UNIVERSIDADE DA BEIRA INTERIOR
Engenharia

Rule Induction for Sentence Reduction

João Paulo da Costa Cordeiro

Tese para a obtenção do Grau de Doutor em
Engenharia Informática
(3º ciclo de estudos)

Orientador: Prof. Doutor Gaël Harry Dias
Co-Orientador: Prof. Doutor Pavel Bernard Brazdil

Covilhã, Novembro de 2010

I would like to dedicate this thesis to my loving wife, *Adelina Amorim*, and my three precious children: *David*, *Tiago*, and *André*.

Acknowledgements

A work like this is almost impossible to be achieved by a single individual working alone. There are always several very important persons involved, more or less directly and in various levels: scientific, personal, or even in the sentimental/spiritual level. In this few lines I would like to express my deepest gratitude to those having been determinate for the conclusion of this long and hard work.

I would like to start by acknowledging my supervisor, *Prof. Doctor Gaël Harry Dias*, for his constant and relentless support throughout this journey. He was indeed an always presently supporter, guiding me many times, and even motivating me toward new unexplored scientific and technological territories. My co-supervisor, *Professor Pavel Bernard Brazdil*, was equally very important for the developing and conclusion of this work. With his long experience as a leading scientist, every piece of advise received from him were carefully observed and incorporated here. I also want to thank all the teachers I had, specially during my B.Sc. and M.Sc. degrees, who contributed significantly to my scientific education.

At a personal level, I must thank my family, specially my wife, *Adelina Amorim*, for her constant and almost unconditional support, during several years, allowing me to be absent many times from our three children's family duties. I also thank my parents, *António Cordeiro* and *Emília Madeira*, for their constant support. Last but not least, I would like to thank God for all the love, meaning, peace, and direction brought to my life. One day the God from the Bible changed my life, making me a very new being! It is an honor to finish this acknowledgments by including the first three verses from a beautiful biblical poem:

> "*The LORD is my shepherd; I shall not want. He maketh me to lie down in green pastures: he leadeth me beside the still waters. He restoreth my soul: he leadeth me in the paths of righteousness for his name's sake.*"
>
> **Psalms 23:1-3**

Abstract

The field of *Automatic Sentence Reduction* has been an active research topic, with several relevant approaches being recently proposed. However, in our view many milestones still need to be reached in order to approach human-like quality sentence simplification. In this work, we propose a new framework, which processes huge sets of *web news stories* and learns sentence reduction rules in *a fully automated and unsupervised way*. This is our main contribution. Our system is conceptually composed of several modules. In the first one, the system automatically extracts paraphrases from on-line *news stories*, using new lexically based functions that we have proposed. In our system's second module, the extracted paraphrases are transformed into aligned paraphrases, meaning that the two paraphrasic sentences get their words aligned through DNA-like sequence alignment algorithms, that has been conveniently adapted for aligning sequences of words. These alignments are then explored and specific text structures called *bubbles* are selected. Afterwards, these structures are transformed into learning instances and used in the last learning module that exploits techniques of *Inductive Logic Programming*. This module learns the rules for sentence reduction. Results show that this is a good approach for learning automatic sentence reduction, while some pertinent issues still need future investigation.

Keywords

Sentence reduction, sentence compression, sentence simplification, paraphrase extraction, paraphrase alignment, automatic text summarization, natural language processing, inductive logic programing, machine learning, artificial intelligence.

Resumo

A área da *Redução Automática de Frases* tem sido um tópico activo de investigação, tendo sido propostas recentemente várias abordagens relevantes. Todavia, do nosso ponto de vista, para que tenhamos uma simplificação de frases mais parecida com a que é feita por humanos, é necessário ainda alcançar várias etapas importantes. Neste trabalho propomos uma nova abordagem que aprende regras de redução de frases, de forma completamente automática e não supervisionada e a partir do processamento de um elevado volume de texto de *notícias da web*. Esta é a nossa contribuição principal. O nosso sistema é conceptualmente composto por vários módulos. No primeiro, o sistema extrai automaticamente paráfrases de *notícias web* através de novas funções lexicais que propusemos. No segundo modulo do nosso sistema, as paráfrases extraídas são transformadas em paráfrases alinhadas, significando isto que duas frases parafrásticas ficam com as suas palavras alinhadas entre si, através de algoritmos alinhamento de sequências de ADN que foram adaptados para o alinhamento de sequências de palavras. Posteriormente estes alinhamentos são explorados para que determinadas estruturas relevantes, denominadas de bolhas, sejam seleccionadas e transformadas em instâncias de aprendizagem. Estas, por sua vez, serão utilizadas no último módulo do nosso sistema que através da *Programação Lógica Indutiva* aprende regras de redução de frases. Os resultados mostram que esta é uma abordagem promissora para a redução automática de frases, sendo todavia ainda necessário continuar a investigação nalgumas questões pertinentes deste domínio.

Palavras-chave

Redução de frases, compressão de frases, simplificação de frases, extracção de paráfrases, alinhamento de paráfrases, sumarização automática de texto, processamento da linguagem natural, programação lógica indutiva, aprendizagem automática, inteligência artificial.

Resumo Alargado

Em termos gerais, o trabalho consistiu na criação de um sistema com capacidade para "aprender" regras de simplificação de frases, por redução de certas componentes menos relevantes, a partir da "observação" de corpora. Trabalhando somente com textos de notícias relacionadas, extraídas diariamente da *Internet*, o nosso sistema tem a capacidade de induzir estas regras de redução de frases, através de técnicas de *aprendizagem automática*. O sistema desenvolvido é composto por quatro componentes principais:

1. Extracção de paráfrases.
2. Alinhamento de termos em paráfrases.
3. Extracção de bolhas.
4. Indução de regras de redução.

A primeira componente consiste na extracção automática de paráfrases, que é realizada a partir de textos de notícias relacionadas, obtidas diariamente a partir do "*Google News*"[1]. Um determinado evento noticioso, como por exemplo um *atentado terrorista*, ou um *discurso presidencial*, contém no mínimo cerca de 30 textos oriundos de diversas fontes noticiosas. São textos que veiculam a mesma informação, no entanto escritos usando formas e estilos diferentes. Estas colecções constituem um "terreno" propício à extracção de paráfrases. Assim, foram propostas várias funções, baseadas em ligações lexicais entre frases, para a identificação de um tipo de paráfrases conveniente para o nosso trabalho, isto é as paráfrases assimétricas, nas quais uma das frases do par contém mais informação que a outra, tal como mostrado no exemplo que se segue:

```
O ministro, que referiu o problema do desemprego, anunciou a subida do IRC.
O ministro anunciou a subida do IRC.
```

A segunda componente do nosso trabalho consiste no alinhamento das palavras que compõe o par de frases da paráfrase. Foram adaptados algoritmos de alinhamento de sequências de ADN para alinhar sequências de palavras. Também propusemos um novo método para escolher dinamicamente o algoritmo de alinhamento (global ou local), em tempo de execução, consoante seja mais conveniente para o par considerado. Ainda neste módulo, propusemos uma nova função simulando a "mutação genética" entre as palavras de uma paráfrase. Esta função tem como objectivo permitir o alinhamento de termos lexicalmente próximos. Um exemplo de um alinhamento global, para a paráfrase apresentada anteriormente é mostrado a seguir:

[1] URL: http://news.google.com [Novembro de 2010]

```
O ministro, que referiu o problema do desemprego, anunciou a subida do IRC.
O ministro, ___ _______ _ ________ __ __________, anunciou a subida do IRC.
```

Portanto, as duas primeiras etapas do sistema tem a capacidade de gerar automaticamente um corpus de paráfrases alinhadas com elevado número de casos (na ordem dos milhões de exemplos), a partir de texto de notícias publicadas electronicamente. Este corpus constitui a matéria-prima utilizada no processo de indução de regras de redução de frases.

Na terceira componente do sistema, partir de um corpus de paráfrases alinhadas, são extraídas um conjunto de instâncias de aprendizagem, denominadas *bolhas*, com o objectivo de "alimentar" um sistema indutor de regras de simplificação de frases. Pela "observação" de milhares de casos são generalizadas regras que permitem cortar certas porções menos relevantes de frases. Uma *bolha* consiste numa porção extraída de um alinhamento, no qual dois segmentos diferentes estão alinhados e delimitados por pares de segmentos iguais, um à esquerda, denominado de *contexto esquerdo* e outro à direita, denominado de *contexto direito*. Ao alinhamento heterogéneo central foi dado o nome de "centro" (o centro da bolha). O exemplo apresentado anteriormente contem precisamente um caso de uma bolha, tomando para contexto esquerdo "`O ministro ,`", para contexto direito a sequência "`, anunciou a subida do IRC .`" e para o centro o par:

```
("que referiu o problema do desemprego"  --->  "")
```

As duas sequências centrais, indicam uma possível transformação de uma expressão linguística de tamanho m para uma outra de tamanho zero, mediante um contexto esquerdo e direito. Foram investigadas e propostas condições para a extracção de bolhas, a partir de paráfrases alinhadas. A função proposta garante que esta extracção só se concretiza se existirem contextos "fortes", i.e. com um tamanho significativo em relação ao tamanho do centro.

Na quarta componente do nosso sistema, foram exploradas técnicas de *Programação Lógica Indutiva*, em especial o sistema *Aleph*, para induzir regras de redução de frases. Estas são aprendidas através de grandes colecções de *bolhas* extraídas automaticamente, como descrito anteriormente.

A abordagem que propusemos para este problema é completamente inovadora, tendo-nos possibilitado alcançar um conjunto significativo de publicações relevantes em importantes conferências e revistas internacionais da especialidade.

Contents

Contents xiii

1 Introduction 1

1.1 Automatic Text Summarization . . . 2

1.1.1 Origins of Automatic Text Summarization . . . 3

1.1.2 The "Renaissance" in Automatic Summarization . . . 5

1.2 Motivation and Objectives . . . 6

1.3 Work Overview . . . 8

1.4 Main Scientific Achievements . . . 12

1.5 Thesis Plan . . . 13

2 Related Work in Sentence Reduction 15

2.1 Automatic Translation Methodologies . . . 16

2.1.1 Origins - Pioneering Projects . . . 16

2.1.2 Using an HMM for Optimizing Translation Templates . . . 20

2.2 Heuristic Based Methodologies . . . 23

2.2.1 Using Rich Linguistic Knowledge . . . 23

2.2.2 As an Optimization Problem . . . 25

2.3 Machine Learning Methodologies . . . 27

2.3.1 Training a Noisy-Channel Model . . . 27

2.3.2 A Symbolic Learning Method . . . 31

2.3.3 Mixing ML and Predefined Rules . . . 34

3 Paraphrase Extraction 37

3.1 Automatic Paraphrase Corpora Construction . . . 38

3.2 Functions for Paraphrase Identification . . . 39

3.2.1 The Levenshtein Distance . . . 41

3.2.2 The Word N-Gram Family . . . 42

3.2.2.1 Word Simple N-gram Overlap . . . 42

3.2.2.2 The BLEU Function . . . 42

3.2.2.3 Exclusive LCP N-gram Overlap . . . 43
3.3 New Functions for Paraphrase Identification . . . 45
3.3.1 The *Sumo* Function . . . 48
3.4 Paraphrase Clustering - An Investigated Possibility . . . 52
3.5 Extraction Time Complexity . . . 54
3.6 Final Remarks . . . 55

4 Paraphrase Alignment **57**
4.1 Biology-Based Sequence Alignment . . . 58
4.1.1 Global Alignment . . . 59
4.1.2 Local Alignment . . . 62
4.2 Dynamic Alignment . . . 67
4.3 Modeling a Similarity Matrix for Word Alignment . . . 70
4.3.1 The Quantitative Value of an Alignment . . . 74
4.4 System Execution . . . 75
4.4.1 Extraction of Web News Stories . . . 75
4.4.2 Paraphrase Extraction and Alignment . . . 76
4.4.3 Class Hierarchy Diagram . . . 78
4.5 Final Remarks . . . 80

5 Inductive Logic Programming **83**
5.1 Machine Learning . . . 83
5.2 Logic Programming . . . 87
5.2.1 Inference Rules . . . 89
5.3 ILP Generals . . . 90
5.3.1 The Structured Hypothesis Search Space . . . 93
5.3.2 Inverse Resolution . . . 96
5.4 The *Aleph* System . . . 99
5.5 Final Remarks . . . 105

6 Induction of Sentence Reduction Rules **107**
6.1 Bubble Extraction . . . 107
6.2 System Execution . . . 111
6.2.1 Data Preparation . . . 111
6.2.2 Exemplifying the Process of Induction . . . 117
6.3 Learned Rules and their Applications . . . 119
6.4 Induction Time Complexity . . . 125
6.5 Final Remarks . . . 125

7 Results 127

7.1 Paraphrase Extraction . . . 127

7.1.1 The Microsoft Paraphrase Corpus . . . 127

7.1.2 The Knight and Marcu Corpus . . . 129

7.1.3 The Evaluation Paraphrase Corpora . . . 129

7.1.3.1 The $MSRPC \cup X^{-}_{1999}$ Corpus . . . 130

7.1.3.2 The $KMC \cup X^{-}_{1087}$ Corpus . . . 130

7.1.3.3 The $MSRPC^{+} \cup KMC \cup X^{-}_{4987}$ Corpus . . . 131

7.1.4 How to Classify a Paraphrase? . . . 131

7.1.5 Experiments and Results . . . 133

7.1.6 The (SP type) Bleu Function - A Special Case . . . 136

7.1.7 The Influence of Random Negative Pairs . . . 137

7.2 Paraphrase Alignment . . . 138

7.2.1 Paraphrase Corpora . . . 138

7.2.2 Quality of Paraphrase Alignment . . . 139

7.3 Application of Reduction Rules . . . 141

7.3.1 Evaluation Measures . . . 142

7.3.2 Evaluation Results . . . 144

8 Conclusion and Future Directions 147

8.1 Conclusions . . . 147

8.2 Future Trends . . . 150

A Mathematical Logic 153

A.1 Propositional Logic . . . 153

A.2 First Order Logic . . . 156

B Relevant Code Fragments 165

C Relevant *Aleph* Elements 169

D System Execution Example 179

E The Penn Treebank Tag Set 183

References 192

List of Figures

1.1 A table from Luhn (1958) showing statistically significant words automatically selected from a scientific article. 4
1.2 The Pipeline Architecture. 9
1.3 An example of an asymmetrical paraphrase extracted from related news events. . . . 9
1.4 An aligned paraphrase sentence pair. 9
1.5 Sentence Reduction Rules, automatically learned. 10
1.6 Sentence schematic division. 10

2.1 An example of a learned template-translation, in the context of sentence reduction. It is assumed that T_1 is in some sense a shorter version of S_2. 20
2.2 An example of a template reduction rule, learned from two similar cases. 21
2.3 The noisy channel model for sentence transmission. 28
2.4 The parse tree for the "*George meets Kate*" sentence. 29
2.5 A rank of reduced sentences, obtained from a given sentence. 30
2.6 Illustration of a transformation for a sentence tree scheme. 32
2.7 Three learned rules of sentence syntactic tree transformation toward reduction. . . . 33
2.8 A sequence of 9 steps showing the operators for transforming $t(s)$ into $t(s_r)$. 33
2.9 Shallow parse tree, with their branch probabilities estimated. 35
2.10 Examples of handcrafted constraints ensuring grammatical correctness. 36

3.1 Lexical paraphrase type. 37
3.2 Semantic paraphrase type. 37
3.3 An asymmetrical parafrase. 38
3.4 *Hill shape* functions for paraphrase identification. 46
3.5 Exclusive lexical links between a sentence pair. 47
3.6 An example of a too similar paraphrase. 47
3.7 A graphical representation for $z = S(x,y)$, with $\alpha = \beta = 0.5$, represented through four different views. 51
3.8 A graphical representation for $y = S(x,k)$, with $\alpha = \beta = 0.5$, and $k \in \{\frac{1}{2}, \frac{3}{4}, 1\}$. . . . 52

4.1 A possible alignment for two paraphrase sentences. 57
4.2 Another possible alignment for the same paraphrase. 58
4.3 DNA sequence alignment. 59
4.4 A global alignment for two paraphrase sentences. 62
4.5 A paraphrase sentence pair which include interchange of paraphrases. 67
4.6 A paraphrase sentence pair with relevant asymmetrical lengths. 68
4.7 Crossings between a sentence pair. 68
4.8 Maximum number of crossings in a complete word inversion case. 69
4.9 The BLOSUM62 substitution matrix. 71
4.10 A paraphrase sentence pair with relevant asymmetrical lengths. 73
4.11 A link pointing to a cluster of web news stories related to a given event. 76
4.12 Illustration selected from an xml web news file, produced by the "GoogleNewsSpider" program. 77
4.13 An xml file of aligned paraphrases, showing the initial part of three examples. 78
4.14 A Selection of the main classes designed for text data representation. 79

5.1 A learned decision tree for the *play tennis* problem. 84
5.2 The "member" predicate, with two clauses and being defined recursively. 88
5.3 The general resolution scheme. 90
5.4 "Humans in love!" 91
5.5 Example: "Why are they different?" 92
5.6 A lattice Hasse diagram for $\mathcal{P}(\{x,y,z\})$. 94
5.7 Part of the refinement graph, for learning the concept "daughter" (Lavrac & Dzeroski 1994) 96
5.8 An *inverse resolution* first step application for the "In Love" example. 98
5.9 An *inverse resolution* second step applied for the "In Love" example (see also Figure 5.8). 99

6.1 Two word-aligned sentences. Each word is represented with a letter. 108
6.2 The only six EQsegments from the alignment of Figure 6.1. 108
6.3 Seven examples of pair-sub-segments containing four bubbles. 108
6.4 The *Bubble* main components: L, X, and R. 109
6.5 The *bubble* term representation. 109
6.6 Examples of extracted bubbles 110
6.7 Examples of two rejected bubbles 110
6.8 Bubbles converted in *Aleph* positive learning instances. 113
6.9 The *.b *Aleph* template header. 114
6.10 Our rule evaluation function, defined in the *.b template file. 115
6.11 Special *Aleph* constraints included in our *.b template file. 116
6.12 Bubble middle size distribution for a set of 143761 extracted bubbles. 120

6.13 Time spent during the induction process, for datasets with size expressed in thousands of bubbles. 126

7.1 Guidelines used by human judges in the *MSRPC* paraphrase corpus construction to determine equivalent pairs. 128
7.2 An asymmetrical paraphrase pair likely to be rejected according to the *MSRPC* construction guideline. 128
7.3 An example of a negative paraphrase pair, randomly selected from *Web News Texts* . 130
7.4 An example of a quasi-equal negative paraphrase pair. 130
7.5 Threshold performance curves for 4 paraphrase identification functions, tested on the $MSRPC^{+} \cup KMC \cup X^{-}_{4987}$ corpus. 132

B.1 The *Sumo-Metric* method for computing sentence similarity. 165
B.2 The "printAlignments" method for computing and printing paraphrase sentence alignments to an XML file. 166
B.3 Count the relative number of *crossings* between two sentences. Method from the *Sentence* class and related to the previous listing. 167

List of Tables

3.1 *Sumo-Metric* output examples. 50
3.2 Precision of clustering algorithms 53
3.3 Figures about clustering algorithms 53
3.4 Paraphrase extraction times for six different functions. 54

4.1 Example of local alignments. 63
4.2 Comparing two word mutation functions. 73

5.1 Quinlan's "Play Tennis" learning example. 84
5.2 Relations expressed in FOL. 92
5.3 Examples of *least general generalizations*. 95

6.1 Bubble values for the previous five examples. 111
6.2 Some examples of good sentence reduction rules showing the eliminated sentence portions in each case. 121
6.3 Six examples of bad sentence reduction rule applications. 122

7.1 Threshold means and standard deviations, obtained using a 10-fold cross validation procedure. 133
7.2 The *confusion matrix* for a binary classification problem. 134
7.3 F_1 evaluation results obtained (in %). 135
7.4 $Accuracy$ obtained (in %). 135
7.5 The $BLEU$ results (in %), with N decreasing. 136
7.6 SP-functions on a corpus without quasi-equal negative pairs 137
7.7 AP-functions on a corpus without quasi-equal negative pairs 137
7.8 The $algval(A_{2\times n})$ values for the previous nine examples shown. 140
7.9 Accuracy obtained in the alignments 141
7.10 Results with four evaluation parameters. 145

A.1 A true table for the formula $A \Rightarrow (B \wedge C)$, for all possible combination of values assumed by their propositional symbols. 155

A.2 The $\mathcal{L}$ language fundamental components. 156

E.1 The *Penn Treebank Tag* Set (1-36). 183

E.2 The *Penn Treebank Tag* Set (37-48: punctuation marks). 184

Chapter 1

Introduction

> "*We can only see a short distance ahead, but we can see plenty there that needs to be done.*"
>
> **Alan Turing**

The advent of the so called *Information Age*, since the beginning of the 1990's and specially due to the dawn and quick expansion of the *World Wide Web* (WWW), has produced a huge amount of electronic on-line documentation of several types: text, image, video and audio. In the past, knowledge was something very rare and only accessible to those belonging to a very restrict elite, like rulers, clerics and naturally the narrowed academic community, which in most of the times was fused with the clerics, as for example in the middle age. Hopefully, nowadays knowledge has benefited from an admirable expansion through scientific, technological, philosophical, among others, remarkable improvements and discoveries, and furthermore through electronic means it has become much more democratized and widespread worldwide. Unlike to what happened in the past, people are now facing a new difficulty which can be synthesized as the problem of information *overload*, the sense of limitation that a present user feels whenever facing the huge amount of available information to be "digested" in feasible time, especially information published in text format.

Considering this context and naturally the continued increasing computational power at our disposal, during the last twenty years, several new information processing research fields have emerged and developed during this time frame, such as *Information Retrieval* (IR), *Information Extraction* (IE) and *Automatic Text Summarization* (ATS). All these three fields share the general common concern of textual information selection/filtering from the huge whole of information available. In IR the main goal is to select relevant documents related to a given subject, according to a specified user query. In IE the goal is to select key-elements from text, for example name-entities, and in ATS the main objective consists in producing more or less compressed summaries from given original texts, without losing syntax, coherence, and the key semantic elements. In ATS one aims at generating the minimum possible set of textual units from an original text maintaining the main original ideas, also referred to as the *gist* (Jones, 1993; Mani, 2001; Nguyen *et al.*, 2004) of the original text.

The field of *Text Summarization* is an old concern, studied even before the dawn of the *Informa-*

tion Age. Since the earlier 1960's that we have professional summarizers, working in many domains, from journalism to technical documentation. The summarization of text documents, still remains and even became a more crucial issue nowadays, with new electronic documents being published every day. The best summarization is still produced by humans. However, it is costly to have sufficient numbers of professional summarizers to tackle the full necessities for the present information volumes. Therefore automatic text summarization mechanisms have been gaining relevance. The ultimate goal in ATS is to approximate, as near as possible, its quality of performance to the one of human beings. This is also the main reason explaining the increasing attention ATS given recently by the research community, with many challenges ahead, waiting to be solved.

The next section provides a general overview of the *Automatic Text Summarization* research area as an introductory background for the work we have been developing, which consists in automatically learn sentence reduction rules that can be applied to simplify new texts.

1.1 Automatic Text Summarization

Among many possible Automatic Text Summarization (ATS) definitions, published in many scientific articles and books throughout the years, we have selected the following three of different authors, each one widely known and closely related to this research area.

> *A reductive transformation of source text to summary text through content condensation by selection and/or generalization on what is important in the source.*
>
> Spärk Jones

> *A summary is a text that is produced from one or more texts, that contains a significant portion of the information in the original text(s), and that is no longer than half of the original text(s).*
>
> Eduard Hovy

> *Text summarization is the process of distilling the most important information from a source (or sources) to produce an abridged version for a particular user and task.*
>
> Mani and Maybury

Generally, ATS is mainly concerned with the simplification of text, keeping as much as possible their original key information units. The main motivation is naturally applicational, aiming to ease the user's work, by providing him with short versions of relatively long texts. The "user" here means any

person working with electronic texts and aiming to have summarization capabilities in its system. There are two opposite forces conditioning any ATS process, which may be stated as *size* and *content*. In the limit ATS is an *utopia*, because beyond a certain horizon, the minimum text length containing the maximum original information will simply become a pure mirage. There are clear boundaries and the natural implication of text reduction will necessary result in information loss. Therefore the key success of any ATS system relays on its ability to establish a trade-off between these two issues, satisfying the particular user's needs at a given moment. Sometimes one may be more concerned with time wishing to do fast reading, so a short overview serves well, while in other situations time may be less relevant than losing original information details, and a less *compressed* version of the original text may be appropriate.

1.1.1 Origins of Automatic Text Summarization

Historically ATS started approximately 50 years ago with the work of Hans Peter Luhn, an IBM scientist which in 1958 published an article entitled "*The Automatic Creation of Literature Abstracts*" (Luhn, 1958). This was the first computational approach to the problem of text summarization. At that time an IBM 704 computer was used to calculate word relevance statistics for a given set of scanned documents, based on their frequency and distribution. Relevant words were chosen considering their frequency range delimited by pre-defined upper (C) and lower (D) boundaries. Words having frequency greater than C were considered *not discriminatory*, what nowadays are mentioned as *stop-words*, and words with frequency smaller than D were considered as *irrelevant* for the domain being processed, a kind of odd linguistic event produced by the author. A table with such relevant words, obtained from the original *Scientific American* article is shown in Figure 1.1.

Sentence significance is calculated based on the number and position of relevant word occurrences within the sentence, since positional proximity of relevant words will increase sentence significance. Using this significance document sentences are ranked and the most relevant sentences are selected to form a representative summary, which was called an *automatic abstract* in Luhn (1958). This is nowadays referred to though an extractive summary.

A second early relevant milestone in ATS with many implications throughout the subsequent years, was the work made by Edmundson (1969), which consists of a refinement and a substantial improvement of Luhn's work, by introducing several new document features for extractive summarization. Edmundson (1969) noticed that textual documentation contains several structural key features denoting content relevancy worth to be used for sentence identification and extraction, as for example position of sentences in paragraphs, position of words in sentences, or words occurring in headings, among others. The overall characteristics considered were divided into *positive* and *negative* depending on whether they relate to selected or unselected sentences by human abstractors. Thus the sentence numerical weight is computed based on a linear combination of four machine-recognizable

Significant words in descending order of frequency (common words omitted).

