_seq(ERM, count_2_len_ent_rel, ent_rel_2_len_all);

        count_len_3_ent_seq(ERM, count_3_len_ent_rel, ent_rel_3_len_all);

        for (i = 0 ; i < MAX_ENT_REL; i++)
                feature_value[ent_index + i] = (double) count_1_len_ent_rel[i] /
                                (double) tot_1_len_seq_ent;

        for (i = 0 ; i < MAX_ENT_REL*MAX_ENT_REL; i++)
                feature_value[ent_index+MAX_ENT_REL+i] =(double) count_2_len_ent_rel[i]/
                               (double) tot_2_len_seq_ent;

        for (i = 0 ; i < MAX_ENT_REL*MAX_ENT_REL*MAX_ENT_REL; i++)
                feature_value[ent_index + MAX_ENT_REL + MAX_ENT_REL*MAX_ENT_REL
+ i] =
    (double) count_3_len_ent_rel[i] / (double) tot_3_len_seq_ent;


        fclose(fp2);
/************************************************************************/
//      printf("\nNumber of 1 Len Discourse Sequence %d\n\n", num_dis_rel);
//      printf("\nNumber of 2 Len Discourse Sequence %d\n\n", index_2_len_rel);
        for(i = 0; i< num_dis_rel; i++)
        {
                for(j = 0; j< token_number; j++)
                {
                        for(k = 0; k< line_number; k++)
                        {
                                if(find_rel(dis_rel_list[i], DRM, k, j) == TRUE)
                                {
```

`

```
                                        count_1_len_rel[i]++;
                                    }
                                }
                        }
                        index = find_1_len_rel(dis_rel_list_all, dis_rel_list[i]);
                        feature_value[index] = (double) count_1_len_rel[i] /
                                (double) total_num_1_len_sub_seq;
                }

                for(i = 0; i< index_2_len_rel; i++)
                {
                        strcpy(temp, dis_rel_2_len[i]);
                        rel1 = strtok( temp, "#" );
                        rel2 = strtok( NULL, "#" );
                        //puts(dis_rel_2_len[i]);
                        for(j = 0; j< token_number; j++)
                        {
                                for(k = 0; k< line_number - 1; k++)
                                {
                                        if(find_rel(rel1, DRM, k, j) == TRUE)
                                        {
                                                if(find_rel(rel2, DRM, k+1, j) == TRUE)
                                                        count_2_len_rel[i]++;
                                        }
                                }
                        }
                        index = find_2_len_rel(dis_rel_2_len_all, dis_rel_2_len[i]);
                        feature_value[MAX_DISC_REL + index] = (double) count_2_len_rel[i] /     (double)
        total_num_2_len_sub_seq;
                //      printf("\nFeature Value:  %d:%lf", i+1, feature_value[index] );
                }

                for(i = 0; i< index_3_len_rel; i++)
                {
                        strcpy(temp, dis_rel_3_len[i]);
                        rel1 = strtok( temp, "#" );
                        rel2 = strtok( NULL, "#" );
                        rel3 = strtok( NULL, "#" );
                        //puts(dis_rel_2_len[i]);
                        for(j = 0; j< token_number; j++)
                        {
                                for(k = 0; k< line_number - 2; k++)
                                {
                                        if(find_rel(rel1, DRM, k, j) == TRUE)
                                        {
                                                if(find_rel(rel2, DRM, k+1, j) == TRUE)
                                                {
                                                        if(find_rel(rel3, DRM, k+2, j) == TRUE)
                                                        {
                                                                count_3_len_rel[i]++;
                                                        }
                                                }
                                        }
                                }
                        }
                        index = find_3_len_rel(dis_rel_3_len_all, dis_rel_3_len[i]);
                        feature_value[MAX_DISC_REL+ MAX_DISC_REL*MAX_DISC_REL + index] =
        `
            (double) count_3_len_rel[i] / (double) total_num_3_len_sub_seq;

                }


        /***************************************************************/

                char *np_num, *coref_num;
                int temp_count = 0, max_hobb_dist = 0;
                np_num = ( char* )malloc(MAX_DRM_ENTRY_LEN );
                coref_num = ( char* )malloc(MAX_DRM_ENTRY_LEN );


                char **coref_list;
                int *count_coref, index_coref = -1, max_coref_index = -1, index_entity = -1;
          int  index_anaphor = -1, hobb_dist = 0;
                count_coref = ( int* )malloc(MAX_COREF_REL * sizeof(int) );
                for ( j = 0; j < MAX_COREF_REL; j++ )
                {
                        count_coref[j] = 0;
                }