46	*nerve*	*12*	*body*	*6*	*disturbance*	*4*	*accumulate*
40	*chemical*	*12*	*effects*	*6*	*related*	*4*	*balance*
28	*system*	*12*	*electrical*	*5*	*control*	*4*	*block*
22	*communication*	*12*	*mental*	*5*	*diagram*	*4*	*disorders*
19	*adrenalin*	*12*	*messengers*	*5*	*fibers*	*4*	*end*
18	*cell*	*10*	*signals*	*5*	*gland*	*4*	*excitation*
18	*synapse*	*10*	*stimulation*	*5*	*mechanisms*	*4*	*health*
16	*impulses*	*8*	*action*	*5*	*mediators*	*4*	*human*
16	*inhibition*	*8*	*ganglion*	*5*	*organism*	*4*	*outgoing*
15	*brain*	*7*	*animal*	*5*	*produce*	*4*	*reaching*
15	*transmission*	*7*	*blood*	*5*	*regulate*	*4*	*recording*
13	*acetylcholine*	*7*	*drugs*	*5*	*serotonin*	*4*	*release*
13	*experiment*	*7*	*normal*			*4*	*supply*
13	*substances*					*4*	*tranquilizing*

Total word occurrences in the document:	2326
Different words in document:	
Total of different words	741
Less different common words	170
Different non-common words	571
Ratio of all word occurrences to different non-common words	~4:1
Non-common words having a frequency of occurrence of 5 and over:	
Total occurrences	478
Different words	39

Figure 1.1: A table from Luhn (1958) showing statistically significant words automatically selected from a scientific article.

characteristics: number of *cue* words (C), *key* words (K), *title* words (T), and *text location* (L). For a given sentence s, from a text document we have:

$$Weight(s) = \alpha \cdot C(s) + \beta \cdot K(s) + \gamma \cdot T(s) + \delta \cdot L(s) \tag{1.1}$$

with α, β, γ, δ being the feature weighting factors. *Cue* words are non-functional words obtained from a corpus according to predefined constraints based on statistics and also meeting certain linguistic criteria. Three subclasses of *cue* words are obtained: *bonus* words which have frequency values above an L-threshold, *stigma* words with frequency under a predefined U-threshold, and also *irrelevant* words (residue), with frequency value within the $[U, L]$ interval and having dispersion less than another threshold. A total of 783 bonus, 73 stigma, and 139 residue words were obtained. Naturally, bonus words will yield a positive effect while stigma words imply a negative effect in the final calculation of $C(s)$. *Key* words are thematic words, similar to cue words but restricted to the specific document being processed. The concept is similar to the one used by Luhn (1958), identifying high-frequency relevant words in the document. *Title* words are obtained from document skeleton key locations, as the title and headings, and they are considered as positively significant. Finally the location method, also considers documental skeleton properties by looking at their ordinal positions in the text, assuming for example that paragraphs occurring at the beginning and end of a document will contain more key sentences as well as sentences starting and ending paragraphs.

1.1.2 The "Renaissance" in Automatic Summarization

The main concerns of *Automatic Text Summarization* and subsequent research developments in the 1970's and 1980's were to refine the extractive methods proposed by the early original work, previously mentioned, mainly in what is known as the *Edmundsonian extractive paradigm*. In 1993 an historical seminar took place in the Dagstuhl school, a German computer science research center, which is known as the *Dagsthul Seminar* (Jones, 2007), marking the beginning of a new research phase in ATS, a kind of renaissance in the area, envisioning new perspectives and goals, as multi-document summarization and abstractive summarization. Also the 1997 ACL Workshop (ACL-97) on summarization started a succession of workshops in this area, in the series of Document Understanding Conferences (DUC)[1] in the subsequent years, between 2000 and 2007. The driving force in each one of these DUC conferences, was a competition where several summarization tasks were pre-defined and competitors, usually summarization research groups, tried to implement a system able not only to achieve the objectives but also get it done more effectively, ideally winning the competition.

British scientist Karen Spärk Jones was a quite important contributor and promotor of the ATS research filed, in the last 20 year. She had a key role in the early Dagstuhl seminar, brought relevant criticisms and insights into the field, generally pointing directions to what has to be done, in order to turn automatic summarization closer to human performance. Spärk Jones defined automatic summarization as a three-step process.

- **interpretation** - From the input raw text, obtain a structured internal representation of it.
- **transformation** - An operation on the original text representation yielding a simpler/reduced structure which represents the summary structure to be generated.
- **generation** - From the simplified text structure obtained in the previous step, generate the corresponding text summary.

In Jones (1999) it is shown that more effective summarizing systems require a detailed and explicit consideration of what is named as the *context factors*. As it was explained, the performance of an ATS system depends on these factors, and a fair evaluation of any of such systems cannot ignore them. For example the purpose for which the system is supposed to satisfy can widely vary, from a mere set of key information, words or phrases, a rough preview of the text content, to a richer content summary preserving text grammaticality and connectivity. Three main summarization context factors that should be considered in any ATS problem were identified in Jones (1999).

- **Input factors** - These are related with the conditions and features contained in the source text, which fall into three classes: *source form*, *subject type*, and *unit*. The first one tackles the

[1] URL: http://duc.nist.gov/ [June 2010]

questions of text *structure*, *scale* (paragraph, article, book), *medium* and *genre* (a description, or a narrative), while the second class handles the question of considering the purpose and reader's knowledge about the text subject and the world. The third class distinguishes between single and multiple input sources.

- **Purpose factors** - A summary is naturally intended to serve the purposes of a required type, generally to ease user's work. Thus from the same source document several quite different summaries might be produced for different target users. In many cases summarization is an ambiguous process, and only the user's needs permit to define the source content to be highlighted. These factors were considered the most important in ATS and may also fall in three categories: *situation*, the context in which the summary is to be used, *audience*, the class of readers who are going to use the summary, and *use*, which specifies what the summary is going to be used for.

- **Output factors** - These features define the format and content type to be included in the summary, for example the extent to which the summary is intended to capture the whole *source content* or just some key issues. It is also related with *format*, i.e: should it be a normal *running text* or a list of headlines-like items. Another concern here is *style*, which may include the following categories: *informative* (convey what the source text says), *indicative* (identify the main topics present in the text), *critical* (commenting summary key ideas), and *aggregative* (normally, when made from multiple sources).

Given the great difficulty involved in the evaluation of a summarization system, another issue stressed and promoted by Spärck Jones was a correct automation of the evaluation of ATS systems (Jones, 1999, 2007). It was stated that any fair/correct evaluation process is closely tied with the context factors mentioned previously and if they are not carefully observed then erroneous results will be obtained in the comparative studies.

1.2 Motivation and Objectives

Automatic text summarization systems are classified in two main categories: *extractive* and *abstractive* (Hahn & Mani, 2000). This last one is much closer to the kind of summarization made by humans and is naturally much more difficult to automate, since it requires a semantical and even pragmatical understanding of the text and knowledge of the world. In abstractive summarization higher complex linguistic issues are involved, like *negation*, *anaphora resolution*, *paraphrasing*, *text rewriting* and even *rhetorical discourse representation* models (Jones, 1993), which nowadays are still be "hot" research topics by themselves, and the solutions still need to be improved. On the other hand, so far the majority of rich electronic linguistic resources are only available for the English language. Due to such reasons, abstractive summarization has not yet been fully addressed by the research commu-

nity, that naturally decided to start by a relatively simpler and at least implementable approach - the extractive summarization. Whereas in abstractive summarization "deep" linguistic analytical knowledge is involved, extractive summarization relies essentially in statistical and shallow techniques, as these are simpler to implement. The *extractive summarization* category is confined to the question of creating summaries solely by selecting the best textual units from an original text. The degree of textual unit granularity may vary and include phrases, sentences, or even paragraphs. Yet the most used minimum textual unit is the sentence. As in the *Edmundsonian Paradigm* (Edmundson, 1969; Mani, 2001), the whole idea in an extractive system is to find a rank value of any textual unit, based on several predefined features like key and title words, and then select the units having highest rank value and generate the final summary by the concatenation of these. The important difficulties revealed by this approach, include the lack of cohesion and coherence. Besides we can also identify another important limitation, which served as a motivation to start our work, and which is based on the fact that this method is unable to summarize beyond the minimum textual unit considered, in our case the sentence. That is, it is not concerned with the issue of sentence simplification, though many text sentences are worth of it, due to their natural complexity. The tests over corpora we made for the English language, in particular for web news stories, provides evidence that the mean sentence length is equal to 20 tokens. In fact, this gave rise to a new research subfield inside the ATS area, known as *Automatic Sentence Reduction*, or *Compression*, as well as *Automatic Sentence Simplification*. In our case, we use this first nomenclature since in this class sentences are simplified through adequate content removal, while in the latter it is meant to be based on more complex syntactical transformations. As an problem illustration we show below an example of a sentence reduction operation.

> *"In Louisiana, the hurricane landed with wind speeds of about 120 miles per hour and caused severe damage in small coastal centers such as Morgan City, Franklin and New Iberia."*

For this sentence one possible and quite good reduced sentence version can be:

> *"In Louisiana, the hurricane landed and caused severe damage."*

As we can see, the reduced sentence preserves the most relevant information, while at the same time maintains its grammaticality, and the final result is indeed a much simpler sentence. So this was the challenge that started and drove our research work, which is going to be presented throughout this thesis.

During the first decade of the 21st century, research in *Automatic Text Summarization* has been continuing its objective to move from the traditional extractive methodology to a more abstractive resembling approach. In this context, *sentence reduction* has been an emerging field trying

to contribute to this research direction, and as presented in the *related work* (Chapter 2) some approaches have been experimented during this time period, based essentially on two main strategies: *knowledge-driven* systems and supervised *machine learning* techniques. We think that despite their individual virtues, both contain important drawbacks and practical limitations that were not addressed. The former is hugely language-dependent, essentially depending on the availability of rich linguistic knowledge resources, which are so far scarce for many non-english languages. Even with linguistic resources at our disposal, knowledge-driven systems may always be incomplete and require a huge human effort for creation of such resources. The latter strategy of using supervised machine learning, requires a dataset of training examples that must have been manually crafted by humans. So, both ways require considerable amounts of manual intervention, which is subject to incompleteness and imperfection. Our approach aims at following a different strategy, by using minimum linguistic resources and an *unsupervised* machine learning strategy. We intend to minimize human inputs by automatically constructing training examples, just from the analysis of corpora.

1.3 Work Overview

In this work we propose a completely new approach within the *Text Summarization* research field, for the specific task of *Sentence Reduction*, which follows an unsupervised learning methodology in order to induce a set of relevant sentence compression rules, based on shallow linguistic processing. We designed a system for learning sentence simplification rules in a completely autonomous manner which is composed of five main modules, organized in a pipeline scheme as shown in Figure 1.2. The first three are responsible for data extraction and preparation and in the fourth one the induction process involves, giving as a result a set of rules, which might be applied (and evaluated) in other web news, simplifying its sentences, which corresponds to the last module.

Let us analyze this process in more detail. In the first step, web news stories are gathered from related news events, collected on a daily basis. For a given event, for example a terrorist attack, several news stories are collected from different web news sources and their sentences joined in order to discover paraphrases. Existing mathematical functions are explored and new ones are proposed, and their performance is compared. A special relevance is given to a special type of paraphrases, named *asymmetrical* (Definition 2, in Section 3.1), since they contain a variability factor which is important for learning sentence simplification patterns, that is an asymmetry in terms of sentence length. An example of an extracted paraphrase is shown below and more details about this issue are presented in Chapter 3.

In the second module, for every extracted paraphrase from the previous step, word alignment between the paraphrase sentences is performed. These similar-sentence word alignments are likely to reveal local differences and asymmetric regions, among the common ones, indicating possible

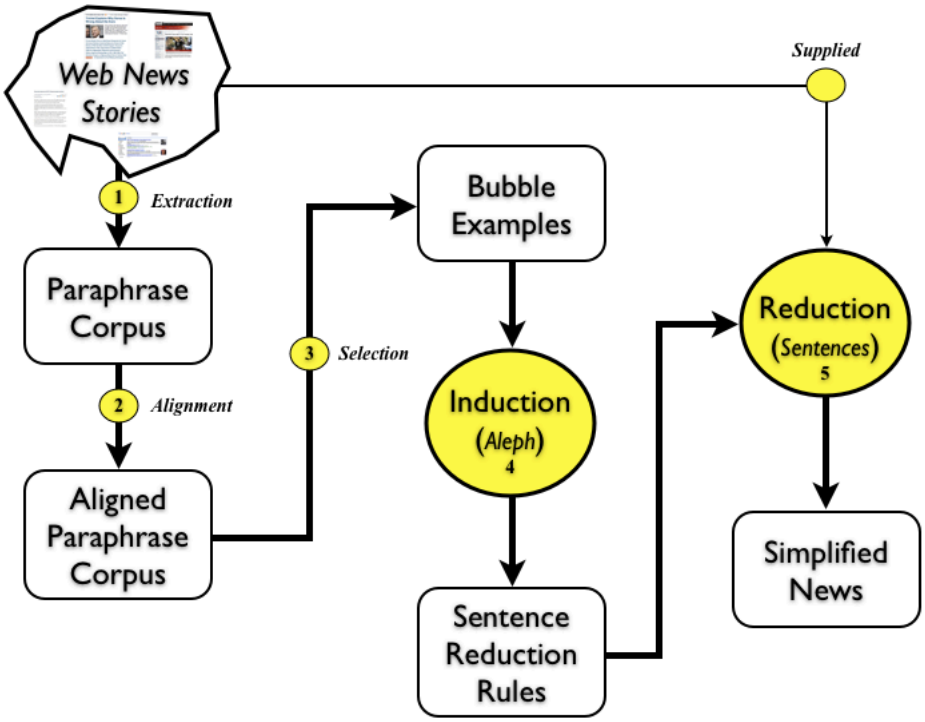

Figure 1.2: The Pipeline Architecture.

S_a: *Police used tear gas to disperse protesters.*
S_b: *Police responded by firing tear gas, sending the protesters scattering.*

Figure 1.3: An example of an asymmetrical paraphrase extracted from related news events.

sentence transformation patterns that may be exploited in learning. Sequence alignment algorithms used for DNA sequence alignments in Bioinformatics were adapted to make the word alignments between our paraphrase sentences. For that purpose, a new dynamic algorithm is proposed in order to select the most suitable alignment algorithm for a given sentence pair. More information about this and their related issues are detailed in Chapter 4. Next we show the alignment obtained for the paraphrase sentences previously shown in Figure 1.3.

```
police _________ __ ______ used tear gas _ _______ to  disperse protesters __________ .
police responded by firing ____ tear gas , sending the ________ protesters scattering .
```

Figure 1.4: An aligned paraphrase sentence pair.

The third step shown in Figure 1.2 aims at looking at these special regions and prepare sets of learning instances to be used in a machine learning environment in order to discover general sentence simplification rules. A special kind of paired-sentence regions are observed, which we call *bubbles*, extracted and conveniently preprocessed to feed the induction process. A complete description of this issue is made in Subsection 6.1, and several examples of *bubbles* are shown there.

The induction process generates sentence reduction rules having the following general structure:

$$|X| = n \wedge L_{cond} \wedge X_{cond} \wedge R_{cond} \Rightarrow eliminate(X) \tag{1.2}$$

This rule scheme means that a sentence segment X of size n may be eliminated if certain conditions hold on the left (L), middle (X) and right (R) segments. In a sentence, the left and right segment contexts are relative to the segment laying in between and called the *kernel* segment. Each segment may in general contain several words. In this work we have decided to maintain the maximum segment size smaller than four, either for contexts and for the kernel, although in the future some size variations may be explored. For the sake of simplicity and compactness, we will omit the consequent rule, since in this rule scheme it will always be equal to "$eliminate(X)$". In order to provide a quick overview and a general flavor of the system output, Figure 1.5 presents three different rules automatically learnt by our system. These rules comply with the rule scheme presented in 1.2 and

$$|X| = 1 \ \wedge \ L_c = \texttt{NP} \ \wedge \ X_1 = \texttt{JJ} \ \wedge \ R_1 = \texttt{IN} \tag{1.3}$$

$$|X| = 2 \ \wedge \ L_1 = and \ \wedge \ X_1 = the \ \wedge \ R_1 = \texttt{JJ} \tag{1.4}$$

$$|X| = 3 \ \wedge \ L_c = \texttt{VP} \ \wedge \ X_1 = \texttt{the} \ \wedge \ R_1 = \texttt{NN} \tag{1.5}$$

Figure 1.5: Sentence Reduction Rules, automatically learned.

they are formed by a conjunction of several literals, setting constraints under which certain sentence subparts may be deleted, thereby transforming the original sentence and giving, as a result, a simplified version of it.

In the previous subset of induced rules, presented in Figure 1.5, the X symbol stands for the candidate segment to be dropped - the kernel segment; the $L_{(\star)}$ and $R_{(\star)}$ are conditions over the left and right context segments, respectively. The numeric subscripts indicate the sentence word position for which a given segment condition applies, and it is always relative to the kernel. The position index starts with 1 and is counted from left to right in the right context R and in the kernel segment X, and from right to left in the left segment L, thus L_1 and X_1 will always represent the neighbor word or *part-of-speech* tag. Such a sentence scheme is exemplified in the next figure, in which we have a real news sentence targeted to be simplified by one of our induced rule.

L_3	L_2	L_1	X_1	X_2	X_3	R_1	R_2	R_3
Congressional	**leaders**	**say**	the	emerging	stimulus	**program**	**could**	**cost**
between	**$400**	**billion**	**and**	**$700**	**billion.**			

Figure 1.6: Sentence schematic division.

The "c" subscript, in the rules in Figure 1.5, indicates a reference to segments with a given syntactic

chunk type. For instance, in the first rule, $L_c = \texttt{NP}$ means that the first chunk on the left of the kernel segment must be a noun phrase. The $|X| = n$ condition defines the kernel length, in terms of the number of words it contains.

In order to illustrate how one of these learned rules can be applied to new raw text, let us for instance consider the rule 1.5 from Figure 1.5. This rule says that a word sequence consisting of three words ($|X| = 3$) will be eliminated if a verb phrase (VP) chunk is observed in the immediate left context ($L_c = \texttt{VP}$), and a noun in the right context, at position one ($R_1 = \texttt{NN}$). This rule also requires that the first word of the sequence being eliminated (the kernel) must be "`the`", as we have $X_1 = \texttt{the}$. This rule is a candidate to be applied to the sentence shown previously in Figure 1.6. To apply a rule to a given sentence we have to mark it with a part-of-speech (POS) tagger and a chunker[1] in order to see whether the rule conditions are satisfied. Note that for POS tagging we are using the *Penn Treebank Tag* set defined in Marcus *et al.* (1993). To help the reader we include a description of the complete 48 tags in Appendix E. So, for the previous example we would obtain the following shallow parsed sentence:

```
[NP congressional/jj leaders/nns] [VP say/vbp] [NP the/dt emerging/vbg stimulus/nn
program/nn] [VP could/md cost/vb] [NP between/in $/$ 400/cd billion/cd and/cc $/$
700/cd billion/cd]
```

In this notation each word is followed by a slash with its POS label and each sentence chunk is delimited by brackets with the syntactic chunk label positioned at its beginning and written in uppercase letters, unlike words and their POS labels, which are written in lowercase. With this notation it turns out evident that rule 1.5 can be applied to this sentence and as a result the word sequence "`the emerging stimulus`" would be delete, giving rise to the following reduced version:

Congressional leaders say program could cost between $400 billion and $700 billion.

This new sentence seems perfectly acceptable, as it preserves the main meaning, and provides a reduction of three words from the original sentence.

Several rules may be applied to a given sentence, and obviously the more rules are applied the more simplified the sentence will end-up. Compositional rule application can be made, if for instance after a certain rule application the sentence is transformed in a new one complying with the conditions of another rule which was not applicable earlier to the original sentence. In particular, we propose a set of rules which allow the application of a simplification of a given sentence. Indeed, some extra constraints must be observed in order to ensure syntactical correctness of the obtained simplified versions. For that purpose a part-of-speech statistical model is employed for checking if the candidate rule would yield a syntactically likely sentence, and if not it will prevent that particular

[1] We used the OpenNLP toolkit in this process. URL: http://opennlp.sourceforge.net/ [November 2010]

rule from firing for that specific case. This process of sentence reduction rules applicability corresponds to the fifth and last module from our system pipeline (Figure 1.6), where new unseen web news documents, supplied by the user, are simplified through the application of previously learned sentence reduction rules. Naturally our focus is the simplification of sentences. However, it could be combined with other already known summarization techniques, in order, for example, to first filter out irrelevant sentences and afterwards apply rule simplifications to remaining sentences in the document.

1.4 Main Scientific Achievements

The research work developed in the course of preparing this thesis gave us the opportunity to achieve several relevant scientific contributions. These were presented at specialized international conferences, and published in the proceedings. We have scientific publications in each one of our three main work components: *extraction*, *alignment*, and *induction*. A brief synthesis of the set of publications, is given here together with a brief description and explanation of each one. The list follows the chronological order of the events.

In the field of *paraphrase identification* and *extraction* from text corpora, we have three publications:

- Cordeiro *et al.* (2007b) (*IEEE - ICCGI 2007*) - This article presents the *Sumo* metric as a better function for paraphrase identification, providing the results of an experimental comparison between this function and four other functions used by others for the same task. The Sumo function is presented in Subsection 3.3.1.

- Cordeiro *et al.* (2007a), (*AAAI - FLAIRS 2007*), DBLP - This contribution is a refinement of the previous one, in which some aspects are better clarified. It also contains an experiment carried out with paraphrase clustering by using the *Sumo* function as a similarity metric between two sentences. This issue is discussed in Section 3.4.

- Cordeiro *et al.* (2007c) (*Journal of Software 2007*), DBLP - Our third publication starts with our *Sumo* function and goes a bit further by proposing a new family of functions sharing several common characteristics ideal for asymmetrical paraphrase identification in corpora. This work contains also a comparative study with the conventional existing functions for shallow symmetrical paraphrase identification. The details of this contribution can be found in Section 3.3.

Related with the topic of *word alignments* in paraphrasic sentences, described in Chapter 4, we have one contribution:

- Cordeiro *et al.* (2007d) (*ACL 2007* workshop) - This article is a concise introduction to the issues presented in Chapter 4, related to the alignment of words between paraphrasic sentences.

Our last publication within the scope of this thesis, is related with the induction process, described in Chapter 6:

- Cordeiro *et al.* (2009) (*ACL 2009* workshop) - This article describes how *Inductive Logic Programming* is used for learning sentence reduction rules, from our corpus of aligned paraphrases. This work was presented in an ACL[1] workshop specially dedicated to the topic of automatic language generation and summarization.

More recently, the work contained in the previously mentioned contributions[2] has been integrated in new research tasks, which go beyond the aim of this thesis. In collaboration with other international researchers, we have produced two publications on the topic of *automatic word semantic relation discovery from corpora*:

- Grigonyté *et al.* (2010) (*COLING*[3] *2010*) - This work uses an adapted version of the *Sumo* metric and of the alignment algorithms in order to automatically discover synonyms from medical corpora.
- Dias *et al.* (2010) (JNLE[4] 2010) - This work explores aligned paraphrase corpora to automatically discover word semantic relations.

1.5 Thesis Plan

This thesis started by giving a short introduction to the area of *Automatic Text Summarization*, in which our work is integrated. This was followed by a presentation of our work main motivations and objectives. After, a work overview about sentence reduction was supplied to give the reader a quick introduction about the research work developed. The work is composed by three main research components, which are investigated separately and then adequately combined in order to produce results to fulfill our initial objectives. These areas are:

1. *Automatic paraphrase extraction from corpora,*
2. *Automatic paraphrase word alignment,*
3. *Sentence reduction rule induction, from aligned paraphrases, and application of rules.*

[1] ACL - *Association for Computational Linguistics.*
[2] In particular the work related with paraphrase detection and alignment.
[3] COLING - *Computational Linguistics*
[4] JNLE - *Journal of Natural Language Engineering*

The last one is certainly the most relevant of our research goals, but its strength is also dependent on the data automatically collected in the first two components. In each component new solutions were investigated and proposed. Therefore we dedicate one chapter for each one of these three components.

In Chapter 2 we present the relevant related work in the area of *sentence reduction*, giving detailed description and also addressing some critical aspects by indicating strengths and difficulties of each approach.

In Chapter 3 we address the problem of automatic paraphrase extraction from corpora, mainly focused on using shallow techniques. Some already existing functions are presented and new ones are proposed, together with justifications favoring the use of these last ones.

The issue of automatic paraphrase word alignment is discussed in Chapter 4. Optimal global and local sequence alignment algorithms are detailed as well as their technical adaptation to our problem of word alignment between paraphrase sentence pairs. Even here, in a well studied area having mature procedures and algorithms established, mainly from *bioinformatics*, we propose new solutions, relevant for the particular problem faced.

Some theoretical foundations like *First Order Logic* and *Inductive Logic Programming*, as well as the induction tools used, are presented in Chapter 5, before giving a complete description of our learning process in Chapter 6. The reader with experience in those areas may skip Chapter 5.

In Chapter 7, we provide detailed experimental results obtained from every relevant issues developed and presented throughout this thesis and finally Chapter 8 presents the main conclusions of our work and points out several future directions of research worth to be followed.

Chapter 2

Related Work in Sentence Reduction

"*No man is an island, entire of itself ...*"

John Donne (1572-1631)

Sentence reduction[1] has been an active subject of research during this decade. A set of approaches involving linguistic knowledge, *machine learning* algorithms and statistical models has been investigated, employed, and discussed in literature. Throughout this chapter we present various relevant approaches found in the literature that address the issue of sentence reduction. We see this as a promising area for enhancing the quality of automatic text summarization. However, so far, we have not found much research work on *sentence reduction*. It is still an area with many issues to be explored.

We noticed that the main approaches made to this area lie in three different sub-areas, in terms of the main methodology employed. These can be labeled as *translation-based*, *heuristic-based*, and *learning-based* methods. In the approaches inspired by *automatic machine translation*, the problem of sentence reduction is interpreted as a kind of translation task, from a "source language" of expanded sentences (input sentences) to a target language of their possible "compressed versions" (output sentences). Section 2.1 addresses the *translation-based* approach. The *heuristic-based* approaches use rich linguistic knowledge resources available electronically. These approaches are presented in Section 2.2. The approaches based on *machine learning*, presented in Section 2.3, use *supervised learning* methods, in which a set of sentence pairs is used to train a system for learning sentence reduction patterns. Each pair consists of a sentence accompanied by its reduced version. Some systems employ techniques from more than one of these three categories, for example exploit *machine learning* and also linguistic knowledge.

Our approach is machine learning based. It differs from other existing systems in that it extracts and formulates the required learning instances, from *web news stories* text, without any human intervention.

[1] In the literature it is also referred as *sentence compression*, *sentence summarization*, or even as *sentence simplification*.

2.1 Automatic Translation Methodologies

Historically, *Automatic Machine Translation* (AMT) was the main motivation leading to the creation of the *Natural Language Processing* (NLP) discipline, back in 1950. Therefore a wide range of research efforts have been dedicated to this throughout the subsequent decades, pursuing the goals of AMT, which originally seemed quite reachable, but has revealed itself as a very hard problem with many open questions to be solved even in the XXI century. Naturally a huge set of methods, algorithms, and approaches developed in the AMT field have proved themselves to be useful in other NLP fields, like *automatic text summarization* (ATS). In fact original ATS approaches began to appear roughly in the second half of the 1990 decade. Many researchers were been working in AMT before realized that their techniques could naturally be adapted to ATS. Therefore, several AMT approaches serve as a basis for summarization. This includes in particular the sentence reduction problem, representing a simplified version of a translation task, from a source language to a target language. In this section we discuss several earlier sentence reduction approaches, mainly inspired by this AMT paradigm, as well as some more recent ones which continue this tradition.

2.1.1 Origins - Pioneering Projects

One of the earliest work starting the sentence reduction field was carried out by Chandrasekar & Srinivas (1997) which aimed at simplifying long sentences so as to serve several practical purposes, among those, to ease machine translation. The whole idea was to transform a long and complicated sentence into two or more shorter sentences and consequently simplify the process of automatic machine translation. The following example was retrieved from the original article and included here for illustration.

> *Talwinder Singh, who masterminded the Kanishka crash in 1984, was killed in a fierce two-hour encounter.*

The previous sentence would be split into the two following sentences:

> *Talwinder Singh was killed in a fierce two-hour encounter.*
> *Talwinder Singh masterminded the Kanishka crash in 1984.*

This method of long sentence splitting was described as a two stage process. The first one provides a structural representation of the sentence and the second one applies a sequence of rules to identify and extract the components that can be simplified. Special sentence splitting points ("articulation-points") such as phrasal extremes, punctuation, subordinate/coordinate conjunctions and relative pronouns are identified, based on predefined rules and grammatical formalisms.

To avoid full sentence parsing, two simpler formalisms were experimented and compared to generate sentence structural representations: the Finite State Grammar (FSG) and a Dependency Model (DM). The first one is simpler than the second one, yielding sentence phrase chunks, while the DM formalism uses partial parsing and is based on a dependency representation provided by a Lexicalized Tree Adjoining Grammar (LTAG). An example of a sentence simplification rule, retrieved from the original article is shown below:

$$\texttt{X:NP, RelPron Y, Z} \quad \rightarrow \quad \texttt{X:NP Z. X:NP Y.}$$

This rule states that a sentence segment composed by a noun phrase `X` followed by a relative pronoun `Y` and some other subsequence `Z`, will be transformed in two sentences starting with the same noun phrase `X` and the first one ending with sequence `Z`, and the other ending with `Y`.

The evaluation of this work was rather too simple from our current perspective, sixteen years later. However, as far as we can tell it was a pioneering work in the field, if not the first attempt in sentence simplification. Using a corpus of newswire data and only considering relative clauses and appositive simplification, 25 out of 28 relative clauses and 14 out of 14 appositives were correctly recovered, by using the DM formalism, whereas with the FSG only 17 relative clauses and 3 appositives were recovered. However, this work is exclusively based on a set of handcrafted rules.

Another pioneering approach in sentence reduction was the work by Grefenstette (1998) aiming at simplifying text to provide audio scanning service for blind users. The major concern of this work was not the generation of grammatically correct reduced sentences, but rather to provide relevant sentence word sequences (telegraphic type). The aim was to let the targeted users to obtain a quick preview of the text and help them to decide whether they would want to hear the original full text. The input sentences are transformed by using a part-of-speech tagger which also groups words into chunks and marks them with group markers identifying verb and noun groups. For each group, special component types are identified, like for example "Head of Passive Verb Group", "Head of Infinitive Verb Group", "FreeNoun" or "PrepNoun". By including a set of filters the system can also identify syntactic dependencies, such as *subject-verb*, *verb-indirect objects*, etc. Afterwards the groups and annotated text are transformed using a process which eliminates words depending on the level of reduction specified according to one of the eight predefined levels, which are shown below. In each case the rule describes which items are kept. The rest is eliminated.

1. Keep only proper names, no subclauses;
2. Keep only subjects and object nouns, no subclauses;
3. Keep only subjects, head verbs, and object nouns, no subclauses;
4. Keep only subjects, head verbs, and object nouns, preposition and dependent noun heads, no subclauses;
5. Keep only proper names, including subclauses;

6. Keep only subjects and object nouns, including subclauses;
7. Keep only subjects, head verbs, and object nouns, including subclauses;
8. Keep only subjects, head verbs, and object nouns, preposition and dependent noun heads, including subclauses.

For example, level eighth is somewhat similar to the process of stop-word elimination from text. It has a relatively minor impact on text reduction. The generated telegraphic text version could feed a speech synthesizer and fulfill the system design main objective. An example from the original article is included below, where the input sentence is:

> *Former Democratic National Committee finance director Richard Sullivan faced more pointed questioning from Republicans during his second day on the witness stand in the Senate□s fund-raising investigation. While he grew flustered at times, the 33-year-old Sullivan stayed tight-lipped, downplaying concerns about former fund-raiser John Huang.*

The generated reductions for each one of the eight predefined levels would be:

1. Richard Sullivan Republicans Senate Sullivan John Huang.
2. Richard Sullivan pointed questioning. Sullivan tight-lipped concerns.
3. Richard Sullivan faced pointed questioning. Sullivan stayed tight-lipped concerns.
4. Richard Sullivan faced pointed questioning from Republicans during day on stand in Senate fund-raising investigation. Sullivan stayed tight-lipped concerns about John Huang.
5. Richard Sullivan Republicans Senate Sullivan John Huang.
6. Richard Sullivan pointed questioning. Sullivan tight-lipped concerns.
7. Richard Sullivan faced pointed questioning from Republicans during day on stand in Senate fund-raising investigation, he flustered, Sullivan stayed tight-lipped concerns about John Huang.
8. Richard Sullivan faced pointed questioning from Republicans during day on stand in Senate fund-raising investigation, he flustered at times Sullivan stayed tight-lipped concerns about John Huang.

The work of Witbrock & Mittal (1999) used more sophisticated approach for this task. It was also inspired by previous work[1] in machine translation, and has already employed *machine learning* to *train* their language models. Here the summarization task is described as the process of translation from a verbose (source) language into a succinct (target) language. It was claimed that it is simpler than translation because here we do not have to capture every sense and nuances of the source document. This work only aims at generating very small summaries, such as just a headline, representing the original document. This system may be trained by feeding it with a set of $\langle document, summary \rangle$

[1] Essentially the IBM CANDIDE system.

pairs that are used to learn translation mappings between the source and the target formulations. During the training process the system constructs a statistical model, representing a mapping between the two formulations. The training process is subdivided in two main tasks named as: *content selection* and *surface realization*. The first one models the likelihood for a given word from the source document to be present in the target document (summary):

$$P(\texttt{word} \in sumary \mid \texttt{word} \in document) \tag{2.1}$$

The second task aims at producing likely sequences of words in the summary, based on a language bigram model. The overall likelihood of a word sequence with n tokens $S = [w_1, w_2, ..., w_n]$ is calculated by multiplying the sequence of their bigram probabilities:

$$likelihood(S) = P(w_1) * \prod_{i=2}^{n} P(w_i \mid w_{i-1}) \tag{2.2}$$

Actually the log-probability (LP) is taken, as is usual in such cases. For unseen bigrams the back-off discount method (Katz, 1987) is used. For example the probability of the sequence "*Details of sexual affair*" would be equal to:

$$LP(details) + LP(of|details) + LP(sexual|of) + LP(affair|sexual) \tag{2.3}$$

These two models are combined in a single one, in order to rank candidate summaries which can meet certain constraints, like for example compression rate[1]. This combination is made simply by weighted sum of the two model probabilities from equations 2.1 and 2.2. In our previous example we would have:

$$\begin{aligned} &\alpha * \left[LP_{cond}(details) + LP_{cond}(of) + LP_{cond}(sexual) + LP_c(affair)\right] + \\ &\beta * \left[LP(details) + LP(of|details) + LP(sexual|of) + LP(affair|sexual)\right] \end{aligned} \tag{2.4}$$

where $LP_{cond}(\texttt{word})$ means $\log\left[P(\texttt{word} \in sumary \mid \texttt{word} \in document)\right]$, and α and β are weights equal to one.

The application of these models for sentence compression on new text rises the issue of searching good combinations in order to generate good sentence compression. That is, choosing the sequence of words that maximizes the probability of the model combination, as in the example shown in 2.4. Thus, a first order Markov model was used and a Viterbi beam search algorithm (Forney & David, 1973) to efficiently find a sub-optimal summary.

To evaluate the system, a set of $\langle document,\ summary \rangle$ pairs were extracted from the *Reuters* news agency, where each "*summary*" is just the "*document*" (news) headline. The evaluation procedure consisted in simply counting term overlaps between the generated headlines and their corresponding original data.

[1]This expression means the size of the summary over the size of the original document.

2.1.2 Using an HMM for Optimizing Translation Templates

Another work in the field of sentence reduction, highly influenced by the AMT area, was made by Nguyen *et al.* (2004). First, they adapted a *translation-template learning* (TTL) method, used in the *example-based machine translation* (EBMT) framework, to the sentence reduction problem. To avoid the problem of complexity that arises in rule combinations, they proposed an optimization based on a *Hidden Markov Model* (HMM), in order to efficiently find the best template reduction rule combinations for a given sentence.

The work on EBMT was initiated by Nagao (1984) and was one of the main approaches to AMT based on corpora. Within this framework, the TTL method (Güvenir & Cicekli, 1998) has been successfully applied in learning translation examples among two languages, for example, from English to Turkish. Therefore in Nguyen *et al.* (2004) sentence reduction is interpreted as a kind of translation task, from a sentence source language to a target language of reduced sentences, in order to apply this TTL method to sentence reduction. This method learns a set of reduction rules, referred to as *template-reduction* rules, from a set of learning examples, which are pairs of long and reduced sentences $\langle \sigma_s, \sigma_t \rangle$[1], obtained from a parallel corpus. A *template-reduction* rule (TRR) is a structure defined as:

$$S_1\, S_2 \ldots S_N \;\leftrightarrow\; T_1\, T_2 \ldots T_K$$

where each S_i and T_j are either constants or variables in the source and target languages, respectively. One example is shown in Figure 2.1. In TRRs, a constant might be a word or a phrase and a

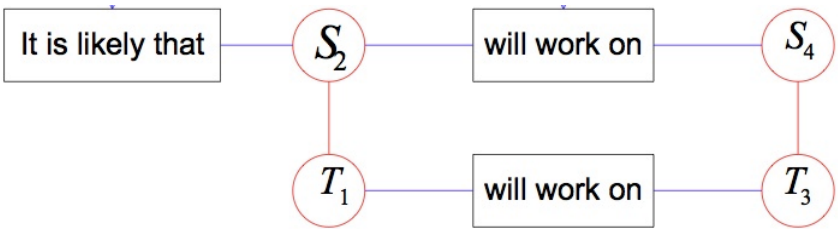

Figure 2.1: An example of a learned template-translation, in the context of sentence reduction. It is assumed that T_1 is in some sense a shorter version of S_2.

variable, like those S_2, T_1, S_4, and T_3 in Figure 2.1, might be substituted by a constant. A TRR without any variable is called a lexical rule, and these may substitute variables in other template-reduction rules. For example in a system having already the rule shown in Figure 2.1 and the following two lexical rules

$$\text{both companies} \;\leftrightarrow\; \text{companies}$$
$$\text{high performance computing} \;\leftrightarrow\; \text{computing}$$

it would reduce the sentence

[1] The *s* and *t* indexes mean respectively *source* and *target*.

It is likely that both companies will work on high performance computing

into

Companies will work on computing

just by combining the TRR with the two lexical rules, by substituting the corresponding variables.

To obtain a template-reduction rule as the one shown previously the template learning algorithm requires two similar reduction cases in the training corpus of $\langle \sigma_s, \sigma_t \rangle$ sentences. For example, the following two training examples

σ_s^1 : *The document is very good and includes a tutorial to get you started*

σ_t^1 : *Document is very good.*

and

σ_s^2 : *This paper is very good and includes a tutorial to get you started*

σ_t^2 : *Paper is very good.*

will give rise to the TRR shown in Figure 2.2. The details about this generalization are explained in

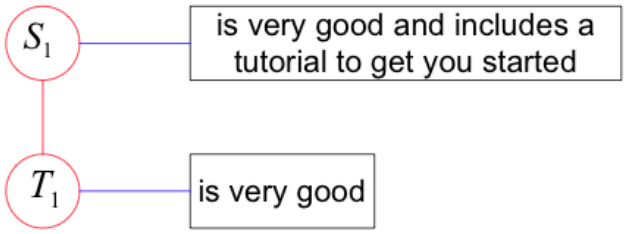

Figure 2.2: An example of a template reduction rule, learned from two similar cases.

Nguyen *et al.* (2004). Hence, this system performs a generalization for two similar learning cases encountered in the training data. Such generalization policy may yield a high number of TRRs, and in our opinion of low quality, heavily affected by the variability or noise in the corpora. This issue will be explained in more detail in Chapter 8, discussing our work.

The major difficulty of this work is its huge computational complexity, arising with even small learning datasets. The problem arises due to the possibility of combining lexical rules with any TRR variable. For instance, in the experiments reported in the author's article (section 4), 1500 $\langle \sigma_s, \sigma_t \rangle$ pairs gave rise to 11034 TRRs and 2297 lexical rules. For a new sentence with a matching TRR having n variables and k matching lexical rules, a brute-force approach would have to explore k^n reduction combinations! For example, let us assume that the system has the TRR shown in Figure 2.1, together

with the following six lexical rules shown below. Let us further assume that it is seeking the best strategy to reduce the following sentence

> *It is likely that two companies will work on integrating multimedia with database technologies.*

For the first TRR aligned variable pair (S_2, T_1) one gets:

$$\begin{aligned}
\text{two companies} &\leftrightarrow \text{two companies} \\
\text{two companies} &\leftrightarrow \text{companies} \\
\text{two companies} &\leftrightarrow \emptyset
\end{aligned}$$

For the second TRR aligned variable pair (S_4, T_3) one gets:

$$\begin{aligned}
\text{integrating multimedia with database technologies} &\leftrightarrow \texttt{<unchanged>} \\
\text{integrating multimedia with database technologies} &\leftrightarrow \text{multimedia} \\
\text{integrating multimedia with database technologies} &\leftrightarrow \text{dayabase technologies}
\end{aligned}$$

In this case one would end up with a total of $9 = 3^2$ combinations. To overcome this complexity barrier, the authors implemented a *Hidden Markov Model* (HMM) to efficiently compute the most likely sequence of lexical rules to be combined, in a given sentence matched to TRR. For this problem, the lexical-rules are the model hidden states and the probabilities for the HMM are trained using the corpus of $\langle \sigma_s, \sigma_t \rangle$ sentences through the *Baum-Welsh* learning algorithm (Baum & Eagon, 1967). More details about it may be found in Nguyen *et al.* (2004). The introduction of this HMM technique allowed to lower the previously mention complexity factor, from $O(k^n)$ to $O(n * k^2)$, thanks partly to the use of the *Viterbi* algorithm (Viterbi, 1967).

The evaluation of this system was performed using a set of sentence pairs of long and reduced sentences, obtained from a vietnamese news agency, where 1500 pairs were manually corrected and used to generate the TRR and lexical rules. Afterwards additional 1200 sentence pairs were used, from which 32 pairs were randomly selected for testing and the remaining for HMM parameters estimation.

The authors described a similar evaluation procedure as used by Knight & Marcu (2002) and claimed that the system achieved a human comparable performance. However some key issues were not well clarified, as for example the kind of manual corrections needed for the 1500 sentence set extracted. Moreover, since they claimed that the algorithms could easily be adapted for any language, why did they not test the system with the English language? In particular with the dataset used by Knight & Marcu (2002), since they made several qualitative comparisons of their work with this system. We

suppose that the main reason for that is because their induced templates are weak generalizations, learned from a very few examples, leading to a set of lexical rules with low support, unlike in AMT, where this technique has been used for induction of translation templates, where a huge set of examples, obtained from parallel corpora, is usually employed. In our work, we have conducted an experiment having several fundamental similarities with this approach and the results obtained were in fact much worse when compared to the performance obtained when rules have a higher generalization power. In our case we use a special type of learning instances, named *bubbles* (Section 6.1) to feed an induction engine and obtain more general sentence reduction rules. Our baseline experience was to employ the set of *bubbles* directly as being sentence reduction rules, since they represent such cases, and the obtained results are clearly much lower (see line **BL** in Table 7.10), when compared with more general reduction rules, obtained after the induction process.

2.2 Heuristic Based Methodologies

In this section we present methodologies which are mainly based on rich language knowledge, as well as specific heuristics designed for sentence reduction, like for example a set of handcrafted rules. These approaches may also employ other methodologies, like machine learning, although their main force relies mainly on supplied linguistic knowledge.

2.2.1 Using Rich Linguistic Knowledge

The work of Jing (2000) presents a sentence reduction system for automatically removing extraneous phrases[1] by combining multiple resources of knowledge in order to decide which are the most appropriate sentence phrases for deletion. As in the majority of work in this field, the main objective is to eliminate or reduce sentences as much as possible, while maintaining their grammatical correctness.

Six major operations that can be used were identified in a previous work (Jing & McKeown, 2000). These are:

1. remove extraneous phrases,
2. combine reduced sentences,
3. syntactic transformations,
4. substitute phrases by their paraphrases,
5. substitute phrases with more general or specific descriptions,
6. reorder extracted sentences.

[1] A *phrase* is a group of words functioning as a single unit in the syntax of a sentence, for example a *noun phrase*, or a *verb phrase*.

Naturally, for sentence reduction the most effective from is removal of extraneous phrases, which was named as *sentence reduction*.

The implemented system operates by using multiple sources of knowledge to make decisions concerning sentence reduction, which include lexical, syntactical, and corpus computed statistics. Basically four linguistic resources were employed, which are summarized in the following list.

- **Corpus** - From a collection of news documents and their corresponding human-written abstracts, a set of 500 sentence pairs were extracted, where for each pair of sentences of the form $\langle s_{original}, s_{reduc} \rangle$, the sentence $s_{original}$ came from the original article and s_{reduc} from the human-written abstract.

- **Lexicon** - A combination of lexical resources, including the computational lexicon COMPLEX (Grishman *et al.*, 1994), which provides a detailed syntactic information for 380k English head words; English verb classes and alternations (Levin, 1993), which studies and defines syntactic and semantic verb proprieties; The *WordNet* lexical database (Miller, 1995) and the *Brown Corpus* tagged with *WordNet* senses. The lexicon includes subcategorizations for over 5000 verbs and was used to identify the obligatory arguments of verb phrases.

- **WordNet** (Miller, 1995) - Provides information on lexical relations: synonym, antonym, meronym, entailment and causation. These lexical links are used to identify the focus in the local context.

- **Syntactic Parser** - The *IBM English Slot Parser* (McCord, 1990), for sentence full parsing including thematic role identification (e.g. subject, object).

These elements are used first to produce a full parse tree, then to identify the critical undeletable sentence components for ensuring grammatical correctness as well as rule exceptions for those sentence components that are usually deleted, like *prepositional* and *to-infinitive* phrases, as well as the *adjectives*.

Afterwards the system computes a score for each sentence phrase, which represents the amount of relatedness to the main topic. This computation is performed by weighting WordNet lexical relations of each phrase word to the main topic, through the function shown in Equation 2.5.

$$ContextWeight(word) = \sum_{i=1}^{9} L_i * NUM_i(word) \tag{2.5}$$

In this function nine types of lexical relations are considered and each one is represented by the i index. The function $NUM_i(word)$ counts the number of i type lexical relations between the $word$ and any other word in the phrase (context), and L_i is a weighting factor for that connection type. The phrase score is calculated by adding all weighted lexical contextual relations.

The last procedure performed by the system is the computation of the likeliness of sentence subtree removal, from the training data of $\langle s_{original}, s_{reduc} \rangle$ sentence pairs, ending up with a set of phrase removal probabilities, such as:

$$Prob\{remove(\texttt{when-clause}) \mid \texttt{verb} = "give"\}$$

which in this case represents the probability of removing a "`when-clause`", when the main verb is "*give*".

This last system component was the only one independent from linguistic knowledge and it aims at modeling how likely humans delete certain phrases from sentences though for probabilistic models. In our view a training set of 500 sentence pairs seems really insufficient. Moreover, the paper does not present a detailed description or algorithm showing how the final reduction decisions are carried out and generally the only thing mentioned is that the system will decide to remove a sentence subtree if it is not grammatically obligatory, if it is not the focus of the local context, and if it has a "reasonable probability" of being removed by humans. However, detailed decision parameters and procedures are omitted. For example, what is the level of the "reasonable probability" just mentioned? In terms of evaluation the authors claim that 81.3% of reduction decisions made by the system agreed with those of humans, though no evaluation concerning grammatical correctness was performed.

This system is hugely dependent on the availability of all rich language knowledge resources and therefore can hardly be adapted to non-english languages, most of them with very scarce computational linguistic tools.

2.2.2 As an Optimization Problem

Recently a different approach to the *sentence reduction* problem was presented as an *optimization* problem, where the goal is to find the optimal reduced sentence version satisfying a set of *integer programming* constraints (Clarke & Lapata, 2006). In this work a sentence S is characterized as a sequence of n words ($S = \langle w_1, w_2, ..., w_n \rangle$) and the space of all possible reductions contains all the 2^n reduced versions of S, obtained by word elimination. As in any optimization problem, a set of constraints guides the search through the narrowing subset of valid solutions (here reductions), until the optimum is reached. The authors codify a set of linguistic constraints as *linear inequalities*, in order to ensure the structural and semantical validity for the generated reductions. Indeed, *integer programming* technique was used in which solutions must be linear combinations of integer

quantities. For example their objective function z is given by

$$\begin{aligned}\text{maximize } z \quad &= \sum_{i=1}^{n} y_i \cdot P(w_i) \\ &+ \sum_{i=1}^{n} p_i \cdot P(w_i|start) \\ &+ \sum_{i=1}^{n-2} \sum_{j=i+1}^{n-1} \sum_{k=j+1}^{n} x_{ijk} \cdot P(w_k|w_i, w_j) \\ &+ \sum_{i=0}^{n-1} \sum_{j=i+1}^{n} q_{ij} \cdot P(end|w_i, w_j)\end{aligned}$$

subject to:

$$y_i, p_i, q_{ij}, x_{ijk} = 0 \; or \; 1 \tag{2.6}$$

where $P(\cdot)$ represents *n-gram* models. In particular *unigram*, *bigram*, and *trigram* models were employed, as well as a special *unigram* model for sentence start and a *bigram* for sentence termination. Thus parameters $y_i, p_i, q_{ij}, x_{ijk}$ are binary functions indicating whether a given *n-gram* probability counts or not for the final scoring function.

Afterwards *sequential* and *linguistic* constraints were defined, which encode a set of language knowledge to guide the search for a useful, correct, and relevant (likely) solution which naturally in this case will be a reduced sentence version from the original one. Naturally, the aim is to reduce the search space as much as possible.

The sequential constraints define some sentence positional restrictions, like for example, a condition that *only one word can start a sentence*, through the equation

$$\sum_{i=1}^{n} p_i = 1 \tag{2.7}$$

or a more complex one stating that if a word is included in a sentence then it must either start it, or be preceded by two words or one other word and the start token "*start*" through the following condition

$$\forall k : k \in \{1, ..., n\} \qquad y_k - p_k - \sum_{i=0}^{k-2} \sum_{j=1}^{k-1} x_{ijk} = 0 \; . \tag{2.8}$$

Overall five sequential constraints were defined. In addition to the linguistic constraints encoding grammatical conditions that must be preserved for the generated reduced sentences, like **modifier constraints** ensuring that head words and their modifiers must still be connected in the reduction, **sentential constraints** express some general intuitions, like one stating that at least one verb must be included in the reduction as shown in Equation 2.9.

$$\sum_{i \in verbs}^{n} y_i \geqslant 1 \tag{2.9}$$

Other constraints state that if a verb is present in a sentence so must be its arguments and vice-versa. Coordination is also handled in this constraint category, i.e. if two head words are conjoined in the

original sentence, then if they are included in the compression the coordinating conjunction must also be included. The third and last linguistic constraint category - **compression-related constraints** - impose hard restrictions on the reduction output, by rejecting any sequence within brackets, forcing personal pronouns to be included and forcing minimum sentence length. Details about such constraints definitions are in the original paper.

Aiming at having relevant content words in the reduction, the first component in the objective function (Equation 2.6) was replaced by a significance score, and so a modified unigram model was utilized, replacing $\sum_{i=1}^{n} y_i \cdot P(w_i)$ by

$$I(w_i) = \frac{l}{N} \cdot f_i \cdot log\frac{F_a}{F_i} \tag{2.10}$$

where f_i and F_i are the frequency of w_i in the document and corpus, respectively, and F_a is the sum of all topic words in the corpus. Considering the sentence parse tree, l is the number of clause constituents above w_i, and N is the deepest level of embedding.

Regards results, the authors claim comparability with their current state-of-art, namely the work of Knight & Marcu (2002) (see Subsection 2.3.1) and followed a similar evaluation method, though using a greater number of human evaluators. Although no parallel corpus was necessary to train the system, it is hugely knowledge-driven and generally can be described as an approach that makes sentence reduction through a human supplied rule set, codified as constraints in the framework of *integer linear programming*. This approach is not easily transportable to other languages, since a redefinition of handcrafted linguistic knowledge is necessary. Furthermore it fails to cover a variety of linguistic phenomena involved in the sentence reduction process.

2.3 Machine Learning Methodologies

In the last twenty years, a great variety of AI problems have been successfully solved by applying *machine learning* (ML) techniques. The fundamental idea behind is to have trainable systems capable of being dynamically adjusted as needed, by obtaining the information from the surrounding environment. Nowadays one can find ML solutions in almost all NLP main subareas, ranging from *machine translation* to *automatic text generation*, and including naturally *automatic text summarization*.

2.3.1 Training a Noisy-Channel Model

The work of Knight & Marcu (2002) is an interesting example of a method that involves applying *machine learning* to the sentence reduction task. This work expresses the sentence reduction problem by following two different directions. The first one, referred to as the *noisy-channel model*, is based on a *bayesian learning model*, borrowed from the coding and communication theory. The second one a *decision tree learning* method that induces a set of sentence reduction rules in a form of sequences

of syntactic transformations.