                if (( coref_list = ( char** )malloc( MAX_COREF_REL * sizeof( char*)))==NULL)
                {
                        printf("\nMemory Allocation failed\n");
                        return ;
                }
                for ( j = 0; j < MAX_COREF_REL; j++ )
                {
                        if (( coref_list[j] = ( char* )malloc(MAX_DRM_ENTRY_LEN )) == NULL)
                        {
                                printf("\nMemory Allocation failed\n");
                                return ;
                        }
                        strcpy(coref_list[j], "-1");
                }

                fp3 = fopen(filename_coref, "r");
                if( fp3 == NULL )
                {
                        printf("\nCan't open file:  %s\n",filename_coref);
                        return ;
                }

                while (!feof(fp3))
                {
                        fflush(stdin);
                        fscanf(fp3, "%s", buffer);
                        if(strcmp(buffer+strlen(buffer)-2, "NP") == 0)
                        {
                                fscanf(fp3, "%s", np_num);
                                np_num = np_num+4;
                                *(np_num+ strlen(np_num)-1) = '\0';
                                fscanf(fp3, "%s", buffer);
                                buffer = strtok(buffer, ">");
                                strcpy(coref_num, buffer+9);
                                *(coref_num+ strlen(coref_num)-1) = '\0';
                    `
                                index_coref = atoi(coref_num);
                                if(index_coref > max_coref_index)
                                    max_coref_index = index_coref;
                                if(strcmp(coref_list[index_coref], "-1") == 0)
                                {
                                    strcpy(coref_list[index_coref], np_num);
                                    strcat(coref_list[index_coref], "#");
                                }
                                else
                                {
                                    strcat(coref_list[index_coref], np_num);
                                    strcat(coref_list[index_coref], "#");
                                }
                                count_coref[index_coref]++;
                      }
                        else
                                continue;

                }
                index = 0;
                    //  printf("\nMax_coref_index:   %d",max_coref_index);
                for ( j = 0; j <= max_coref_index; j++ )
                {
                        temp_count = 0;
                        max_hobb_dist = 0;
                        if(count_coref[j] > 4)
                        {
                                buffer = (char *) malloc(MAX_DRM_ENTRY_LEN);
                                strcpy(buffer, coref_list[j]);
                                //puts(coref_list[j]);
                                //buffer = strtok( buffer, "#" );
                                index_entity = j;
                                while( buffer != NULL )
                                {
                                    buffer = strtok( NULL, "#" );
                                    if(buffer == NULL)
                                        break;
                                    index_anaphor = atoi(buffer);
                                    hobb_dist = index_anaphor - index_entity;
                    //              printf("\nHobb_dist:  %d",hobb_dist);
                    //              index_entity = index_anaphor;
                                    feature_value[disc_index + index++] =
                                                        (double) hobb_dist;
                                    if(hobb_dist < 0)
                                        hobb_dist = -hobb_dist;
                                    if(hobb_dist > max_hobb_dist)
                                        max_hobb_dist = hobb_dist;
                                    temp_count++;
                                }
                                //printf("\nTemp_count :  %d",temp_count);
                                for(i = temp_count; i > 0; i--)
                                {
                                feature_value[disc_index + index - i] /= max_hobb_dist;
                                }
                                index += (MAX_COREF_COUNT - temp_count);
                        }
                }
```

```
`

    fclose(fp3);

/*******************************************************************/

    fp = fopen("test1.dat", "a");
    fp1 = fopen("count_test1.dat", "r+");

    int id;
    char c, *num;
    num = (char *) malloc(MAX_WORD_LEN);
    fscanf(fp1, "%s", num);

    id = atoi(num);
    rewind(fp1);
    if(strcmp(filename_parse+(strlen(filename_parse) - 11), "-1-p.parsed") == 0)
    {
        fprintf (fp, "%s", "2 qid:");// qid =2 for original text
        fprintf (fp, "%s", num);
        fprintf (fp1, "%s", num);
    }
    else
    {
        fprintf (fp, "%s", "1 qid:");  // qid =1 for permuted text
        fprintf (fp, "%s", num);
        cvtInt( &num, ++id);
        fprintf (fp1, "%s", num);
    }

    for(i = 0; i< max_feature_index; i++)
    {
        if(feature_value[i] != 0.0)   // writting non-zero features to vector
            fprintf(fp, " %d:%lf", i+1, feature_value[i]);
    }

    fprintf (fp, "%s", " # ");
    fprintf (fp, "%s\n", argv[1]);          // file name for reference

    fclose(fp1);
    fclose(fp);

    return 0;

}
```

`

APPENDIX A: ABBREVATIONS

NLP	Natural Language Processing
SVM	Support Vector Machine
NER	Named Entity Recognition
POS	Part of Speech
GPL	General Public License
PDTB	Penn Discourse Treebank
NP	Noun Phrase
AI	Artificial Intelligence
NLU	Natural Language Understanding
ACE	Arabic, Chinese and English
MUC	Message Understanding Conference