The fist approach was inspired by a quite "old" NLP metaphor introduced by the well-known information theorist Claude Shannon around 1960 - the *noisy channel model* and *decoding*, in the context of language transmission through media, like communication channels and speech acoustics (Jurafsky & Martin, 2000). The general idea consists of looking at the sentence reduction problem as a message transmission phenomenon, from a source space of reduced sentence versions to a target space of expanded versions, where the extra information responsible for this expansion is considered to come from some sort of random effect which superfluously inflates the original pure and concise sentence. Therefore the main task of this approach consists in learning a new model - the *decoder* - from training examples of sentence pairs, in order to decode the expanded sentences back to their reduced versions.

In this sentence transmission adaptation of the *noisy-channel* model there are two sentences involved `s` and `t`, standing for the compressed and expanded sentence versions respectively, as illustrated in Figure 2.3. The transmission process adds some noise to `s`, some spurious words, and as a result

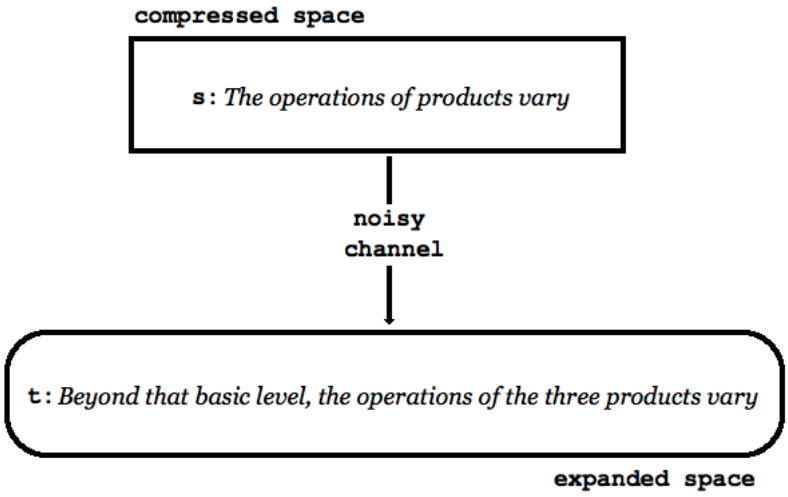

Figure 2.3: The noisy channel model for sentence transmission.

an expanded sentence version is obtained from the transmission output. The goal of the sentence reduction system is to recover the most likely original message from their noisy, expanded version t. In order to do that the process is normally subdivided into three steps, common in any noisy channel problem.

- The **source model**: Where the probabilities of the source sentences, $P(s)$, are calculated and estimated from corpora, hence preserving certain language principles, as low value for ungrammatical sequences of words.

- The **channel model**: It calculates the probability that sentence s is transformed into sentence t through the noisy communication channel, which can be expressed as $P(t \mid s)$, that is, a probability of t given s. This information is obtained from the *training process*, by looking at a set of n training examples in the form of $\{\langle s_1, t_1\rangle, ..., \langle s_n, t_n\rangle\}$.

- The **decoder**: This model applies the previous models, estimated during the training process, to compute the most likely source sentence (reduced sentence version), for a particular new assumed expanded sentence t. This is done by searching the sentence s that maximizes $P(s \mid t)$, through the usually used bayesian equivalent form:

$$\underset{s}{argmax}\ \ P(s \mid t) = \underset{s}{argmax}\ \ P(t \mid s) * P(s) \tag{2.11}$$

To compute the **source model** probabilities, aiming at grammatical correctness and language word sequence likelihood, a combination of a *Probabilistic Context Free Grammar* (PCFG) score and a standard word-bigram score is employed. First, the sentences are fully parsed through Collin's parser

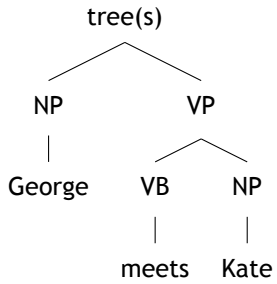

Figure 2.4: The parse tree for the "*George meets Kate*" sentence.

(Collins, 1997) and then the model probabilities are calculated from the parsed trees. For example, to calculate the source probability of the sentence `s` = "*George meets Kate*", `tree(s)` will be taken (see Figure 2.4) and its PCFG computed as shown below, where each conditional probability is estimated from language corpora.

$$\begin{aligned} P\{tree(s)\} = \ & P_{cfg}\{\texttt{TOP --> S} \mid \texttt{TOP}\} * P_{cfg}\{\texttt{S --> NP VP} \mid \texttt{S}\} * \\ & P_{cfg}\{\texttt{NP --> } George \mid \texttt{NP}\} * P_{cfg}\{\texttt{VP --> VB NP} \mid \texttt{VP}\} * \\ & P_{cfg}\{\texttt{VP --> } meets \mid \texttt{VB}\} * P_{cfg}\{\texttt{NP --> } Kate \mid \texttt{NP}\} * \\ & P_{bigram}\{George \mid \texttt{EOS}\} * P_{bigram}\{meets \mid George\} * \\ & P_{bigram}\{Kate \mid meets\} * P_{bigram}\{\texttt{EOS} \mid Kate\} \end{aligned}$$

For the **channel model** probability estimation the authors use a parallel corpus of sentences and their reduced versions, extracted from the *Ziff-Davis* corpus of newspaper articles announcing computer products. A total of 1067 pairs were extracted and used to train the model by aligning related tree nodes among reduced and expanded sentences. From these related subtree alignments the

estimates of syntactical tree expansions were used to define the decoder model. Such pairs are called expansion-templates. Below we show five examples of possible expansion-templates from the "`NP --> DT NN`" structure and their estimated probabilities:

$$P_{exp}\{\texttt{NP --> DT NN} \mid \texttt{NP --> DT NN}\} = 0.8678$$
$$P_{exp}\{\texttt{NP --> DT JJ NN} \mid \texttt{NP --> DT NN}\} = 0.0287$$
$$P_{exp}\{\texttt{NP --> DT NNP NN} \mid \texttt{NP --> DT NN}\} = 0.0230$$
$$P_{exp}\{\texttt{NP --> DT JJS NN} \mid \texttt{NP --> DT NN}\} = 0.0115$$
$$P_{exp}\{\texttt{NP --> DT NNP CD NN} \mid \texttt{NP --> DT NN}\} = 0.0057$$

Afterwards the system is ready to compute the **decoder model**, based on the learnt source and channel models. The aim is to apply it to the new unseen sentences and obtain their likely reduced versions. For a sentence t with n words, there exist a huge set of possible reduction combinations, namely $2^n - 1$ possibilities, though many of them may be eliminated *a priori* due to their ungrammaticality. In order to generate a set of grammatical reduced sentence versions from t the authors applied a generic extractor which was designed for a hybrid symbolic-statistical natural language generation called Nitrogen (Langkilde, 2000). With this tool a selection of sentence subtrees from t are scored, guided by the combination of word bigrams and expansion-templates inferred during the training process. A ranked set of such generated sentence reductions for the sentence

"*Beyond that basic level, the operations of the three products vary widely*"

is shown in Figure 2.5. Each sentence is followed by its negative log-probability score, divided by

Beyond that basic level, the operations of the three products vary widely (1514K)
Beyond that level, the operations of the three products vary widely (1430K)
Beyond that basic level, the operations of the three products vary (1333K)
Beyond that level, the operations of the three products vary (1249K)
Beyond that basic level, the operations of the products vary (1181K)
The operations of the three products vary widely (939K)
The operations of the products vary widely (872K)
The operations of the products vary (748K)
The operations of products vary (690K)
Operations of products vary (809K)
The operations vary (522K)
Operations vary (662K)

Figure 2.5: A rank of reduced sentences, obtained from a given sentence.

the reduction length and this way rewarding longer strings. Generating such a sentence reduction

ranking with different sentence lengths is certainly an interesting feature which enables a system to better comply the user's desired compression rate.

Later Turner & Charniak (2005) made an extension of this work, showing a cleaned-up, and slightly improved the noisy-channel model by incorporating supervised and *semi-supervised* methods for sentence compression and additional linguistic constraints to improve compression. This work stresses out conceptual problems inherent to the original noisy channel model used by Knight & Marcu (2002). Since the noisy-channel is a statistically-based method its quality depends on large training quantities of data, contrary to the relatively scarce amount of data used in Knight & Marcu (2002) - 1035 training and 32 test sentences. The main reason for that was naturally the great difficulty involved in extracting $\langle s,\ t\rangle$ examples selection from corpora. Therefore Turner & Charniak (2005) proposed an unsupervised method to compress sentences without parallel training data by selecting sentence joint rules[1] from the *Penn Treebank*, counting probabilistic context free grammar (PCFG) expansions and then matching up other similar rules to form unsupervised joint events. An example of such a joint rule is

$$\langle \texttt{NP} \longrightarrow \texttt{DT NN},\quad \texttt{NP} \longrightarrow \texttt{DT JJ NN}\rangle\ .$$

This data was used to train a new version of the *decoder-model*, which was named as $P_{expand}(t \mid s)$. In order to avoid ungrammatical compressions, due to artificially generated training pairs, several constraints were added, like for example one that does not allow the deletion of any subtree head, as well as others preventing the deletion of syntactic complements.

The last section of Turner & Charniak (2005) explains the existence of a more theoretical problem in using the noisy-channel model for sentence compression, which generally relies on assuming that the probability of a compressed sentence, the $P(s)$ from Equation 2.11, will simply be its probability as a sentence in the language. However, this is generally false. In fact, and as stated by the authors, one would expect that the probability of a compressed sentence should be higher as a member of the set of all compressed sentences, than it is as a member of all English sentences.

2.3.2 A Symbolic Learning Method

Another approach, proposed by Knight & Marcu (2002), was the **decision-based model** in which instead of inferring a statistical sentence reduction model, the learning strategy was directed towards learning a set of tree sentence transformation rules to simplify sentences. This method is more ambitious, since it aims at rewriting of sentence structure, through syntactical tree transformation, instead of simple tree pruning as in the first model. For example the two schematic sentence parse trees shown below represent a pair of expanded ($t(s)$) and reduced ($t(s_r)$) sentences.

In this approach the system learns syntactic tree rewriting rules, defined through four operator types:

[1] A sentence rule here means basically a sentence parse tree.

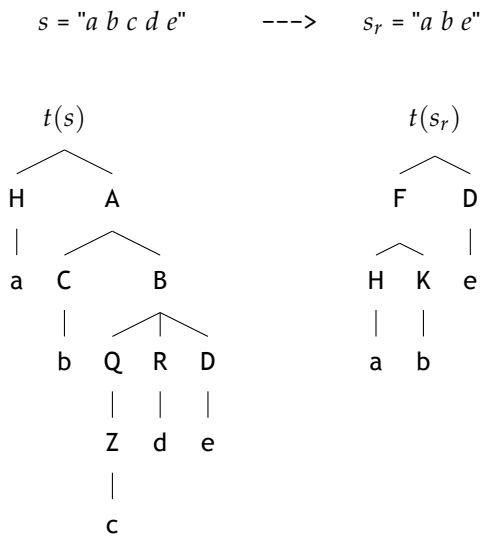

Figure 2.6: Illustration of a transformation for a sentence tree scheme.

Shift, Reduce, Drop, and AssignType. Sequences of these operators are learnt from the training set. Each sequence defines a complete transformation from an original sentence to the compressed version. For each learning instance $\langle t(s), t(s_r) \rangle$ the system will generate possible transformation sequences which codify a rewriting process from $t(s)$ to $t(s_r)$. Any of these rewriting processes starts with an empty stack and an input list with the words subsumed by $t(s)$. Each word in the input list is labeled with all syntactic constituents in $t(s)$. At each step, the rewriting process applies the appropriate operation that aims at reconstructing the smaller sentence $t(s_r)$. The four operation types mentioned above may be briefly described as follows:

- **Shift** - Transfer the first word from the input list to the stack,
- **Reduce** - Pop the k syntactic trees from the top of the stack, combine them into a new tree and push it on the top of the stack,
- **Drop** - Delete sequences of words from the input list that corresponds to syntactic constituents,
- **AssignType** - Change the POS tag of trees at the top of the stack.

The system uses 109 reduce operations, 63 drops, 37 assign types, one for each POS tag, and naturally one single shift operator. This sums up to a total of 210 possible operations that may be applied at each iteration of the reconstructing path from $t(s)$ to $t(s_r)$.

Afterwards, the system uses the C4.5 (Quinlan, 1993) decision tree induction algorithm to learn a set of conditional sequencing operator rules such as the ones shown in Figure 2.7. Each sequence of operators can be compared to a small sentence tree transformation program, consisting of a set of transformation instructions, taking the original sentence tree to its reduced version. The rules

Rule 1: IF previous operation was not "Reduce" AND
previous operation was not "Shift" AND
previous operation was not "AssignType" AND
the input list starts with a syntactic constituent of type WHPP
THEN drop from the input list the words subsumed by WHPP.
Rule 2: IF there is only one tree in the stack AND
previous operation was "Reduce" AND
the syntactic label of the tree in the stack is NP-A AND
the input list starts with a syntactic constituent of type WHNP
THEN drop from the input list the words subsumed by WHNP.
Rule 3: IF previous operation was "Drop" AND
the input list starts with a syntactic constituent of type ADJP AND
the input list does not start with a syntactic constituent of type NP
THEN drop from the input list the words subsumed by ADJP.

Figure 2.7: Three learned rules of sentence syntactic tree transformation toward reduction.

in Figure 2.7 are self-explanatory and the symbols used are from the *Penn Treebank*[1] tag set. For instance WHNP and WHPP mean, respectively, "WH noun-phrase" and "WH prepositional" phrase, while "WH" indicates that the phrase contains a Wh-determiner, as in "*which car*", "*whose father*". The ADJP tag stands for an *adjective phrase* like "*greatest achievement*".

From the same dataset of 1067 related sentence pairs used for the noisy-channel model, the system generates a set of likely transformation paths, yielding a total of 46383 learning instances. An exam-

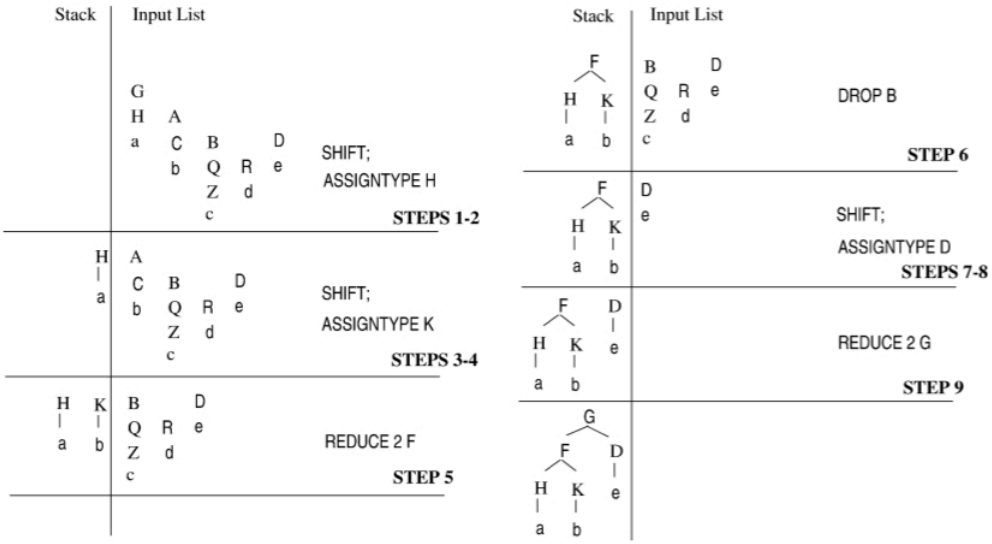

Figure 2.8: A sequence of 9 steps showing the operators for transforming $t(s)$ into $t(s_r)$.

ple of a sequential tree transformation through several operation steps is shown in Figure 2.8, which

[1] http://www.cis.upenn.edu/~treebank/ [October 2010].

was retrieved from the original paper and is related to the tree pair presented in Figure 2.6.

These last approach of learning transformation rules for reducing syntactical trees seems quite interesting. However, it has at least two major drawbacks. First, the rules were learned from a small training data set. Second, restricting the transformation operations to just these four operations seems insufficient. We think that more sophisticated transformation should be integrated to be able to tackle more complex transformations, like syntactic subtree fusions, needed when we have for example phrases conveying similar information.

The improvement of this approach could be fruitful, but it is hard to supply a sufficient training set for inducing a relevant rule set. This tends to be even more critical as new transformational operations are introduced in the system. We think that instead of using a propositional learner (C4.5), the learning task could be better reformulated using a relational learning method, like *Inductive Logic Programming* approach. This is what we have done in our case (see chapters 5 and 6). We have also decided to work with features based on a *shallow parser*, instead of full parser, as it is more simple and the majority of human languages still do not have electronic versions of full parsers available.

2.3.3 Mixing ML and Predefined Rules

A work which combines the *machine learning* framework with a set of handcrafted rules was presented by Vandeghinste & Pan (2004). They described a sentence compression system, built with the purpose of automatic subtitle generation for the deaf and hard-of-hearing people. The system uses a set of lexical and syntactical tools, like a part-of-speech tagger, a word abbreviator, a shallow parser and a subordinate clause detector, to pre-process the input sentence and obtain a shallow parse tree.

Using paired texts obtained from a parallel corpus containing transcripts of television programs and their correspondent subtitles, the system uses statistically-based approaches to recognize irrelevant sentence elements, such as words, chunks, or subordinate clauses, which are candidates for elimination and thereby yielding simplified sentence versions.

Below is a complete list of all language transformation modules used:

- **Tagger** - The TnT *part-of-speech* tagger (Brants, 2000), trained on the Spoken Dutch Corpus (CGN),
- **Abbreviator** - Employs a database of common abbreviations, like for example: $\langle European\ Union,\ EU \rangle$
- **Numbers to Digits** - This module converts all numbers expressed through words, either cardinal or ordinal, to sequences of digits,
- **SharPa** - This module includes a sentence chunker, a subordinate clause detector and generates

a shallow parse tree for that sentence, thus avoiding full sentence parsing.

To train the model, for each related sentence pair $\langle s,\ s_r \rangle$ obtained from the parallel corpus, shallow parse trees are generated and aligned with each other on a chunk basis, yielding a pair of aligned trees $\langle tree(s),\ tree(s_r) \rangle$. The learning task consists of estimating the tree node probabilities of three

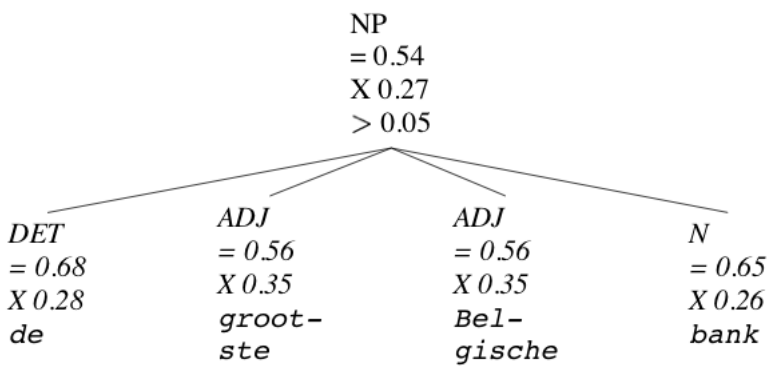

Figure 2.9: Shallow parse tree, with their branch probabilities estimated.

possible operations: *remove* ("X" symbol), *not remove* ("=" symbol) and *reduce* ("$>$" symbol). This last operator is only meant to be applied on the tree of non-terminal nodes. For terminal nodes the probabilities are obtained directly from language corpora. Figure 2.9 shows an example reproduced from the original paper showing different component estimates, for the sentence: "*De grootste Belgische bank*", meaning "*The largest Belgian Bank*".

Afterwards, this probabilistic model is applied to a new sentence, to suggest reductions, which are represented in a similar shallow parse tree format. The probabilities of removal and reduction are calculated. Then for a tree like the one shown in Figure 2.9 all possible removal combinations are computed, including the possibility of no compression at all. Finally a rank of potential sentence compressions is generated. In the previous example the probability of no compression would be equal to $0.54 * (0.68 * 0.56 * 0.56 * 0.65) = 0.07485$ and the probability for sentence reduction ("$>$") through the "grootste" (largest) word removal would be equal to: $0.05 * (0.68 * 0.35 * 0.56 * 0.65) = 0.00433$, while the removal of "bank" yields a lower value, thus being less likely: $0.05 * (0.68 * 0.56 * 0.56 * 0.26) = 0.00277$.

This probabilistic model revealed itself insufficient for ensuring grammatical correctness in most cases. Therefore, several handcrafted constraints for ensuring that critical sentence elements were not removed, were supplied to the system. Now, for a given sentence, after obtaining the sentence reduction versions, from the model, the system uses this rule set to filter out those versions that are violating these grammatical constraints. Figure 2.10 shows three examples of such rules.

Even with this engineered rule set ensuring grammaticality, supplied to the system, the results re-

- If a node is of type NP, keep
  - each noun.
  - each nominalised adjectival phrase.
  - each token which is in the list of negative words.
  - the determiners.
  - the numerals.
  - the indefinite prenominal pronouns.
- If the node is of type adjectival phrase, keep
  - the head of the adjectival phrase.
  - the prenominal numerals.
  - each word which is in the list of negative words.
- If a node is of type PP, keep
  - the preposition.
  - the determiners.
  - the necessary tokens of the NPs.
- If the node is of type OTI, keep
  - the verbs.
  - the *te* + *infinitives*.

Figure 2.10: Examples of handcrafted constraints ensuring grammatical correctness.

ported in Vandeghinste & Pan (2004) still show rather low quality rate, with so called "acceptable correctness" ranging from 32% up to 51%, depending on the test set.

Having presented the main relevant approaches that we have identified as being closely related to our work, we finish this chapter by restating the main issues of our system, which is explained in detail in the following chapters. Unlike any system presented earlier, our system is designed to operate in a real web/text scenario. From a huge collection of automatically gathered web news stories, paraphrases are automatically identified and extracted. Then these are automatically aligned and certain type of learning instances are extracted. Afterwards, a learning process is carried out, which generates sentence reduction rules. Finally the system can use these new induced rules to simplify sentences from new unseen documents. This brief description is represented in the pipeline scheme, shown previously in Figure 1.2. In our system the whole process of sentence reduction is completely automatic. It exploits a huge amount of on-line text available nowadays, extracting and generating the sentences to be used in the learning process.

The next chapter starts the description of our system's first main module, concerned with automatic *paraphrase extraction* from *web news texts*.

Chapter 3

Paraphrase Extraction

"He that will not apply new remedies must expect new evils, for time is the greatest innovator."

Francis Bacon, Essays

In general, a paraphrase consists in a pair of text segments, usually sentences[1], conveying mainly the same information yet possibly each one may be written differently. The term *paraphrase* derives from the Latin *paraphrasis* and from the Greek *para phraseïn*, which means *additional manner of expression*. There are a range of paraphrase types, varying from lexical reorganization, as in the example from Figure 3.1, to more complex ontological or semantic rearrangements, like in Figure 3.2.

S_a: *Due to high energy prices, our GDP may continue to fall.*
S_b: *It is likely that our GDP will continue to fall, as a consequence of high energy prices.*

Figure 3.1: Lexical paraphrase type.

S_a: *To be or not to be.*
S_b: *Should I kill myself?*

Figure 3.2: Semantic paraphrase type.

These previous two paraphrase examples represent two points, each one at an extreme point of the paraphrase "spectrum". Our main objective is not to study paraphrases in general, but rather to use certain types of paraphrases, that are closer to the lexical type (Figure 3.1), as our "raw material" for the sentence reduction problem. Therefore, our initial work was directed towards the extraction of paraphrases from a corpus in a complete automatic form. We aimed at creating a relatively large corpus, containing several hundreds of thousands pairs or even more. As a consequence, we searched for "suitable" electronic texts and existing methods of paraphrase identification, in order to fit our needs. A first attempt of using these tools in such setting encountered some difficulties, which led us to design new functions, more appropriate for our main objective. A discussion of this is detailed throughout the subsequent sections in this chapter.

[1]Throughout this work, a paraphrase refers always to a pair of two sentences.

3.1 Automatic Paraphrase Corpora Construction

Paraphrase corpora represents a golden resource for learning monolingual text-to-text rewritten patterns[1], satisfying specific constraints, such as length in summarization (Barzilay & Lee, 2003; Jing & McKeown, 2000; Knight & Marcu, 2002; Nguyen *et al.*, 2004; Shinyama *et al.*, 2002) or style in text simplification (Marsi & Krahmer, 2005). However, such corpora are very costly to be constructed manually and will always be an imperfect and biased representation of the language paraphrase phenomena. Therefore reliable and efficient automatic methodologies capable of paraphrase extraction from text, leading to construction of paraphrase corpora, are crucial. In particular, we are mainly interested in a special type of paraphrases, named *asymmetrical paraphrase*, where one sentence entails the other one, as stated in Definition 2. In terms of information content, we define two types of paraphrases: symmetrical and asymmetrical.

Definition 1. *A symmetrical paraphrase is a pair of sentences* $\langle S_a, S_b \rangle$*, where both sentences contain the same information, or in terms of entailment, each sentence entails the other one, i.e:* $S_a \vDash S_b \wedge S_b \vDash S_a$.

The asymmetrical case is defined below in Definition 2.

Definition 2. *An asymmetrical paraphrase is a pair of sentences where at least one sentence is more general or contains more information than the other one, that is either* $S_a \vDash S_b \ \wedge S_b \nvDash S_a$ *or* $S_a \nvDash S_b \wedge S_b \vDash S_a$.

An example of an asymmetrical paraphrase is shown next:

S_a: *The control panel looks the same but responds more quickly to commands and menu choices.*
S_b: *The control panel responds more quickly.*

Figure 3.3: An asymmetrical parafrase.

In fact, text-to-text generation is a particularly promising research direction, given that there are naturally occurring examples of comparable texts that convey the same information yet, written in different styles. *Web News Stories* are obviously a natural space for searching this type of texts. There we find a multitude of events, from different subjects, happening every day, like a *presidential election*, a *space shuttle mission*, or a *terrorist attack*. For each event there are several *news agencies* reporting it. Therefore a set of different texts can be created about the same event. So, given such texts, one can pair sentences that convey the same information, thereby building a training set of rewriting examples i.e. a paraphrase corpus.

[1]For example by applying machine learning techniques.

Algorithm 1 The process of automatic paraphrasic corpus construction.

1: $nclusters \leftarrow readNewsClusters(\text{"WebNewsStories"})$
2: $paraphcorpus \leftarrow \emptyset$
3: **for all** $clust \in nclusters$ **do**
4: $\quad v \leftarrow getArrayOfSentences(clust)$
5: $\quad$ **for** $i = 1$ **to** $size(v) - 1$ **step** 1 **do**
6: $\quad\quad$ **for** $j = i + 1$ **to** $size(v)$ **step** 1 **do**
7: $\quad\quad\quad$ **if** $FuncPsim(v_i, v_j) > \theta$ **then**
8: $\quad\quad\quad\quad paraphcorpus \leftarrow paraphcorpus \cup \{\langle v_i, v_j \rangle\}$
9: $\quad\quad\quad$ **end if**
10: $\quad\quad$ **end for**
11: $\quad$ **end for**
12: **end for**
13: $save(paraphcorpus)$

We have investigated several existing functions for paraphrase identification in corpora. However, the difficulties and limitations revealed by these led us to propose new functions, as explained later in this chapter. Before entering in this topic we provide a conceptual view, expressed through an Algorithm 1, of the process used for extracting paraphrases from corpora. From a given "cluster of web news stories"[1], the set of all sentences are gathered in an array of sentences ("v" in Algorithm 1). Afterwards paraphrasic sentence pairs are searched sequentially in vs (line 5 to 10) and added to the corpus (line 8). Note that "$FuncPsim(v_i, v_j)$" is the paraphrase detection function, in the algorithm. The decision is made upon a pre-defined threshold (θ in line 7), which depends from the function being used and certain practical needs one may have.

In Algorithm 1 we have a computational complexity of $O(n^2)$ in the process of paraphrase identification from a vector of sentences. Therefore an important concern in choosing "$FuncPsim(v_i, v_j)$" was efficiency. This is further described in Section 3.6.

3.2 Functions for Paraphrase Identification

The issue of identifying paraphrases in monolingual comparable corpora has become increasingly more relevant, as researchers realize the importance of such resources for many *Natural Language Processing* (NLP) areas, such as *Information Retrieval*, *Information Extraction*, *Automatic Text Summarization*, and *Automatic Text Generation* (Barzilay & Lee, 2003; Carroll *et al.*, 1999; Chandrasekar & Srinivas, 1997; Grefenstette, 1998; Jing & McKeown, 2000; Knight & Marcu, 2002; Lapata, 2003; Marsi & Krahmer, 2005; Nguyen *et al.*, 2004; Shinyama *et al.*, 2002). In particular, three differ-

[1] A set of related news texts covering approximately the same event.

ent approaches have been proposed for paraphrase detection: unsupervised methodologies based on lexical similarity (Barzilay & Lee, 2003; Dolan *et al.*, 2004), supervised methodologies based on context similarity measures (Barzilay & Elhadad, 2003) and methodologies based on linguistic analysis of comparable corpora (Hatzivassiloglou *et al.*, 1999).

Microsoft researchers (Dolan *et al.*, 2004) carried out a work to find and extract monolingual paraphrases from massive comparable news stories. They used the *Edit Distance*[1] (Levenshtein, 1966) and compared it with an heuristic derived from Press writing rules, where the initial sentences from equivalent news stories were considered as paraphrases. The evaluation showed that the data produced by the *Edit Distance* is cleaner and more easily aligned than by using the heuristic. However, when evaluating by using *word error alignment rate*, a function borrowed from statistical machine translation (Och & Ney, 2003), shows that both techniques perform similarly.

In Barzilay & Lee (2003) the authors use the simple word n-gram ($n = 1, 2, 3, 4$) overlap function in the context of paraphrase lattices learning. This string similarity measure is used to generate paraphrase clusters with hierarchical complete-link clustering algoritm. This function is also often employed for string comparison in NLP applications (Lita *et al.*, 2005; Sjöbergh & Araki, 2006).

More advanced techniques rely on context similarity measures such as the one proposed by Barzilay & Elhadad (2003), where sentence alignments in comparable corpora are identified by considering the sentence contexts (local alignment) after semantically aligning equivalent paragraphs. To combine the lexical similarity[2] and the proximity feature, local alignments are computed on each paragraph pairs, through dynamic programming. Although this methodology shows interesting results, it relies on *supervised learning* techniques, which requires huge quantities of training data that may be scarce and difficult to obtain in many situations.

Others, as Hatzivassiloglou *et al.* (1999), go further by exploring heavy linguistic features combined with *machine learning* techniques to propose a new text similarity function. Once again, this is a supervised approach and also heavily dependent on valuable linguistic resources, which are not available for the vast majority of languages. We agree on the issue that linguistic resources may improve accuracy and accordance with human judges. However, this approach limits the application of such systems to very few languages, for which such resources are available.

Finally, we address the work carried out by Stent *et al.* (2005), where a set of evaluation metrics are compared for the task of text-to-text generation. They compare NIST simple string accuracy (SSA), BLEU, and NIST n-gram co-occurrence metrics, Melamed's F-measure (Turian *et al.*, 2003), and *latent semantic analysis* (LSA). The comparison was done with respect to *fluency* (word order variation) and *adequacy* (word choice variation) of generated candidate sentences by comparison with one or more

[1] Also known as the *Levenshtein Distance*

[2] With the cosine similarity measure.

reference sentences. The authors claim that these automatic evaluation metrics are not adequate for the task of fluency evaluation, and are barely adequate for evaluating adequacy. So, these results reinforce our motivation to investigate new sentence similarity functions for paraphrase detection. We have taken into account in particular the fluency issue, as we were also aiming to focus on shallow paraphrase identification functions, due to efficiency and portability reasons[1]. To address adequacy one would have to introduce deeper linguistic analysis, working at least with a thesaurus.

In the literature (Barzilay & Lee, 2003; Levenshtein, 1966; Papineni *et al.*, 2001), we can find the *Levenshtein Distance* (Levenshtein, 1966) and what we call the *Word N-Gram Overlap Family* (Barzilay & Lee, 2003; Papineni *et al.*, 2001). Indeed in the latter case, some variations of word n-gram overlap functions are proposed, but not clearly explained. In this section, we review these existing functions and also propose a new n-gram overlap function based on the *Longest Common Prefix* (Yamamoto & Church, 2001).

3.2.1 The Levenshtein Distance

This function, also known as *Edit Distance*, was created for string similarity computation. Considering two strings, the function computes the number of character insertions, deletions and substitutions that would be needed to transform one string into the other one (Levenshtein, 1966). The function may be adapted for calculating *Sentence Edit Distance* by using words instead of characters (Dolan *et al.*, 2004). Considering two sentences, it calculates the number of word insertions, deletions and substitutions that would be needed to transform one sentence into the other one.

A problem that we observed, when applying this function for paraphrase detection on text, was that it fails on certain types of true paraphrases like the ones where there are high lexical alternations or different syntactical structures. For example, sentences (1) and (2) would probably no be identified as paraphrases using the *Edit Distance* similarity function, since a high value is computed, indicating the existence of a high string dissimilarity between the sentences and leading thus to the unwanted rejection outcome. This happens despite the fact that both sentences convey exactly the same information and are true paraphrases of each other.

(1) *Due to high energy prices, our GDP may continue to fall, said Prime Minister, early morning.*

(2) *Early morning, Prime Minister said that our GDP may continue to fall, due to growing energy prices.*

Edit Distance was specially conceived to be used with words, where character reordering will neces-

[1]Deeper linguistic analysis implies more time spent on identifying paraphrases and a more language dependent system as well.

sary imply lexical dissimilarity. However, in sentence proximity calculation our basic symbolic unit is the *word*, not the character, and word reordering between paraphrasic sentences is very likely to happen. For instance, in the previous example we have *Edit Distance* of 11, which is quite a lot, considering the sentences word lengths (16 and 17 words, respectively). This value represents a normalized similarity of only 0.3125 ($1 - \frac{11}{16}$).

3.2.2 The Word N-Gram Family

Throughout the literature there are several text similarity measures based on word n-gram overlap. Sometimes it is not clear or it is unspecified which word n-gram version was employed in a given experience and so it remains a bit obscure. Mainly two functions are usually mentioned in the literature: the Word Simple N-gram Overlap and the BLEU Metric, which is detailed next. The analysis and experiment with word n-gram overlap functions also led us to consider and propose a new function from this type, based on the *Longest Common Prefix* (Yamamoto & Church, 2001).

3.2.2.1 Word Simple N-gram Overlap

This is the simplest function that uses word n-gram overlap count between sentences. For a given sentence pair, the function counts how many 1-grams, 2-grams, 3-grams, ..., N-grams overlaps exist between the sentences. Usually N is chosen to be less or equal to 4 (Barzilay & Lee, 2003). Let us name this counting function as $Count_{match}(\text{n-gram})$. For a given $N \geqslant 1$, a normalized metric that equally weights any matching n-gram and evaluates similarity between sentences S_a and S_b, is given by Equation 3.1:

$$sim_o(S_a, S_b) = \frac{1}{N} * \sum_{n=1}^{N} \frac{Count_{match}(\text{n-gram})}{Count(\text{n-gram})} \tag{3.1}$$

where the function $Count(\text{n-gram})$ counts the maximum possible number of n-grams that exist in the shorter sentence, as it governs the maximum number of overlapping n-grams.

For example, in the sentence pair ⟨(1), (2)⟩ previously shown, we have three 4-gram matches: "*our GDP may continue*", "*GDP may continue to*", "*may continue to fall*". Thus we have $Count_{match}(\text{4-gram}) = 3$. We also have four 3-grams, ten 2-grams, and fifteen 1-grams, in that same sentence pair. Therefore we have:

$$sim_o(S_a, S_b) = \frac{1}{4} * \left(\frac{3}{13} + \frac{4}{14} + \frac{10}{15} + \frac{15}{16}\right) = 0.5302$$

3.2.2.2 The BLEU Function

The BLEU (Papineni *et al.*, 2001) metric was introduced as a function to automatically evaluate the performance achieved by a translation system and was employed in some conference contests to judge the quality of their competing systems. In such a competition each system has to generate a translation text from a given source text supplied by the evaluator. Afterwards the evaluator calcu-

lates the quality of the generated translation, by comparing it with a set of reference translations from the original source text, usually created by humans. This comparison is executed through the BLEU function, shown in Equation 3.6. First, the p_n value is calculated as follows:

$$p_n = \frac{\sum_{\mathcal{C} \in \{Candidates\}} \sum_{\text{n-gram} \in \mathcal{C}} Count_{clip}(\text{n-gram})}{\sum_{\mathcal{C} \in \{Candidates\}} \sum_{\text{n-gram} \in \mathcal{C}} Count(\text{n-gram})} \tag{3.2}$$

and then

$$BLEU = BP * \exp \sum_{n=1}^{N} w_n \log p_n \tag{3.3}$$

where BP is a brevity penalty factor and w_n are weighting factors ($w_n \in [0,1]$ and $\sum_{n=1}^{N} w_n = 1.0$). In Equation 3.2 the set $\{Candidates\}$ is the reference translation set and the function $Count_{clip}(\text{n-gram})$ counts the number of n-gram clippings (like matches) between a generated translation and a given reference translation. This means that this function calculates the n-gram relation strength, among two texts and, as a consequence, BLEU can be adapted to compute proximity among sentences. This function adaptation is named here $BLEU_{adapted}$.

$$BLEU_{adapted} = \exp \left[\frac{1}{N} \sum_{n=1}^{N} \log C_n \right] \tag{3.4}$$

where C_n is computed as follows:

$$C_n = \sum_{n=1}^{N} \frac{Count_{match}(\text{n-gram})}{Count(\text{n-gram})} \tag{3.5}$$

For the brevity penalty factor we chose $BP = exp(1 - \frac{r}{s})$, where r and s are respectively the shorter and longer sentence lengths, in terms of word counts. For the weighting factors, we took each one equal to the same value, $w_n = \frac{1}{N}$, and so end up with nearly the geometrical mean of the C_n ratios:

$$BLEU_{adapted} = BP * \left[\prod_{n=1}^{N} C_n \right]^{\frac{1}{N}} \tag{3.6}$$

The $Count_{match}(\text{n-gram})$ function counts the number of n-grams co-occurring between the two sentences, and the function $Count(\text{n-gram})$ counts the maximum number of n-grams existing in the shorter sentence. The maximum number of n-grams (N) was chosen equal to 4 in our experiments, like in many other cases where BLEU is employed.

Considering the sentence pair ⟨(1), (2)⟩ shown previously, we have $BP = e^{1-\frac{16}{17}}$ and

$$BLEU_{adapted} = e^{(1-\frac{16}{17})} * \left(\frac{3}{13} * \frac{4}{14} * \frac{10}{15} * \frac{15}{16} \right)^{\frac{1}{4}} = 0.4856$$

3.2.2.3 Exclusive LCP N-gram Overlap

In most NLP works, the longer a string is, the more meaningful should be the function value (Dias *et al.*, 2000). Based on this principle, we were motivated to investigate an extension of the word

simple n-gram overlap function. The difference between simple and exclusive n-gram overlap lays in the fact that the exclusive form counts *prefix overlapping*[1] 1-grams, 2-grams, 3-grams, ..., N-grams, regarding the Longest Common Prefix (LCP) technique proposed by Yamamoto & Church (2001). For example, if some maximum overlapping 4-gram match is found, then its 3-grams, 2-grams and 1-grams prefixes will not be counted. Only the 4-gram and its suffixes will be taken into account. This is based on the idea that the longer the match, the more significant the match is. Therefore smaller matches are discarded.

In particular, we compute exclusive n-grams co-occurring in a sentence pair by using a suffix-array algorithm proposed by Yamamoto & Church (2001), to efficiently compute n-grams in a long corpus and calculate term and document frequencies. So far, we have never come across an implementation of this n-gram overlap method and so we decided to carry out some experiments with this function as it is based on a sound hypothesis from Natural Language Processing applications. However, in Section 7.1 we show that the simple word n-gram overlap function gives overall better results within this n-gram family. However, this new exclusive n-gram overlap function shows interesting results to classify false paraphrases.

In order to clarify how the word LCP is calculated, the following example is presented, with sentences (3) and (4), having some n-grams in common:

(3) *The President ordered the final strike over terrorists camp.*

(4) *President ordered the assault.*

Between these two sentences we have the LCP n-gram overlap given by: "President ordered the" which is a 3-gram. So the complete set of overlapping n-grams, besides the 3-gram, is: "ordered the" (2-gram) and "the" (1-gram), i.e all its suffixes. To normalize the n-gram overlap, a particular difficulty rises, due to the LCP n-gram considerations, i.e. the maximum number of overlapping n-grams depends on the number of (n+1)-gram overlaps that exist. For example, in the previous case and for 1-grams, we only have one overlapping 1-gram ("the") between the two sentences and not 3 as it could be computed with the word simple n-gram overlap metric, i.e. "*the*", "*President*" and "*ordered*". Thus, with this process of considering exclusive n-grams, it is unlikely to compute similarity based on a weighted sum like in formula 3.1 and another method is defined in Equation 3.7

$$sim_{exo}(S_a, S_b) = \max_{\text{n}} \left\{ \frac{Count_{match}(\text{n-gram})}{Count(\text{n-gram})} \right\} \tag{3.7}$$

where S_a and S_b are two sentences and the functions $Count_{match}(\text{n-gram})$ and $Count(\text{n-gram})$ are the same as above with this new matching strategy. We first calculate $sim_{exo}(S_a, S_b)$ for 4-grams and then for the remaining 3-grams and so on and so forth, and after that the maximum ratio is chosen.

[1] This concept is exemplified later in this subsection.

Continuing with the example of the sentence pair ⟨(1), (2)⟩ shown earlier in this subsection, we obtain

$$sim_{exo}(S_a, S_b) = \max\left\{\frac{3}{13}, \frac{4}{14}, \frac{10}{15}, \frac{15}{16}\right\} = 0.9375$$

A few metrics have been applied to automatic paraphrase identification and extraction (Barzilay & Lee, 2003; Dolan *et al.*, 2004). However, these unsupervised methodologies show a major drawback by extracting quasi-exact or even exact match pairs of sentences as they rely on classical string similarity measures such as the *Edit Distance* in the case of Dolan *et al.* (2004) and Word N-gram Overlap in Barzilay & Lee (2003). For these functions, the more similar two strings are the more likely classified as paraphrases will be. In the limit the "best" pair will be precisely two exactly equal strings. This is clearly naive and we may state that the more similar two strings are, the poorer is the quality of the paraphrases generated. This kind of paraphrasing is usually what a naive plagiarist does! It is desirable to identify paraphrases which share some dissimilarities among their sentences, because this will open or widen the field for semantic relation discovery, according to the *Distributional Hypothesis* (Harris, 1968). This principle suggests that counting the contexts that two words share improves the chance for correct guessing whether they express the same meaning or not. One of our later work confirmed this claim (Dias *et al.*, 2010), where paraphrases, automatically extracted with the functions proposed in Section 3.3, were used to discover semantic relations between words.

3.3 New Functions for Paraphrase Identification

To overcome the difficulties of the existing functions reported in the previous section, we have investigated new paraphrase identification functions (Cordeiro *et al.*, 2007d). Contrary to the classical functions, these new ones share the common characteristic of having an "hill shape curve", with zero or near zero values near the domain boundaries and a maximum value reached in between, as illustrated in Figure 3.4 where some of these functions are drawn. The domain for these functions is usually[1] the $[0, 1]$ interval. These new functions have two main objectives. On one hand, to avoid paraphrases where both sentences are equal or almost equal, and on the other hand also to reject pairs of sentences which do not share any lexical unit, i.e completely different sentences. A certain degree of sentence asymmetry in the paraphrase is desired, but only until a certain level, and beyond that level the sentence pair will become progressively less relevant for our objectives. We will call these new functions as *asymmetrical paraphrase identification* functions or simply as **AP-functions**, while the classical paraphrase identification functions will be referred from here on as **SP-functions**, where the "**S**" stands for "symmetrical", i.e symmetrical paraphrase identification functions. In our

[1] It is straightforward to rescale the domain in order to adjust this to some specific feature combinations with value outside the $[0, 1]$ interval.

research work five AP-functions were experimented with, and these are show below.

$$y_1(x) = 1 - 2|x - \tfrac{1}{2}| \quad (triangular) \tag{3.8}$$

$$y_2(x) = 4x - 4x^2 \quad (parabolic) \tag{3.9}$$

$$y_3(x) = sin(\pi x) \quad (trignometric) \tag{3.10}$$

$$y_4(x) = ae^{-\frac{(x-b)^2}{2c^2}} \quad (gaussian) \tag{3.11}$$

$$y_5(x) = -xlog_2(x) - (1-x)log_2(1-x) \quad (entropic) \tag{3.12}$$

Each function curve is shown in the graph from Figure 3.4, identified by its name $y_1, ..., y_5$. It is

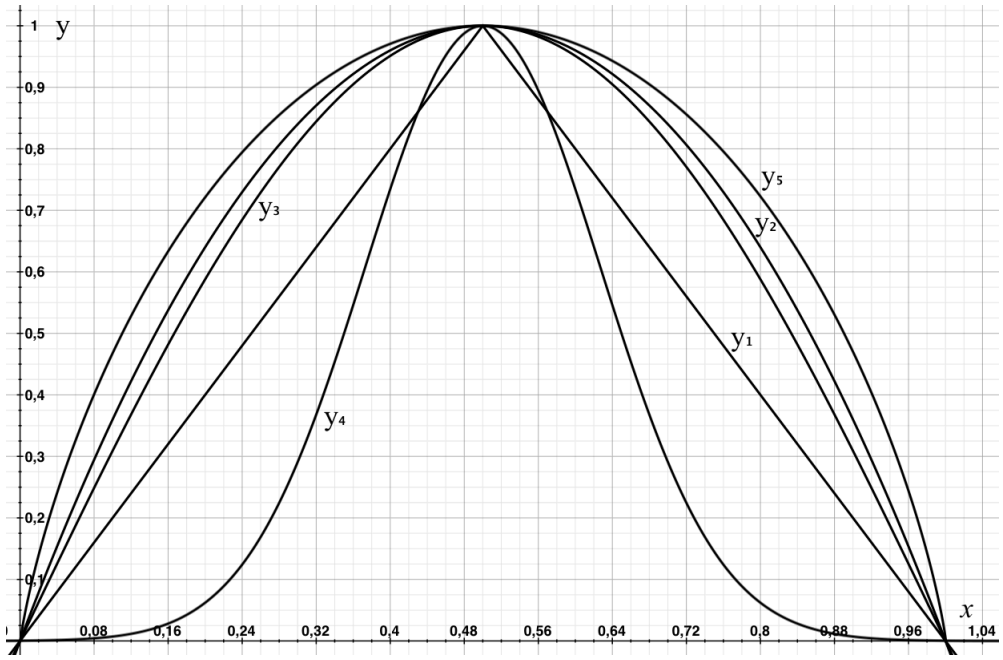

Figure 3.4: *Hill shape* functions for paraphrase identification.

important to remark that the gaussian function (y_4) is in fact a family of functions, depending on the a, b, and c parameters, which in our tests were chosen as $a = 1$, $b = 0.5$, and $c = 0.3$. For a gaussian function, the a parameter defines the function maximum value, the b parameter sets the x value, where the function reaches its maximum, and the c quantity shapes the curve x, where greater c values means more stretched bell curves.

In the paraphrase detection functions presented earlier (e.g. $y_1(x)$), the x represents certain features, computed for the paraphrase sentences, like for example the number of word n-gram overlaps[1] or the number of matching sinsets or any other combination of features one may decide to consider in order to calculate this sentence connectivity/relatedness. A wide range of more or less complex possibilities may be considered here. However, in our work (Cordeiro *et al.*, 2007d) we simply count the *exclusive lexical links* between two sentences. For a given sentence pair, an *exclusive lexical*

[1] More often used.

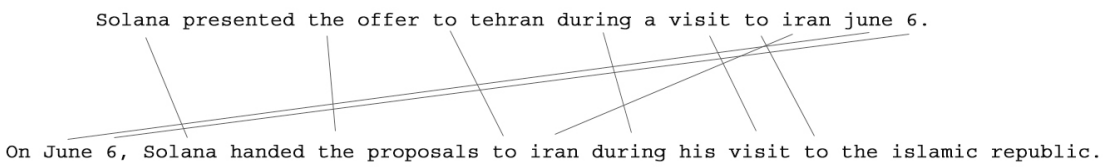

Figure 3.5: Exclusive lexical links between a sentence pair.

link is a connection between two equal words, from each sentence. When such a link holds then each word becomes bounded and cannot be linked to any other word. This is illustrated in Figure 3.5, where, for example, the word "the" in the first sentence has only one link to the first word "the" in the second sentence and the second word "the" in that sentence remains unconnected.

The number of exclusive links bounding two sentences, is represented by λ, and we may now take $x = \frac{\lambda}{m}$ and $x = \frac{\lambda}{n}$, where m is the number of words in the longer sentence, and n the number from the shorter one. Each fraction represents a participation rate for a "relationship" connecting these two sentences. We may even take a combination of these two proportions, like for example the geometric mean: $\sqrt{\frac{\lambda}{m} * \frac{\lambda}{n}}$.

The main relevant issue with this type of hill-shaped functions is not the exact form of how x is calculated but the general shape of the curve. These curves convey a common meaning, since the maximum value is reached strictly inside the $]0, 1[$ interval, in some cases near the 0.5 value, which means, on one hand, that a certain degree of dissimilarity between the paraphrase sentences is "desirable" while, on the other hand, that both the excessive dissimilarity and similarity, tends to be penalized as we when $\lim_{x \to 0} f_{hill}(x) = 0$ and $\lim_{x \to 1} f_{hill}(x) = 0$. The main difference from the classical paraphrase detection functions, is that the latter maintain their monotonicity and reach the maximum when $x = 1$, i.e. $f(x_{max}) = 1$. An example of a paraphrase that would have a high value with classical functions and lower value for f_{hill} functions, is shown in Figure 3.6.

1. *The stock has gained 9.6 percent this year.*
2. *The stock has gained 9.6% this year.*

Figure 3.6: An example of a too similar paraphrase.

From the textual entailment standpoint, this example is obviously of very low utility. Hence, the main advantage of a hill-shaped curve is that it allows a degree of "controlled" variability between the sentences forming the paraphrase pair, thereby useful for the construction of an asymmetrical paraphrase corpus, which is explored for sentence compression.

The experimental results achieved provided evidence that these AP-functions perform better than the classical SP-functions. Better paraphrases are extracted from web news corpora, that are asym-

metrical and satisfy the *Distributional Hypothesis*. Results also showed that even when using a corpus containing only symmetrical type paraphrase, like for example the *Microsoft Paraphrase Corpus*, the f_{hill} (or AP-functions) still achieve better results than the SP-functions.

3.3.1 The *Sumo* Function

In Cordeiro *et al.* (2007d) another function - *Sumo* - containing the same main characteristics as the f_{hill} functions, was investigated and the results showed that it preforms even better (though only slightly better in some cases) than any other f_{hill} function, when tested in the majority of corpora. This may be verified by looking at the results presented in Subsection 7.1.5.

Indeed, the *Sumo* metric was proposed for paraphrase identification even before we have conceptualized the f_{hill} mathematical functions. Our aim was to obtain better results than with the traditional functions. Besides, we required that our function would be computed efficiently, while processing the huge amount of text, from *web news stories*.

The *Sumo* metric was inspired by the *Entropy* function that is used to compute the *Information Gain* of a particular attribute in a propositional learning problem. Viewing asymmetrical paraphrasing as a one way entailment, as defined previously, we may see the entailed sentence as a compressed output obtained from a given input sentence through a *noisy channel model* transmission, similarly to what is presented by Knight & Marcu (2002). Hence, a question arises regards what the information gain value of the compressed sentence is in relation to the input expanded one. This thought has led us to define a kind of entropic function, calculated on the basis of the relative exclusive links[1] connecting two sentences. Let λ be the number of exclusive lexical links connecting two sentences, m the number of words in the larger sentence, and n the number of words in the shorter one. Then we consider the fractions $\frac{\lambda}{m}$ and $\frac{\lambda}{n}$, and combine them trough the function:

$$S(m,n,\lambda) = -\alpha \log_2(\frac{\lambda}{m}) - \beta \log_2(\frac{\lambda}{n}) \tag{3.13}$$

where $\alpha, \beta \in [0,1]$, such that $\alpha + \beta = 1$. Note that if $\alpha = \beta = 0.5$, then $2^{S(m,n,\lambda)}$ is equal to the geometric mean of the two proportions: $\sqrt{\frac{\lambda}{m} * \frac{\lambda}{n}}$. These are weighting parameters, giving more or less importance to one or another proportion. In our experiments[2] we used $\alpha = \beta = 0.5$, equally weighting both proportions.

Since this function is undefined for $\lambda = 0$, we naturally defined it to be equal to zero in that case, because no lexical connection exists between the two sentences. Of course, we are aware that there are paraphrases where their sentences do not have any lexical match at all. However, these type of paraphrase analysis requires more advanced linguistic tools, like thesaurus and ontologies, which we decided not to use, at least for now, maintaining our method as efficient as possible. Besides, these

[1] See the explanation, near Figure 3.5.

[2] The results are reported in Subsection 7.1.5.

tools are not available for many languages, as we have mentioned earlier. The reader may observe that:

$$\lim_{\frac{\lambda}{m} \to 0} S(m,n,\lambda) = +\infty \tag{3.14}$$

which conceptually looks a bit strange. The simple fact that $S(m,n,\lambda) > 1.0$ is not problematic. To overcome the previous functional singularity and to rescale its range to a more conventional $[0,1]$ interval, we have redefined the sentence similarity function by introducing a new condition (branch), named the **penalization branch**. So, we define our *Sumo* as shown in Equation 3.15.

$$Sumo(S_a, S_b) = \begin{cases} S(m,n,\lambda) & if \;\; S(m,n,\lambda) \leq 1.0 \\ 0 & if \;\; \lambda = 0 \\ e^{-k*S(m,n,\lambda)} & otherwise \end{cases} \tag{3.15}$$

The penalization branch $e^{-k*S(m,n,\lambda)}$ recomputes the *Sumo* output to the $[0,1]$ interval and, as the name indicates, it penalizes the too dissimilar sentence pairs. The k parameter is the penalization magnification parameter. The greater its value, the more likely it is that dissimilar sentences will be rejected as paraphrases. We have decided to fix $k = 3$, after several initial experiments, revealing better results with this value.

For the the sentence pairs $\langle$(1), (2)$\rangle$ and $\langle$(3), (4)$\rangle$ previously shown we have:

$$S_{\langle(1),(2)\rangle}(17,16,15) = -0.5 \cdot \log_2(\frac{15}{17}) - 0.5 \cdot \log_2(\frac{15}{16}) = 0.1368$$

and

$$S_{\langle(3),(4)\rangle}(9,4,3) = -0.5 \cdot \log_2(\frac{3}{9}) - 0.5 \cdot \log_2(\frac{3}{4}) = 1.0$$

Note that the first pair gets a relative low value, smaller than any one obtained with the other functions, including the word *edit distance*, while in the second case the maximum value is achieved. This is so because the $\langle$(3), (4)$\rangle$ pair fits perfectly in the kind of asymmetrical paraphrases we are searching for, serving as learning examples for sentence reduction. On the contrary, the $\langle$(1), (2)$\rangle$ pair is much less interesting for learning reduction rules, as one sentence is almost a reordered version of the other one.

The *Sumo* function may raise some mathematical and practical questions to the reader, hence in order to clarify some possibly obscure issues, more details are presented throughout the rest of this

subsection. Let us start by looking at which conditions we still obtain $S(m,n,\lambda) \leq 1.0$:

$$\begin{aligned} S(m,n,\lambda) \leq 1.0 \quad &\Leftrightarrow \quad -\alpha * log_2(\frac{\lambda}{m}) - \beta * log_2(\frac{\lambda}{n}) \leq 1.0 \\ &\Leftrightarrow \quad log_2(\frac{\lambda}{m})^{\alpha} + log_2(\frac{\lambda}{n})^{\beta} > -1.0 \\ &\Leftrightarrow \quad (\frac{\lambda}{m})^{\alpha} * (\frac{\lambda}{n})^{\beta} > \frac{1}{2} \\ &\Leftrightarrow \quad \lambda > \frac{1}{2}\, m^{\alpha}\, n^{\beta} \end{aligned} \tag{3.16}$$

Condition 3.16 states that the number of lexical links is bounded by the sentences number of words, and if that number is low, with $\lambda \leq \frac{1}{2}\, m^{\alpha}\, n^{\beta}$ then $S(m,n,\lambda) \geq 1.0$ and consequently the *Sumo* value should start to decrease. In this case the output value is calculated through the penalization branch, in which we have:

$$\lim_{S(m,n,\lambda) \to +\infty} Sumo(S_a, S_b) = 0 \tag{3.17}$$

In our experiments[1] we took $\alpha = \beta = 0.5$, which implies that the connection constraint is $\lambda > \frac{1}{2}\sqrt{m\,n}$, and the following table illustrates numerically several situations with different input values and their effects on this constraint and the final *Sumo* output values.

Table 3.1: *Sumo-Metric* output examples.

	m	n	λ	$\frac{1}{2}\sqrt{m\,n}$	$S(m,n,\lambda)$	$Sumo(S_a,S_b)$
1	20	18	17	9.49	0.158	0.15846
2	20	18	11	9.49	0.786	0.78649
3	20	18	3	9.49	2.661	0.00034
4	20	10	9	7.07	0.652	0.65200
5	20	10	6	7.07	1.237	0.02446
6	20	10	2	7.07	2.822	0.00021
7	20	5	5	5.00	1.000	0.04979
8	20	5	3	5.00	1.737	0.00546
9	20	5	1	5.00	3.322	0.00005

In this example the size of the larger sentence is maintained at 20 words for all cases, and only the smaller one varies from 18 down to 5, simulating an increasing asymmetric pair size. There are three examples with length 18, three with 10 and the last three with 5 words, representing shorter sentences. Within any of these three subsets, we have a variation in the number of connecting lexical links: λ. This example shows how far the sentence asymmetry is allowed to vary before entering the penalization region ($\lambda \leq \frac{1}{2}\sqrt{m\,n}$) and what the effects are of the respective function's penalization branch. Note that only cases 1, 2 and 4 fall within the "normal" region. Case 2 has a greater output value than case 1, despite the fact that in this last case almost all words from the smaller sentence

[1] See Subsection 7.1.5.

are connected (17). This shows how *Sumo* prefers some degree of dissimilarity between sentences, which is important from the point of view of the linguistic *Distributional Hypothesis*. It helps to obtain pairs of sentences with high lexical matches and high lexical differences as well, and at the same time, which may be helpful to induce sentence transformation rules (i.e. sentence reduction rules) the main objective of this work.

In order to have a graphical representation of *Sumo*, one may consider $x = \frac{\lambda}{m}$ and $y = \frac{\lambda}{n}$, which transforms $S(m, n, \lambda)$ in a two-dimensional function: $S(x, y) = -\alpha * log_2(x) - \beta * log_2(y)$, illustrated in Figure 3.7, for $\alpha = \beta = 0.5$.

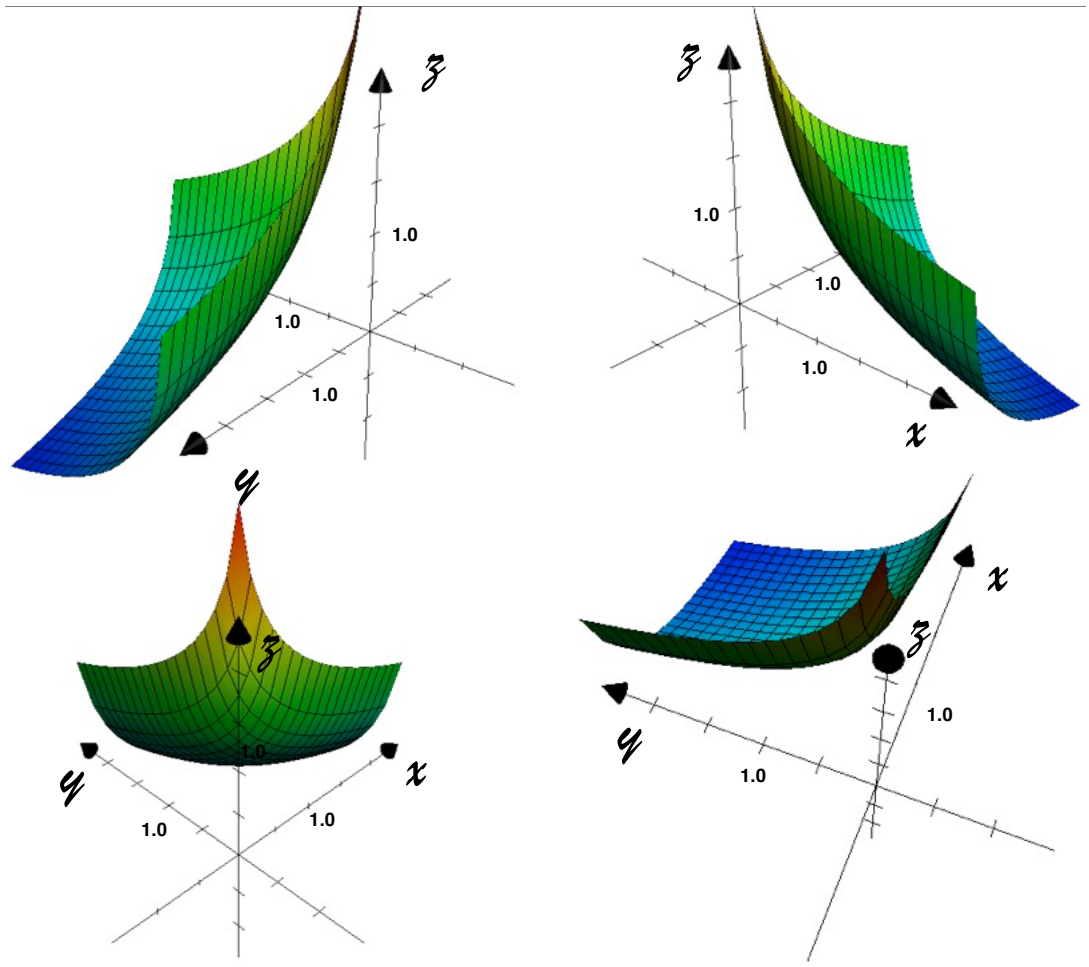

Figure 3.7: A graphical representation for $z = S(x, y)$, with $\alpha = \beta = 0.5$, represented through four different views.

For the sake of simplification and to have a clear view of the function, we also consider projections in the two-dimensional subspace, with $y = S(x, 1)$, $y = S(x, \frac{3}{4})$, and $y = S(x, \frac{1}{2})$. These functions are represented in Figure 3.8 respectively with colors black, blue and green.

The red line represents the penalization branch, in this case $e^{-3*S(x,1)}$, which was rotated and trans-

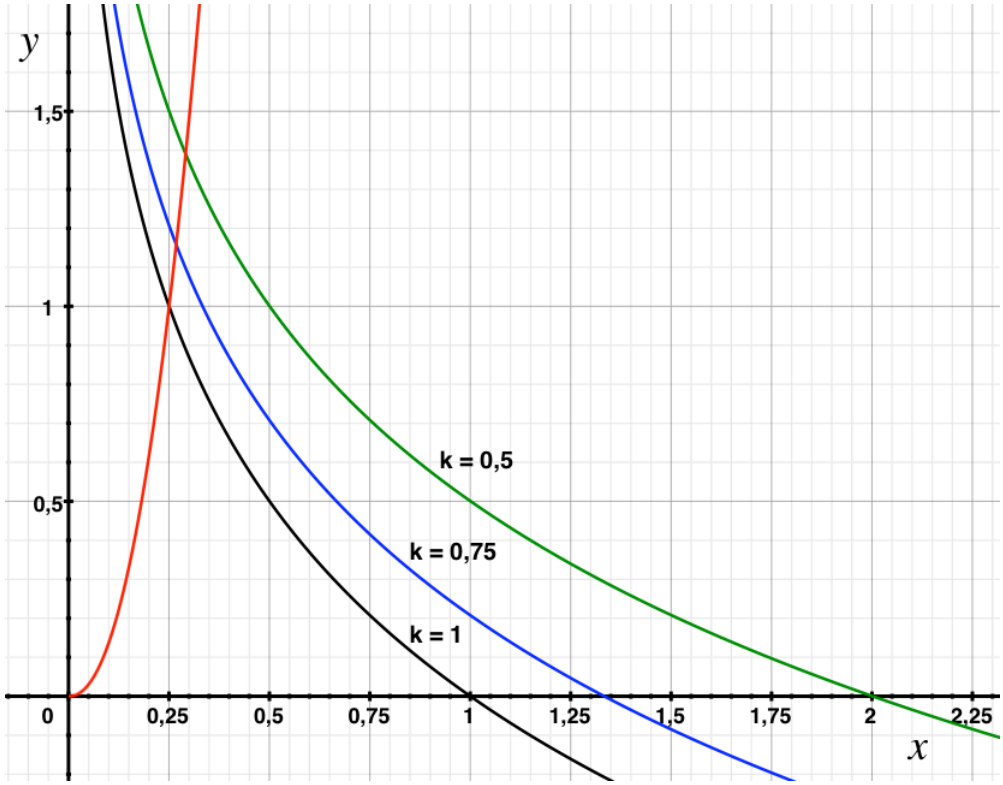

Figure 3.8: A graphical representation for $y = S(x,k)$, with $\alpha = \beta = 0.5$, and $k \in \{\frac{1}{2}, \frac{3}{4}, 1\}$.

lated in order to match the $S(x,1)$ function at point $(\frac{1}{4},1)$. Looking just at the conjunction of the black and red graphs, we can see a complete graphical example of the *Sumo* function, which is a projection among many possible ones, evidencing that this function has an hill-shape curce with the properties described at the beginning of this section.

3.4 Paraphrase Clustering - An Investigated Possibility

In an early phase of our research, also motivated by some other's work (Barzilay & Lee, 2003), we have decided to investigate the application of clustering algorithms to extract paraphrase clusters from text sentences. A paraphrase cluster is a set of sentences where each pair constitutes a paraphrase. In the work of Barzilay & Lee (2003) it is claimed that clusters of paraphrases could lead to better learning of text-to-text rewriting rules compared to simple paraphrasic examples. However, as Barzilay & Lee (2003) did not propose which clustering algorithm should be used, we carried out a study with a set of clustering algorithms and present the comparative results. Contrarily to what was expected clustering did not bring any benefit.

Clustering is based on a similarity or (distance) matrix $A_{n \times n}$, where each element a_{ij} is the similarity (distance) between sentences s_i and s_j. We have conducted experiments with four clustering

algorithms on a corpus of *web news stories* and then three human judges manually cross-classified a random sample of the generated clusters. They were asked to classify a cluster as a "wrong cluster" if it contains at least two sentences without any relation between them. Results are shown in Table 3.2. The "BASE" column represents the baseline, where the *Sumo* function was applied rather than

Table 3.2: Precision of clustering algorithms

BASE	S-HAC	C-HAC	QT	EM
0.618	0.577	0.569	0.640	0.489

clustering. Columns "S-HAC" and "C-HAC" express the results for *Single-link* and *Complete-link Hierarchical Agglomerative Clustering* (Jain *et al.*, 1999). The "QT" column shows the *Quality Threshold* algorithm (Heyer *et al.*, 1999) and in the last column "EM" stands for the *Expectation Maximization* clustering algorithm (Hogg *et al.*, 2005).

The main conclusion we can draw from Table 3.2 is that clustering tends to achieve worse results than the paraphrase pair extraction. Only the QT achieves a somewhat better results. Moreover, these results with the QT algorithm were applied with a very restrictive value for cluster attribution with an average of 2.32 sentences per cluster (see Table 3.3). In fact, Table 3.3 shows that most of the

Table 3.3: Figures about clustering algorithms

Algorithm	# Sentences/# Clusters
S-HAC	6.23
C-HAC	2.17
QT	2.32
EM	4.16

clusters have less than 6 sentences, which forces us to question the results presented by Barzilay & Lee (2003) who only keep the clusters that contain more than 10 sentences. So, not only the number of generated clusters is very low, but more importantly, all clusters with more than 10 sentences appeared to be of very bad quality.

Given this, we have abandoned the idea of extracting clusters of paraphrases and decided to follow the extraction of just sentence pairs. Nevertheless, we think that more directed algorithms might be investigated in the future for special paraphrasic cluster extraction, instead of using conventional clustering algorithms. For example, it could use two steps. After the extraction of a paraphrasic sentence pair, one may search in the text for other reduced versions of the shorter sentence and add them to the initial pair. This could generate a paraphrasic cluster with one relatively longer sentence and several related reduced versions.

3.5 Extraction Time Complexity

At the beginning of this chapter we have presented the algorithm for automatic paraphrase corpus construction (Algorithm 1). There we mentioned that the process of extracting paraphrases from arrays of sentences, obtained from news clusters, is of a complexity of $O(n^2)$. This led us to search for efficient paraphrase identification functions. Our new proposed functions are more efficient, as they rely on the number of exclusive lexical links. For example counting word n-gram overlaps, which is the basis of several existing functions[1] is computationally more costly. Assuming that we are computing the similarity of two sentences with m and r words ($m \geq r$), *Sumo* a complexity of $O(m * r)$, in terms of word comparisons and in the worst case scenario. However, if any n-gram based function is used, the complexity rises approximately to $O(m * r * N)$, where N is the maximum number of n-grams used. This is the case for Sim_o, Sim_{exo}, and *Bleu*. These differences have been observed in some experiments as shown in the following table. Only three AP-functions are shown,

Table 3.4: Paraphrase extraction times for six different functions.

	Sumo	*Gaussian*	*Entropy*	*N-Gram*	*Bleu*	*Edit*
time	00:31:48	00:30:12	00:32:45	01:30:27	01:45:58	01:31:19

as the others are also based on exclusive lexical links and so also spend nearly the same time, about 30 minutes. The SP-functions spent nearly 3 times more. This is also verified for the Edit function, which has to compute a $(m+1) \times (r+1)$ matrix and carried out several operations on it, in order to obtain the sentence edit distance. Time for Sim_{exo} are not shown, but it is even worse than the time taken by Sim_o.

These *times* were measured on a dataset obtained from seven days of news[2], totaling to about 37MB of text, containing a total of 238 news clusters (34 per day), $11\ 548$ news stories, and a total of $293\ 642$ sentences. This gives an average of 48.52 related news per cluster and 25.43 sentences by each *news story*. We can see that the array of sentences from which paraphrases are searched for and extracted has an average size of $25.43 \times 48.52 = 1233.86$ sentences. So, the spatial complexity in terms of number of clusters processing (p) is linear: $O(p)$. This is because each news cluster is processed independently for the paraphrase search. For example, the computation time spent for 4 weeks (nearly one month) of news data would be approximately 2 hours for the AP functions and 6 hours for the SP-type functions.

These computations were measured on a machine with an *Intel Core 2 Duo* processor, working at 1.83Ghz, having 2GB of RAM, and running a POSIX[3] operating system.

[1] For example *Word N-gram Overlap*, *BLEU*, and *Exclusive LCP N-gram Overlap*

[2] Each day was stored in a separate xml file.

[3] In our case the *Mac OS X*.

3.6 Final Remarks

This chapter was dedicated to the topic of functions for paraphrase detection in text corpora. Several existing functions were described (SP-type), as well as their limitations for asymmetrical paraphrase identification. A set of new functions, AP-type, was also investigated and these are proposed as a better type for *asymmetrical* paraphrase identification. Comparative results are presented in Section 7.1. These also revealed that the AP functions are at least as good as the SP type, even in corpus containing almost symmetrical paraphrases.

In future, improvements could be investigated in order to enhance the results, despite the already high quality achieved (Subsection 7.1). We suggest to investigate the inclusion term relevancy, like for example tf.idf (Salton & Buckley, 1988), and *Part of Speech Tagging*, as input features for the paraphrase identification functions. We believe that word links between sentences should have distinct weights according to such characteristics, since it seems obvious that words do not have the same information or relevancy. It is less important to have a match between *determiners* than between *names* and *verbs*, which convey information about entities and actions. One may also integrate the notion of content n-grams that can be extracted from monolingual corpora as in Dias *et al.* (2000).

The next chapter presents the natural follow-up of work undertaken by discussing the issue of word alignment between paraphrasic sentences. We end this chapter by referring the main scientific contributions achieved in this subject. The work of paraphrase extraction enabled us to produce three publication (Cordeiro *et al.*, 2007a,b,c). These are more detailedly listed and described in Section 1.4.

Chapter 4

Paraphrase Alignment

> "*He that will not apply new remedies must expect new evils, for time is the greatest innovator.*"
>
> **Francis Bacon, Essays**

In Chapter 3 we presented our paraphrase gathering method, which is able to automatically identify paraphrases from text. The next natural step in our research was to investigate different paraphrase alignment techniques. By "paraphrase alignment" we mean the alignment of words inside the pair of sentences forming a paraphrase. Identical or even similar[1] word pairs, one from each paraphrasic sentence pair, are aligned and dissimilar words aligned with empty gaps. For example, consider the following paraphrase, which was automatically extracted from the news:

> S_a: *Tropical Storm Gilbert formed in the eastern Caribean and strengthened into a hurricane Saturday night.*
>
> S_b: *Tropical Storm Gilbert strengthened into an eastern Caribean hurricane on Saturday night.*

One possible alignment is shown in Figure 4.1. A closer analysis shows that in general there may

```
Tropical Storm Gilbert ____________ formed in   the eastern Caribean ...
Tropical Storm Gilbert strengthened ______ into an  eastern Caribean ...

                                      ... and strengthened into a hurricane __ Saturday night.
                                      ... ___ ____________ ____ _ hurricane on Saturday night.
```

Figure 4.1: A possible alignment for two paraphrase sentences.

be more than one possible alignment generated by different algorithms. For instance, the previous example could have been realigned as shown in Figure 4.2.

By analysing qualitatively these two alignments, the latter appears less satisfactory than the former. We note that the latter contains more word gaps. Hence, we have devoted some research effort

[1] We specify later in this chapter what type of similar words we use.

```
Tropical Storm Gilbert formed       in the eastern Caribbean ...
Tropical Storm Gilbert strengthened __ ___ _______ _________ ...

                    ... and strengthened into a  _______ _________ hurricane __ Saturday night.
                    ... ___ ____________ into an eastern Caribbean hurricane on Saturday night.
```

Figure 4.2: Another possible alignment for the same paraphrase.

to discover and employ the algorithms which produce alignments as "good" as possible. We have investigated methods for global and local alignments and proposed a dynamic strategy for choosing the best one at run time. We also propose a new function to model word alignment in paraphrasic sentences. It also allows the alignment of non-identical yet lexically similar words, which mimics the *mutation matrices* in DNA sequence alignments.

In Subsection 4.3.1 we define a function to calculate the lexical quality of an alignment (Equation 4.9). Using this function for the previous two alignment examples we obtain $algval(\text{Figure } 4.1) = 0.728$ and $algval(\text{Figure } 4.2) = 0.679$, which confirms our initial perception.

4.1 Biology-Based Sequence Alignment

Sequence alignment aims at pairing, as best as possible, the symbols contained in at least[1] two sequences. This is done by satisfying a set of alignment constraints, usually defined as a *mutation matrix* in biology. It has been extensively explored in *bioinformatics* for a long time, where one of the most notable early achievement is the Needleman-Wunsch global alignment algorithm (Needleman & Wunsch, 1970). More recently, after the beginning of the *Human Genome Project* in 1990, and boosted by the gradually increasing computation power, a wide and fruitful research was pursued with the aim to perform near optimal and efficient DNA sequence alignment. A draft of the human genome was finished in 2000 and was announced jointly by then US president Bill Clinton and the British Prime Minister Tony Blair on June 26, 2000. Subsequent sequencing efforts led to the announcement of the essentially complete genome in April 2003. In May 2006, another milestone was achieved on the way to the completion of the project, when the sequence of the last chromosome (chromosome one) was published in the scientific journal *Nature* (Gregory *et al.*, 2006).

In *bioinformatics*, sequence alignment is a process of arranging the sequences of DNA, RNA, or a protein to identify regions of similarity that may be a consequence of functional, structural, or evolutionary relationships between the sequences. Aligned sequences of nucleotide or amino-acid residues are typically represented as rows within a matrix. Gaps are inserted between the residues so that identical or similar characters are aligned in successive columns, as shown in Figure 4.3.

[1] There are also *Multi-Sequence-Alignment* methods.

Figure 4.3: DNA sequence alignment.

The alignment construction is essentially algorithmic, like dynamic programming, which in turn is guided by a *mutation matrix* codifying the likelihood for gene mutation. These *mutation matrix* values have a direct implication on the quality of the created alignments.

In Natural Language Processing, sequence alignment has already been employed in some sub-domains like *Text Generation* (Barzilay & Lee, 2002). However, as far as we know word alignment within sentences has not been explored. Therefore, in our work we have investigated alignment methods for aligning words between two sentences that form a given paraphrase. In our problem the sentences represent the sequences and the words are the *base blocks*, the symbols, in those sequences.

Computational sequence alignment usually falls into two main categories: *global* and *local* alignments. In the first one, the algorithm tries to completely align all the symbols in both sequences, while in the second one the objective is to find at least one relevant sub-alignment. The appropriateness of each category depends on particular characteristics. Usually, global alignment is more useful when the sequences in the query set are similar and with near equal size, while local alignments are more appropriate when we have highly asymmetrical sequence sizes containing similar subsequences.

4.1.1 Global Alignment

The Needleman & Wunsch (1970) algorithm was the first method for determining sequence homology (Waterman, 1984). It uses *dynamic programming* to find the best possible alignment between two sequences, with respect to the scoring system used, which includes a *substitution matrix* and a *gap-scoring scheme*. The former defines the cost of aligning two symbols in the sequence, either equal or different. The latter, also known as the *gap penalty*, specifies the alignment cost between a symbol and a gap, which is a void space. This algorithm computes the optimal[1] global sequence alignment

[1]The term "optimal" here means that the best alignment match is generated, depending on the symbol-to-symbol alignment function.

between the two sequences, in which different symbols may be pairwise-aligned, or else a symbol may be aligned with an empty space (a gap). To illustrate how this alignment algorithm works, we present here a small example. Suppose the aim is to align the following two DNA sequences, where each letter represents a gene:

```
SEQUENCE(x):  A C A C A C T A
SEQUENCE(y):  A G C A C A C A
```

Let us assume that the scoring function governing the alignment of two symbols is defined as shown in Equation 4.1, where x_i and y_j represents the *i-th* and *j-th* symbols in the x and y sequences, respectively:

$$w(x_i, y_j) = \begin{cases} 1 & \text{if } x_i = y_j \\ -1 & \text{if } x_i \neq y_j \\ -1 & \text{if } x_i = \text{"\_\_"} \quad \text{or} \quad y_j = \text{"\_\_"} \end{cases} \tag{4.1}$$

This function assigns value 1 for a match, that is, aligning two equal symbols, -1 for a mismatch and for gap penalty, that is, aligning different symbols, including voids, will have a unit cost of -1. The algorithm starts by iteratively (line by line or row by row) filling in a *similarity matrix*, usually designated as $\mathbf{F}$, where the alignment path is dynamically computed and $\mathbf{F}(i, j)$ holds the global alignment score for the subsequences $x_{[1..i]}$ and $y_{[1..j]}$. The exception will is top row and the leftmost column which are initialized with multiples of the gap penalty value, -1 in this case, representing the scores of aligning prefixes of x or y to sequences of gaps. The m and n values in 4.2 and 4.3 represent the lengths of the x and y sequences.

$$\mathbf{F}(i, 0) = (-1) * i \qquad 0 \leq i \leq m \tag{4.2}$$

$$\mathbf{F}(0, j) = (-1) * j \qquad 0 \leq j \leq n \tag{4.3}$$

$$\mathbf{F}(i, j) = maximum \begin{cases} 0 \\ \mathbf{F}(i-1, j) + w(x_i, \_\_) \\ \mathbf{F}(i, j-1) + w(\_\_, y_j) \\ \mathbf{F}(i-1, j-1) + w(x_i, y_j) \end{cases} \tag{4.4}$$

As we can see from the function definition 4.4, each matrix value $\mathbf{F}(i, j)$ is calculated by taking into account three recursive values and the respective cost for establishing the (x_i, y_j) alignment. They are exactly the immediate upper, left, and diagonal matrix values and we may call them as the

previous neighbors. Hence the $\mathbf{F}$ matrix for our two x and y sequences will be:

$$\mathbf{F} = \begin{pmatrix} & \_\_ & A & G & C & A & C & A & C & A \\ \_\_ & 0 & -1 & -2 & -3 & -4 & -5 & -6 & -7 & -8 \\ A & -1 & \mathbf{1} & \mathbf{0} & -1 & -2 & -3 & -4 & -5 & -6 \\ C & -2 & 0 & 0 & \mathbf{1} & 0 & -1 & -2 & -3 & -4 \\ A & -3 & -1 & -1 & 0 & \mathbf{2} & 1 & 0 & -1 & -2 \\ C & -4 & -2 & -2 & 0 & 1 & \mathbf{3} & 2 & 1 & 0 \\ A & -5 & -3 & -3 & -1 & 1 & 2 & \boxed{\mathbf{4}} & 3 & 2 \\ C & -6 & -4 & -4 & -2 & 0 & 2 & 3 & \mathbf{5} & 4 \\ T & -7 & -5 & -5 & -3 & -1 & 1 & 2 & \mathbf{4} & 4 \\ A & -8 & -6 & -6 & -4 & -2 & 0 & 2 & 3 & \mathbf{5} \end{pmatrix} \tag{4.5}$$

In this matrix, the first line and first column are used just for labeling, containing exactly each sequence symbol, the fist column for the x sequence and the first line for the y sequence.

The matrix values starts at the second line and second column, which have indexes equal to zero, i.e. the indexes in matrix **F** vary from zero to m for lines, and to n for columns. Thus we have one line and one column more than the sequence sizes to account for the gap symbol "__", which is positioned at the beginning. For example, the square signaled value in $\mathbf{F}(5,6)$ has as *previous neighbours* $\mathbf{F}(4,6) = 2$, $\mathbf{F}(4,5) = 3$ and $\mathbf{F}(5,5) = 2$. The reward of aligning the 5-th element from the x sequence with the 6-th element from the y sequence is given by $w(x_5, y_6) = w(A, A) = 1$, given the defined scoring set. Therefore the matrix value for matrix position (5,6) is computed as $\mathbf{F}(5,6) = max\{0,\ 2+(-1),\ 3+1,\ 2+(-1)\}\ =\ 4$.

After this construction we are ready to finally make the alignment by following a backward path, starting at the matrix lower rightmost position and successively jumping back to one of the three *previous neighbours*. That is, we have three possible directions to follow let us name them: *north* ($\mathbf{F}_{i,j} \mapsto \mathbf{F}_{i-1,j}$), *north-west* ($\mathbf{F}_{i,j} \mapsto \mathbf{F}_{i-1,j-1}$) and *west* ($\mathbf{F}_{i,j} \mapsto \mathbf{F}_{i,j-1}$). The direction is decided according to the function inherent to the dynamic matrix construction made in the first step, which resembles a kind of remembrance process. In each $\mathbf{F}_{i,j}$ position one needs to determine whether this value was obtained from $\mathbf{F}(i-1,j) + w(x_i, \_\_)$, or from $\mathbf{F}(i,j-1) + w(\_\_, y_j)$, or from $\mathbf{F}(i-1, j-1) + w(x_i, y_j)$, to determine whether the jumping-back direction is *north*, *west*, or *north-west*, respectively.

The effect of jumping back one step forces at least one symbol, either from sequence x or y, to be aligned, with another symbol or an empty space ("__"). If $\mathbf{F}_{i,j}$ is the actual position, then a *north-west* jump will align symbols x_i and y_i, while a *north* jump will align x_i with a void space and a *west* jump aligns y_j with void space too. When a tie exists and there are more than one likely direction to jump back, one is randomly chosen. This process stops when the $\mathbf{F}_{1,1}$ position is reached, providing a full alignment for both sequences. The jumping-back path in our previous example is marked in the

matrix shown using a bold face, and the obtained alignment in this case is:

```
SEQUENCE(x):  A G C A C A C _ A
SEQUENCE(y):  A _ C A C A C T A
```

This alignment algorithm reveals space and time inefficiency when sequence lengths increase substantially, due to the fact that an $m * n$[1] matrix must be processed and held in memory during the computation. This is not problematic for relatively small sequences. However, in the general case of DNA sequence alignments usually many thousands of nucleotides have to be aligned. This complexity bottleneck lead to further research effort in order to discover more efficient algorithms. In particular, several new algorithms are now able to overcome the complexity barriers of many problems and produce useful practical results, as for example the BLAST family (Altschul *et al.*, 1997) and FASTA (Orengo *et al.*, 1992) algorithms.

In our alignment task we do not have these complexity obstacles, because in *web news stories* the average length of a sentence is equal to 20.9 words, which is quite insignificant when compared to DNA sequences which are at least, three or four orders of magnitude longer. Therefore, we could afford to implement an adapted version of the Needleman-Wunsch algorithm for making global alignments between paraphrase sentences.

```
To the horror of their television fans , Miss Ball and Arnaz were divorced in 1960.
__ ___ ______ __ _____ __________ ____ _ ____ Ball and Arnaz ____ divorced in 1960.
```

Figure 4.4: A global alignment for two paraphrase sentences.

Figure 4.4 exemplifies another global word alignment for a new paraphrase. The Needleman-Wunsch algorithm depends directly on the mutation matrix used, which is a substitution matrix that reflects the probabilities of a given symbol-to-symbol (word-to-word, in our case) substitution. Therefore if a given mutation matrix does not model correctly what occurs in the problem domain, then low-quality, or even unsatisfactory alignments may be generated. This is the reason behind the differences between the examples contained in figures 4.1 and 4.2, which show that from the same original paraphrase sentences we can obtain different alignments. The issue of having a mutation matrix to model word substitutions is discussed in Subsection 4.3.

4.1.2 Local Alignment

The Smith-Waterman algorithm is similar, in many aspects, to its predecessor for global sequence alignment - the Needleman-Wunsch algorithm, described in the previous subsection. Proposed nearly ten years later (1981), the Smith-Waterman algorithm was specially conceived to extract at least one sub-alignment from a given sequence pair. According to the literature (Smith & Waterman, 1981) this

[1] Where m and n are the sequence lengths.

algorithm is more adequate than the global alignment for heterogeneous sequences, either having asymmetrical lengths or a great proportion of symbol dissimilarity among sequences, or even both, though in this last situation we tend to have a more unrelated sequence pair and any alignment attempt would be unworthy. For highly similar sequences there is not much difference between applying the global or local alignment procedures, and even if local alignment is used we may have a global alignment as a result. Table 4.1 presents three local alignments, obtained from their respective main sequences and it is evident that a high degree of dissimilarity exists between each main sequence pair is unlikely to produce a global alignment with any real quality. For the sake of representation simplicity and without loosing any generality, we use here character sequences.

Table 4.1: Example of local alignments.

N	Char. Sequences	Alignments
1	ABBAXYTRVRVTTRVTR	X Y T R V
	FWHWWHGWGFXYTVWGF	X Y T _ V
2	ABCDXYDRQZR	A B _ C D X
	GADQZZSTABZCDX	A B Z C D X
3	ABCD	A B _ _ C D
	DQZZSTABZYCDUQRTVUAA	A B Z Y C D

The complexity difficulties remain the same as for the global alignment, due to the same reasons, which are related to the maintenance and processing of the $m \times n$ matrix, where m and n are the sequence lengths. For the same reasons as for the global alignment algorithm, several more efficient algorithms have been investigated during the last 20 years, in order to extract aligned subsequences from long DNA sequences. For our specific word alignment problem this matrix size is relatively small and we can afford to employ the Smith-Waterman algorithm, which produces the best alignment match, depending on the word-to-word alignment function.

The original algorithm extracts only the sub-alignment with maximum score . However, other relevant sub-alignments can exist, besides the best one. For example, by looking carefully at the second example in Table 4.1 we note another possible sub-alignment of length 4:

```
D _ Q Z
D R Q Z
```

This shows that the algorithm could be adapted to generate not only the maximum sub-alignment, but a set of sub-alignments that satisfy some criterium, for example having an alignment value greater than some minimum threshold. In fact, this is relevant for our sentence word alignment problem, since sentences may be rewritten by rearranging their parts in a different order. Furthermore, this motivated us to implement a dynamic alignment method, described in the next section (4.2). Having a method of extracting not just the best, but several relevant sub-alignments, from one para-

phrase, has the effect that a richer aligned paraphrase corpus will be constructed, with different and more detailed phrase alignments. Hence, we implemented a modified version of the original Smith-Waterman algorithm, where several sub-alignments are likely to be generated.

Similarly to what was done for the Needleman-Wunsch algorithm described in the previous subsection, an illustration of the Smith-Waterman algorithm is presented here, and in particular our adaptation involving the extraction of multiple sub-alignments. The sequences used for this illustration are those contained in the second example, in Table 4.1, hence we have:

```
SEQUENCE(x):  A B C D X Y D R Q Z R
SEQUENCE(y):  G A D Q Z Z S T A B Z C D X
```

Here we assume that the scoring function returns 2 for a symbol match and -1 either for a mismatch and for gap penalty alignment:

$$\breve{w}(x_i, y_j) = \begin{cases} 2 & \text{if } x_i = y_j \\ -1 & \text{if } x_i \neq y_j \\ -1 & \text{if } x_i = \text{"\_\_"} \quad \text{or} \quad y_j = \text{"\_\_"} \end{cases}$$

Another difference is the filling values for the initialization rows, the line zero and column zero are initialized with zero values:

$$\breve{\mathbf{F}}(i, 0) = 0 \qquad 0 \leq i \leq m$$
$$\breve{\mathbf{F}}(0, j) = 0 \qquad 0 \leq j \leq n$$

The recursive dynamic filling function, for the rest of the matrix elements is exactly the same as in the global alignment, though $w(x_i, y_j)$ weights differently.

$$\breve{\mathbf{F}}(i, j) = maximum \begin{cases} 0 \\ \breve{\mathbf{F}}(i-1, j) + \breve{w}(x_i, \_\_) \\ \breve{\mathbf{F}}(i, j-1) + \breve{w}(\_\_, y_j) \\ \breve{\mathbf{F}}(i-1, j-1) + \breve{w}(x_i, y_j) \end{cases}$$

and in this case our $\breve{\mathbf{F}}$ matrix will be:

$$\breve{\mathbf{F}} = \begin{pmatrix}
 & _ & G & A & D & Q & Z & Z & S & T & A & B & Z & C & D & X \\
_ & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
A & 0 & 0 & 2 & 1 & 0 & 0 & 0 & 0 & 0 & \boxed{2} & 1 & 0 & 0 & 0 & 0 \\
B & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & \boxed{4} & \boxed{3} & 2 & 1 & 0 \\
C & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 3 & 3 & \boxed{5} & 4 & 3 \\
D & 0 & 0 & 0 & 2 & 1 & 0 & 0 & 0 & 0 & 0 & 2 & 2 & 4 & \boxed{7} & 6 \\
X & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 3 & 6 & \boxed{9} \\
Y & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 2 & 5 & 8 \\
D & 0 & 0 & 0 & \underline{\mathbf{2}} & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 4 & 7 \\
R & 0 & 0 & 0 & \underline{\mathbf{1}} & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 3 & 6 \\
Q & 0 & 0 & 0 & 0 & \underline{\mathbf{3}} & 2 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 2 & 5 \\
Z & 0 & 0 & 0 & 0 & 2 & \underline{\mathbf{5}} & 4 & 3 & 2 & 1 & 0 & 2 & 1 & 1 & 4 \\
R & 0 & 0 & 0 & 0 & 1 & 4 & 4 & 3 & 2 & 1 & 0 & 1 & 1 & 0 & 3
\end{pmatrix}$$

Whereas in global alignment the jumping-back alignment path starts at the lower rightmost position, here the construction starts at the position holding the maximum value, in this case $\breve{\mathbf{F}}(5,14) = 9$. The jumping-back direction, is computed easily, since we only need to pick the maximum value from the *prior neighbors* and jump to that position, while in the previous algorithm more arithmetic calculations are involved. For example, we can see the justification for the jump $\breve{\mathbf{F}}_{2,11} \mapsto \breve{\mathbf{F}}_{2,10}$, because $\breve{\mathbf{F}}(2,10) = max\{0,\ \breve{\mathbf{F}}(2,10),\ \breve{\mathbf{F}}(1,10),\ \breve{\mathbf{F}}(1,11)\} = max\{0,4,1,0\} = 4$. The complete jumping-back path is marked with boxes and the sub-alignment output is:

```
A B _ C D X
A B Z C D X
```

To seek for another possible sub-alignment, one must look at the values outside the optimal path, assigned to lines and columns. These are positions where their line and column indexes do not intercept lines and columns of the optimal path elements - a new starting point in a disjoint region. In our example, we have to scan $\breve{\mathbf{F}}(i,j)$ positions such that $i \in \{6,7,...,11\}$ and $j \in \{1,2,...,8\}$. If the maximum scanned value is greater than a minimum threshold δ then a starting point for another sub-alignment can be found. Admitting that $\delta = 4$ was predefined, then $\breve{\mathbf{F}}(10,5) = 5$ is our new alignment starting point. Afterwards, the jumping-back walk works exactly the same as for the global alignment. Our second sub-alignment is marked in the matrix by using an underlined bold font. The search for sub-alignments in disjoint regions continues while unassigned lines and columns exist and as long as $max\ \{\breve{\mathbf{F}}(i,j) : free(i.j)\} > \delta$, where $free(i,j)$ represents an unassigned position. The application of this adapted version of the Smith-Waterman algoritm to our example would generate two sub-alignments:

```
ALIGNMENT 1     ALIGNMENT 2
-----------------------------------
A B _ C D X       D R Q Z
A B Z C D X       D _ Q Z
```

We conclude this section by showing some examples of aligned word sub-sequences obtained from our paraphrase corpus. The reader may check the appropriateness of this algorithm in this particular kind of sentence pairs, usually sharing some length dissimilarity, i.e. asymmetrical paraphrases (see definition 2 in Section 3.1). Let us see a case with three paraphrases in which local alignment is more appropriate. The "$algval(A_{2\times n})$" function, defined in Subsection 4.3.1, is used to provide quantitative comparisons. First we show what would be the bad effect of global alignment applied to these three paraphrases:

```
(1)  The McMartin Pre-school molestation case was the longest _ _________ _ criminal trial in
     The ________ __________ ___________ ____ ___ ___ longest , costliest , criminal trial in

      ________ U.S . history _ ___ ________ __________ ___________ ____ _ _____ ________ .
      American ___ _ history , the McMartin Pre-School molestation case , ended Thursday .

      [algval(1): 0.500]


(2)   ________ ________ After the 45-minute service , the families attended a private
      Moosally attended _____ the _________ service _ ___ ________ ________ _ _______

      reception with Moosally .
      _________ ____ ________ .

      [algval(2): 0.388]


(3)   __ _ _______ __________ The storm , ___ _ _______ __ ________ ____ packing winds of up
      21 , hitting Charleston ___ _____ , S.C . head-on at midnight with _______ winds __ up

      to 135 mph , raged into Charleston Thursday night .
      to 135 mph _ _____ ____ __________ ________ _____ .

      [algval(3): 0.358]
```

Now, for each one, we show the sub-alignments obtained through the application of the local (Smith-Waterman) alignment algorithm:

```
(1)
     a)  longest _ _________ _ criminal trial in ________ U.S . history    [algval(1a): 0.815]
         longest , costliest , criminal trial in American ___ _ history

     b)  McMartin Pre-school molestation case        [algval(1b): 1.000]
         McMartin Pre-school molestation case


(2)
     a)  the 45-minute service        [algval(2a): 0.840]
         the _________ service


(3)
     a)  winds of up to 135 mph        [algval(3a): 0.927]
         winds __ up to 135 mph

     b)  , _______ raged into Charleston        [algval(3b): 0.657]
         , hitting _____ ____ Charleston
```

It appears that these obtained sub-alignments are much more useful and make more sense for our

main objective of the identification of sentence reduction rules, than the corresponding global alignments. Thus in some situations local alignment is preferable to global alignment. In the next section we present a new method for determining which alignment algorithm should be applied at run time.

4.2 Dynamic Alignment

In the last section we presented two classically alignment algorithms for symbol sequences, the Needleman-Wunsch for global and the Smith-Waterman for locally alignment. In particular, we proposed an adaptation of this last one in order to be able to generate more than one sub-alignment from the same sequence pair. This last new method revealed to be more adequate to align our data, in many cases due to the high presence of asymmetrical paraphrases in our corpus. However, in many other cases a global alignment will comply better with the alignment task needed. So, a natural and relevant question raises:

Which alignment algorithm should be employed in our sentence word alignment problem?

Initially, we were inclined to use only the global alignment algorithm, since a full word alignment for each sentence is generated. However, we noticed that the global strategy was not appropriate for many paraphrase sentence pairs, in particular when there are syntactically valid variants, like the pair shown in the next example:

***During his magnificent speech**, the president remarkably praised IBM research.*
*The president praised IBM research, **during his speech**.*

Figure 4.5: A paraphrase sentence pair which include interchange of paraphrases.

This type of phrase interchange may occur with some frequency in a paraphrase corpus, and global alignments in such cases may align many unrelated sentence portions to word gaps, and as a result this may lead to a poor quality alignment. This has negative consequences for the induction of sentence transformation and reduction rules. The global alignment for the paraphrase example from Figure 4.5 is represented below.

```
during his magnificent speech the president remarkably praise ibm research ______ ___ ______
______ ___ ___________ ______ the president __________ praise ibm research during his speech

[algval: 0.644]
```

This is clearly a situation where our adapted local alignment algorithm (see Subsection 4.1.2) would perform better than the global one, since the local method would generate the following two sub-alignments:

```
(1) the president remarkably praised ibm research       [algval: 0.927]
    the president __________ praised ibm research

(2) during his magnificent speech       [algval: 0.886]
    during his ___________ speech
```

There are other situations, even without such syntactical alternations, where the local alignment procedure seems more adequate. For example, when one of the paraphrase sentences is considerably longer than the other one, like in the following example in Figure 4.6, a local alignment would

> ***Sayako has taken driving lessons and practiced shopping at supermarkets**, the couple has studied catalogs to choose furniture and appliances for their new home.*
>
> ***Sayako took driving lessons and practiced shopping at supermarkets**.*

Figure 4.6: A paraphrase sentence pair with relevant asymmetrical lengths.

produce a more specific alignment, which would be better suited for learning more restricted and simple sentence transformations:

```
Sayako ____ has taken driving lessons and practiced shopping at super
Sayako took ___ _____ driving lessons and practiced shopping at super

[algval: 0.874]
```

However, this kind of asymmetrical length examples may be useful to infer regularities specifying the condition in which the long sentence portions can be eliminated, which is one of our main goals. Hence, the best of "two worlds" is desirable, and ideally both global and local alignment algorithms are worth to be applied, each one in its particular cases.

The previous situations motivated and led us to search for a strategy for run-time-choice of an appropriate alignment algorithm, between the global and local methods mentioned. For that purpose, we use the concept of *exclusive link* discussed in Section 3.3. Considering a connecting line set between two sentences, we noticed that when we have high syntactic alternations, the number of line intersections is high, whereas the absence of any alternations is reflected by no line intersections. Based on this notion, we have formalized our method for algorithm selection. We refer to these intersection points as ***crossings***. An illustration is given in the Figure 4.7, where 15 crossings are identified using small squares.

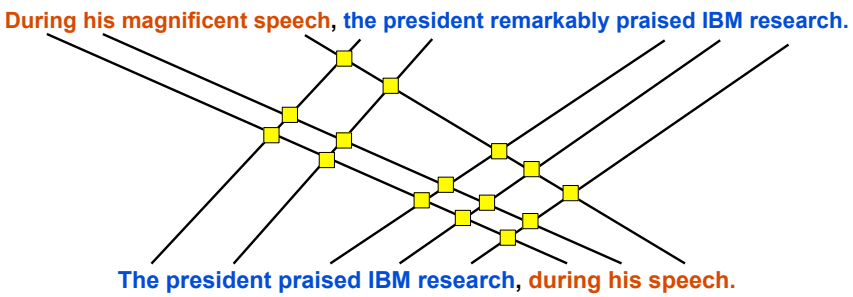

Figure 4.7: Crossings between a sentence pair.

The maximum number of crossings, between two sequences with n exclusive links is equal to $\theta = \frac{1}{2} * n * (n-1)$, which is the case when the words in the second sentence appear in reverse order

when compared with the first sentence.

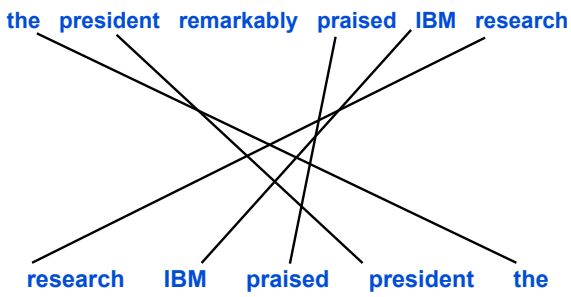

Figure 4.8: Maximum number of crossings in a complete word inversion case.

This is an extreme situation and almost never happens, because the order of words in a sentence is subject to various linguistic constraints such as the syntactical structures. The result would be a bunch of unrelated words. However, this value is a relevant upper limit for our algorithm, allowing us to calculate a kind of "reversion proportion" and take decisions at run time. Such an extreme inversion situation is illustrated in Figure 4.8, where each line has an intersection point with all the others, and so we obtain the total of $\theta = \frac{1}{2} * 5 * 4 = 10$ crossings.

In order to count the total number of intersections, which in the general case may include n lines, we start by removing a given line and count the number of intersections vanished through this operation. For the first line, $n-1$ crossings will disappear. We continue this process until no lines are left. Hence, in the second removal $n-2$ crossings disappear, in the third one $n-3$ and so on. When only two lines are left and one is removed, exactly one intersection is removed. The removal of the last line implies obviously the removal of zero crossings. Therefore, it clear that:

$$\begin{aligned}\theta &= (n-1) + (n-2) + ... + 2 + 1 + 0 \\ &= \frac{1}{2} * n * (n-1)\end{aligned}$$

With this upper bound of the number of crossings θ, we can count the number of crossings in a given case, lets say κ. This enables us to decide whether κ is greater than a certain pre-defined proportion of θ (for example whether $\kappa \geq 0.5 * \theta$) and opt for global alignment if this condition is not guaranteed. This is synthesis below in Algorithm 2.

Given two sentences, the method to calculate a crossing point takes into account the word index in the sentence, that is the position in which a word occurs within the sentence, which starts with one. For instance, in our previous example from Figure 4.7, the word "president" occurs in the sixth position, whereas in the second sentence it occurs in the second position. This connecting line is represented through a tuple $\langle w, i, j \rangle$, where the first component is the word being connected, and the second and third components are respectively the indexes of that word in the first and second

Algorithm 2 Runtime decision to chose between local and global alignment.

1: $n \leftarrow NumExclusiveLinks(S_1, S_2)$
2: $\theta \leftarrow \frac{1}{2} * n * (n-1)$
3: $\kappa \leftarrow NumCrossings(S_1, S_2)$
4: **if** $\kappa \geq 0.5 * \theta$ **then**
5: $\quad LocalAlignment(S_1, S_2)$
6: **else**
7: $\quad GlobalAlignment(S_1, S_2)$
8: **end if**

sentence. So the connecting line would be represented as $\langle president, 6, 2\rangle$, whereas the connection line relative to the word "during" would be represented as $\langle during, 1, 6\rangle$. With this representation it is fairly straightforward to verify whether any two connecting lines have an intersection point. If we have connections $\langle w, i_w, j_w\rangle$ and $\langle v, i_v, j_v\rangle$, a crossing point exists if and only if:

$$(i_w - i_v) * (j_w - j_v) < 0 \tag{4.6}$$

For instance, the previous example that involves $\langle president, 6, 2\rangle$ and $\langle during, 1, 6\rangle$ is covered by this condition, because we have $(6-1) * (2-6) < 0$, revealing that a crossing point exists for these two connections. The total number of crossings found between two sentences S_1 and S_2, corresponds to the $NumCrossings(S_1, S_2)$ function in Algorithm 2. As shown, this value is used for deciding the alignment algorithm that should be applied in a given case. The decision threshold of 0.5 in line 4 is a value that should be tuned according to the input data type being processed and also the final alignments one would prefer. In our case this value seems reasonable, however other values may also be considered.

4.3 Modeling a Similarity Matrix for Word Alignment

In bioinformatics, the DNA sequence alignment algorithms are usually guided by a scoring function, defining the probability of gene mutation. Such scoring functions are defined using matrices that relate the mutation likelihood among amino-acid characters. Two well-known and widely used matrices are the PAM[1] (Dayhoff *et al.*, 1978) and the BLOSUM[2] (Henikoff & Henikoff, 1992), which encode evolutionary approximations regarding the rates and probabilities of amino-acid mutations. The PAM family were designed to be used for a more global distant or global gene alignment investigation, whereas the BLOSUM family is more directed towards local alignment searching between evolutionarily divergent protein sequences. Similarly as for PAM matrices, several sets of BLOSUM exist, using different alignment databases, identified by numbers. BLOSUM matrices labeled with high numbers

[1] Point Accepted Mutation.
[2] BLOcks of Amino Acid SUbstitution Matrix.

are designed to compare closely related sequences, while BLOSUM matrices with low numbers are intended for comparing less related sequences. An example of a BLOSUM matrix is presented in Figure 4.9.

	Ala	Arg	Asn	Asp	Cys	Gln	Glu	Gly	His	Ile	Leu	Lys	Met	Phe	Pro	Ser	Thr	Trp	Tyr	Val
Ala	4																			
Arg	-1	5																		
Asn	-2	0	6																	
Asp	-2	-2	1	6																
Cys	0	-3	-3	-3	9															
Gln	-1	1	0	0	-3	5														
Glu	-1	0	0	2	-4	2	5													
Gly	0	-2	0	-1	-3	-2	-2	6												
His	-2	0	1	-1	-3	0	0	-2	8											
Ile	-1	-3	-3	-3	-1	-3	-3	-4	-3	4										
Leu	-1	-2	-3	-4	-1	-2	-3	-4	-3	2	4									
Lys	-1	2	0	-1	-3	1	1	-2	-1	-3	-2	5								
Met	-1	-1	-2	-3	-1	0	-2	-3	-2	1	2	-1	5							
Phe	-2	-3	-3	-3	-2	-3	-3	-3	-1	0	0	-3	0	6						
Pro	-1	-2	-2	-1	-3	-1	-1	-2	-2	-3	-3	-1	-2	-4	7					
Ser	1	-1	1	0	-1	0	0	0	-1	-2	-2	0	-1	-2	-1	4				
Thr	0	-1	0	-1	-1	-1	-1	-2	-2	-1	-1	-1	-1	-2	-1	1	5			
Trp	-3	-3	-4	-4	-2	-2	-3	-2	-2	-3	-2	-3	-1	1	-4	-3	-2	11		
Tyr	-2	-2	-2	-3	-2	-1	-2	-3	2	-1	-1	-2	-1	3	-3	-2	-2	2	7	
Val	0	-3	-3	-3	-1	-2	-2	-3	-3	3	1	-2	1	-1	-2	-2	0	-3	-1	4

Figure 4.9: The BLOSUM62 substitution matrix.

Any similarity matrix models a set of concepts accepted in a particular domain, as probable evolutionary gene mutations. Hence different matrices will generate different alignments, since such matrices provide direct and fundamental guidance for the alignment.

Subsequently, this inspired us to develop a model that governs the likeliness of "word mutation". Such a model could then be used to guide the word alignments between our paraphrase sentences and could produce more relevant material for the learning phase. We note that different authors employ synonyms or even more simpler lexical similarities, like plurals, or different verb modes in their formulations. A word mutation model could combine different types of linguistic features: lexical, syntactical or even semantic. In our work, limit our proposal to a simple model which is based exclusively in lexical features. However, in the future more complex combinations could be experimented with.

Our main concern was to define a simple lexical mutation function that would reward naturally related word transformations, whereas unlikely word transformations would be significantly penalized. For example, it seems more likely to admit a lexical mutation between the words *spirit* and *spiritual* than between the words *spiritual* and *hamburger*, which would not occur much in a paraphrase word alignment.

The first idea we had to introduce a word mutation function was to consider the well-known *Edit*

Distance (Levenshtein, 1966) and adapt it for word alignment by introducing a negative reward. The reason for taking a negative value is to comply with the content type of a mutation matrix, since it represents costs, not rewards, for allowing gene alignments. So when using the negative *Edit-distance* for a word pair $\langle w_i, w_j \rangle$, the lower the negative *Edit-distance* value, the more unlikely the word w_i would be aligned with word w_j.

However, despite many good alignments generated in some early experiments with this *negative-edit-distance* function, we found that it was not appropriate in many situations, as it lead to alignments between very different words, like for example: $\langle total,\ israel \rangle$, $\langle fire,\ made \rangle$, $\langle troops,\ members \rangle$.

The main reason why this happens is because the *edit-distance* returns relatively small values, incapable of sufficiently penalizing morphological differences, like those listed before, in order to inhibit their alignment. In terms of bioinformatics concept, this means that our model is still giving relatively high mutation probability scores for such pairs. Another problem with this function is related to the fact that it does not make any distinction between long and small word sizes, for instance the pairs $\langle in, by \rangle$ and $\langle governor, governed \rangle$ have exactly the same *Edit-distance* equal to 2, despite the huge differences between the pairs.

After some further research, we have defined a more adequate function to model lexical word mutations, avoiding the previous difficulties - the $costAlign(w_i, w_j)$ function. Thus, we proposed a new word alignment scoring function in Equation 4.7) which is an adaptation of the *Edit-distance* function, by dividing it by a normalization factor, the $maxseq(w_i, w_j)$ function, which discriminates among word lengths.

$$costAlign(w_i, w_j) = -\frac{edist(w_i, w_j)}{\varepsilon + maxseq(w_i, w_j)} \tag{4.7}$$

The $edist(w_i, w_j)$ is the string edit distance, and the $maxseq(w_i, w_j)$ function calculates a normalized maximum common sequence value between two words. That is, it represents the length of the longest common subsequence of the two words divided by the maximum length value. For example, the longest common subsequence for the pair $\langle reinterpretation, interpreted \rangle$ is "$interpret$", with length equal to 9, thus:

$$maxseq(reinterpretation, interpreted) \ = \ \frac{9}{max\{16, 11\}} = 0.5625$$

On the other hand the pair $\langle hamburger, spiritual \rangle$ has a maximum common sequence equal to 1 (just one letter: "a" or "u" or "r") and so we would obtain:

$$maxseq(hamburger, spiritual) \ = \ \frac{1}{max\{9, 9\}} = 0.1111$$

In Equation 4.7 the ε represents a small value[1] near zero, acting like a "safety hook" against divisions by zero whenever $maxseq(w_i, w_j) = 0$. This happens when we have completely different words

[1] We took $\varepsilon = 0.01$ in our experiments.

that do not share any character in common. We named our function as "$costAlign$", instead of, for example, "$scoreAlign$", since it looks more like a cost function, with values always less than zero. In fact its maximum value is zero, when we have two identical words, however in such cases a positive value should be assigned, rewarding the occurring match. This is exactly what we make, according to the function $w(x_i, y_j)$ (4.8), explained later.

Table 4.2: Comparing two word mutation functions.

word 1	word 2	-edist	costAlign
rule	ruler	-1	-1.235
governor	governed	-2	-2.632
pay	paying	-3	-5.882
reinterpretation	interpreted	-7	-12.227
hamburger	spiritual	-9	-74.312
in	by	-2	-200.000

In Table 4.2 we present several cases, comparing the simply negative *edit-distance* and the defined $costAlign(w_i, w_j)$ function for word alignment. These examples demonstrates the advantages of the $costAlign(w_i, w_j)$ over negative *edit-distance*, specially in the case of small word lengths. For instance, we note that although the pairs $\langle in, by \rangle$ and $\langle governor, governed \rangle$ have exactly the same negative *edit-distance* value, their $costAlign$ values are hugely different, giving no alignment chance in this case. With our $costAlign$ function the word mutations are more admissible for longer words, whereas much restricted for the short ones.

The $costAlign(w_i, w_j)$ function defines our word mutation matrix for different words, in which lexically similar words are admitted to be aligned. We finalize this chapter by showing a paraphrase alignment example where $costAlign(w_i, w_j)$ is employed, yielding the alignment of the paraphrase words. In our case, considering the type of values possibly returned by the $costAlign(w_i, w_j)$ function, and after some experiments, we decided to assign a reward of 10 for a match. Therefore in our case the $w(x_i, y_j)$ function was defined as follows:

$$w(x_i, y_j) = \begin{cases} 10 & \text{if } x_i = y_j \\ -3 & \text{if } x_i = \text{"\_\_"} \quad \text{or} \quad y_j = \text{"\_\_"} \\ costAlign(x_i, y_j) & \text{if } x_i \neq y_j \end{cases} \tag{4.8}$$

As it can be seen by looking at the second brach, the gap penalty used was equal to -3. With this parameterization suppose one want to align the paraphrase sentences shown in Figure 4.10 then the

S_1 : *Gilbert was the most intense storm on record in the west.*

S_2 : *Gilbert was the most intense hurricane ever recorded in western hemisphere.*

Figure 4.10: A paraphrase sentence pair with relevant asymmetrical lengths.

alignment matrix dynamically constructed was equal to:

$$
\left(\begin{array}{rrrrrrrrrrrrr}
 & \_\_ & \textit{Gilbert} & \textit{was} & \textit{the} & \textit{most} & \textit{intense} & \textit{storm} & \textit{on} & \textit{record} & \textit{in} & \textit{the} & \textit{west} \\
\_\_ & \mathbf{0.0} & -3.0 & -6.0 & -9.0 & -12.0 & -15.0 & -18.0 & -21.0 & -24.0 & -27.0 & -30.0 & -33.0 \\
\textit{Gilbert} & -3.0 & \mathbf{10.0} & 7.0 & 4.0 & 1.0 & -2.0 & -5.0 & -8.0 & -11.0 & -14.0 & -17.0 & -20.0 \\
\textit{was} & -6.0 & 7.0 & \mathbf{20.0} & 17.0 & 14.0 & 11.0 & 8.0 & 5.0 & 2.0 & -1.0 & -4.0 & -7.0 \\
\textit{the} & -9.0 & 4.0 & 17.0 & \mathbf{30.0} & 27.0 & 24.0 & 21.0 & 18.0 & 15.0 & 12.0 & 9.0 & 6.0 \\
\textit{most} & -12.0 & 1.0 & 14.0 & 27.0 & \mathbf{40.0} & 37.0 & 34.0 & 31.0 & 28.0 & 25.0 & 22.0 & 19.0 \\
\textit{intense} & -15.0 & -2.0 & 11.0 & 24.0 & 37.0 & \mathbf{50.0} & 47.0 & 44.0 & 41.0 & 38.0 & 35.0 & 32.0 \\
\textit{hurricane} & -18.0 & -5.0 & 8.0 & 21.0 & 34.0 & \mathbf{47.0} & 44.0 & 41.0 & 38.0 & 35.0 & 32.0 & 29.0 \\
\textit{ever} & -21.0 & -8.0 & 5.0 & 18.0 & 31.0 & \mathbf{44.0} & \mathbf{41.0} & \mathbf{38.0} & 35.0 & 32.0 & 29.0 & 26.0 \\
\textit{recorded} & -24.0 & -11.0 & 2.0 & 15.0 & 28.0 & 41.0 & 38.0 & 35.0 & \mathbf{35.4} & 32.4 & 29.4 & 26.4 \\
\textit{in} & -27.0 & -14.0 & -1.0 & 12.0 & 25.0 & 38.0 & 35.0 & 36.0 & 33.0 & \mathbf{45.4} & \mathbf{42.4} & 39.4 \\
\textit{western} & -30.0 & -17.0 & -4.0 & 9.0 & 22.0 & 35.0 & 32.0 & 33.0 & 30.0 & 42.4 & 39.4 & \mathbf{37.2} \\
\textit{hemisphere} & -33.0 & -20.0 & -7.0 & 6.0 & 19.0 & 32.0 & 29.0 & 30.0 & 27.0 & 39.4 & 36.4 & \mathbf{34.2}
\end{array}\right)
$$

and the jumping back path followed after constructing the alignment matrix is marked in bold face, which determines the sequence alignment. The resulting alignment for the two paraphrase sentences was:

```
Gilbert was the most intense _________ ____ storm on record   in the west    __________
Gilbert was the most intense hurricane ever _____ __ recorded in ___ western hemisphere
```

We note that in this example we got heterogeneous word pairs aligned, as for example $\langle recorded, record \rangle$ and $\langle western, west \rangle$. These two cases show that the $costAlign(w_i, w_j)$ function play it's role in the construction of the alignment matrix. In the first case we have $costAlign(recorded, record) = -2.632 \approx -2.6$ and we have a north-west jumping back, as $35.4 = \mathbf{F}(8,8) = 38.0 + (-2.6)$. The same occurs at $\mathbf{F}(10,11)$, where $costAlign(western, west) = -5.16$.

4.3.1 The Quantitative Value of an Alignment

Several times, when we are discussing about paraphrase alignments we subjectively compare different alignments, stating that a given one looks better that another one. In order to have a more objective criteria to compare alignments, at least at a surface level[1], we define here a function for that, based on the word aligning function have used on the alignment construction - Equation 4.8.

To define this function let us consider an alignment as an $2 \times n$ matrix of tokens, where each line contains the words of each sentence. For example:

$$
A_{2\times 7} = \begin{bmatrix} \texttt{Maxwell} & \texttt{plans} & \texttt{\_\_} & \texttt{raising} & \texttt{\_\_\_\_\_\_} & \texttt{\$448} & \texttt{million} \\ \texttt{Maxwell} & \texttt{planed} & \texttt{to} & \texttt{raise} & \texttt{around} & \texttt{\$448} & \texttt{million} \end{bmatrix}
$$

Then our lexical alignment evaluation function $algval(A_{2\times n})$ is defined as shown below in equation 4.9.

$$
algval(A_{2\times n}) = \frac{1}{2} + \frac{\sum_{j=1}^{n} \sigma(A_{1j}, A_{2j})}{10 \cdot (n_1 + n_2)} \tag{4.9}
$$

The $\frac{1}{2}$ and $\frac{1}{10\cdot(n_1+n_2)}$ values are normalization factors in order to have as a result $0 \leq algval(A_{2\times n}) \leq 1$, where n_1 and n_2 are respectively the number of non-empty tokens in sentences A_1 and A_2. The

[1] Only lexical relations are considered.

token match function - $\sigma(w_1, w_2)$ - is based on the previously defined $w(x_i, y_j)$ function from Equation 4.8 and calculates the similarity value between two tokens. Thus we have:

$$\sigma(x,y) = \begin{cases} 10 & \text{if } x = y \\ -3 & \text{if } x = \text{"\_\_"} \quad \text{or} \quad y = \text{"\_\_"} \\ \frac{10}{-costAlign(x,y)} & \text{if } x \neq y \end{cases} \tag{4.10}$$

The value 10 in the third branch of this function is related with the positive reward giving to a match, as well as the 10 appearing in the denominator of the *algval* function.

4.4 System Execution

Before giving final remarks and finishing this chapter, we present here some implementation and execution details of our system, until the alignment phase, which comprehends steps one and two[1]. Further details about subsequent system steps are given in Chapter 6.

Excepting the *Aleph* execution module, all the other modules shown in Figure 1.2 were implemented using the *Java* programming language. The conceptual presentations of paraphrase extraction and alignment were already done. Here we show system execution examples involving the extraction and alignment of paraphrases from *web news stories*. More deeply implementation details about some relevant issues/algorithms are presented in Appendix B.

4.4.1 Extraction of Web News Stories

According to the scheme in Figure 1.2, the first step consists in automatically gather a collection of *web news stories*, organized in clusters of events. Each cluster contains several news about a single event, for example: "The presidential speech in the state of the union". Each *news* document comes from a different electronic source. A quite handy place, on the Web, to look for such data is naturally the "*Google News*" web site[2].

From a given *news event*, a set of related *news stories* links are followed. The launch of the web news extractor is made through the execution of the "`GoogleNewsSpider`" class, which is included in the `GNewsSpider.jar` package. By default the Google's English version is crawled. Other *Google News* language versions can be indicated in a configuration file (with name "`.gnews_spider`"), in the user's home directory, as well as other configuration parameters for news extraction can also be included in this file. The user may even supply this configuration file directly in the command line, through the "`-conf`" parameter. The syntax and parameters of the `GoogleNewsSpider` program is listed below.

[1]See the system scheme in Figure 1.2.

[2]URL: http://news.google.com [November 2010]

Figure 4.11: A link pointing to a cluster of web news stories related to a given event.

```
$ java GoogleNewsSpider
    HELP:
       java <P> [help] [-test] [-clustptxt <string>] [-serverurl <string>] [-conf <file>]
            help:........... Print this help.
            clustptxt:...... The string pattern to access the web news cluster.
            conf:........... To supply the configuration file.
            serverurl:...... The server url starting point, usualy: http://news.google.com
            test:........... Run in test mode, with more output details.
```

The whole set of *news* data used in this research was collected with this program, scheduled to automatically run once a day in a server[1], collecting *news* data (320 files) between the dates from 2005/11/05 until 2008/12/31. It extracted a total of 1.4 GB of text data, storing these in XML files, one file for each execution. Figure 4.12, in the next Subsection gives an idea about the structure of one such data file.

4.4.2 Paraphrase Extraction and Alignment

The second step of our system scheme (Figure 1.2) consists in the extraction of paraphrastic sentences and their subsequent word alignment, from the set of *web news stories* data files, collected in the previous step. This operation is executed through the "`java GenAlignParaphCorpus`" command, which corresponds to a *Java* class contained in the "`Factory.jar`" package. A simple execution of this command, without supplying any parameter, will reveal a small command help, showing the set of parameter possibilities.

[1] A machine running a POSIX operating system, with access to the *World Wide Web*.

```
<?xml version="1.0"?>
<news-clusters>

<cluster i="1"  url="http://news.google.com/?ncl=1285686762&hl=en&topic=h">
   <new i="0"  url="http://www.youtube.com/results?sa=N&tab=n1">
   </new>
   <new i="1"  url="http://www.washingtonpost.com/wp-dyn/content/article/2008/12/30 ...">
       President-elect  spent yesterday afternoon in Honolulu, going to the zoo with
       his daughters and visiting his high school campus. And yet, he couldn't escape the
       political melodrama unfolding more than 4,000 miles away. While Obama vacationed,
       some of the main characters from his political past took turns starring in a
       bizarre Chicago news conference. First to the lectern was embattled Illinois
       ...
   </new>
   ...
   <new i="75"  url="http://firstread.msnbc.msn.com/archive/2008/12/30/1727681.aspx">
       First Read is an analysis of the day's political news, from the NBC News political
       unit. First Read is updated throughout the day, so check back often. But can the
       Senate do that? There's reason to think not, and here's why. In January 1967, Adam
       Clayton Powell of New York was re-elected by the Harlem district he represented
       since 1942, despite allegations that he had misused official travel funds and made
       improper payments to his wife.?The House, invoking a provision of ...
       ...
   </new>
</cluster>

...
...
...

<cluster ... >
   ...
</cluster>

</news-clusters>
```

Figure 4.12: Illustration selected from an xml web news file, produced by the "GoogleNewsSpider" program.

```
$ java GenAlignParaphCorpus
        SYNTAX:
           java <P> -dir <folder>  [-out <fout>] [-m <metric>] [-aln <type>] [-thres <value>]
                    dir -----> the directory of xml web news stories files
                    fout ----> the xml output file, example: pac.xml
                    metric --> the paraphrase detection metric:
                               {sumo,  ngram,   xgram,
                                bleu,  entropy, gauss}
                    aln -----> the alignment method: {nwunsch, swaterman, dynamic}
                    thres----> the decision threshold, example: 0.5, 0.75, etc.
```

This command was designed to process a directory containing a set of XML files with *web news stories*, similar to the one illustrated in Figure 4.12. The "`<P>`" tag represents the class name, and the set of parameters is briefly described. The "`dir`" and "`fout`" parameters indicate respectively the input and output pointers. The output is also an XML file, containing the set of all paraphrases extracted and their corresponding generated alignments. The "`metric`" parameter let the user specify the sentence similarity function to be applied, and the "`thres`" parameter is closely related with this last one, as it establishes the boundary value for paraphrase extraction decision. The "`algn`" parameter sets the method that will be applied on sentence alignment, which according to sections 4.1 and 4.2 can be: *Needleman-Wunsch* (`nwunsch`), *Smith-Waterman* (`swaterman`), or dynamic (`dynamic`). An excerpt from a generated file with aligned paraphrases is listed in Figure 4.13.

```
<?xml version="1.0" encoding="UTF-8"?>
<root>
<group>
   ...
   <paraph id="850" alg="NW">
      <nx>(4, 28, 0.864793)</nx>
      <s1>sales were down 34.5 percent at the ford motor company, 32.8 percent at ...
      <s2>sales were down 24 percent at honda motor co ltd, 32 percent at toyota ...
      <sa1>sales were down __ 34.5 percent at _____ the ford motor __ company ...
      <sa2>sales were down 24 ____ percent at honda ___ ____ motor co _______ ...
   </paraph>
   ...
   <paraph id="1063" alg="NW">
      <nx>(12, 35, 0.903677)</nx>
      <s1>this woman is a serious politician.</s1>
      <s2>she is a politician not a leader.</s2>
      <sa1>___ this woman is a serious politician ___ _ ______ .</sa1>
      <sa2>she ____ _____ is a _______ politician not a leader .</sa2>
   </paraph>
   ...
   <paraph id="26160" alg="NW">
      <nx>(41, 57, 0.913970)</nx>
      <s1>obama would require that all employers either offer health benefits to their ...
      <s2>obama's plan would require large employers to either offer insurance or ...
      <sa1>obama   ____ would require _____ that all employers __ either offer ...
      <sa2>obama's plan would require large ____ ___ employers to either offer ...
   </paraph>
   ...
</group>
...
<group>
   ...
</group>
</root>
```

Figure 4.13: An xml file of aligned paraphrases, showing the initial part of three examples.

Looking at the excerpt in Figure 4.13, we can see that paraphrases extracted from a cluster of related *web news stories* are stored within the same "`<group>`" tag. Each paraphrase is delimited by a "`<paraph>`" tag, which has two parameters, one is a sequential identifier (`id`) and the other one ("`alg`") contains the alignment algorithm applied, "`NW`" for *Needleman-Wunsch* and "`SW`" for *Smith-Waterman*. The "`<nx>`" tag contains a 3-ary tuple, where the first two arguments hold the sentence identifiers from where the sentences were extracted, and the third argument is the calculated sentence similarity for that sentence pair. The "`<s1>`" and "`<s2>`" tags contains naturally the paraphrasic sentences, and "`<sa1>`" and "`<sa2>`" their corresponding alignments.

To give an idea about the amount of data produced by this system's module, from a set of just three files of *web news stories*, collected from three different days, near 64 000 paraphrases were extracted and aligned. This represents 53.4 MB of textual data.

4.4.3 Class Hierarchy Diagram

We finish this subsection by showing in Figure 4.14 a **class diagram** containing the key classes involved in our data representation. This diagram is expressed using UML notation (OMG, 2010), where rectangles represent *classes* and the connecting arrows represent the relationships among them. For

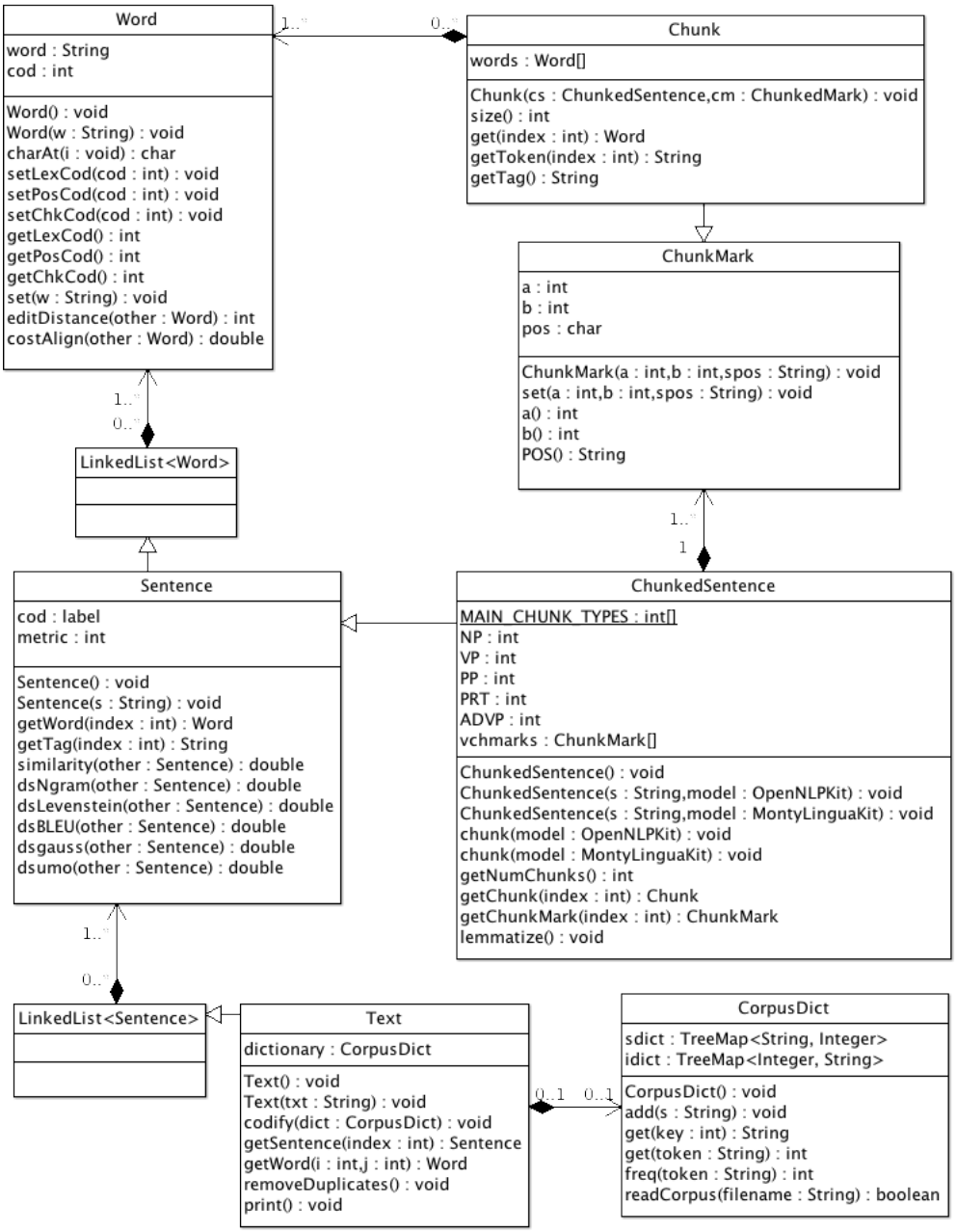

Figure 4.14: A Selection of the main classes designed for text data representation.

any rectangle we have three main blocks. The first line contains the *class name* and in the second block we have the *class attributes*. The third block contains the class *methods*.

There are only two types of connecting lines in this diagram. The line with a *hollow triangle* shape (△) represents the *generalization/specialization* relationship, with the class directly connect to the triangle extremity being a generalization (*super class*) of the other one (*derived class*). The other line type, with a *solid diamond* shape at one of its extremities, represents a composition ("includes

a") relationship between the two classes involved. The class next to the diamond is a *container* of the other one (*contained*). The labels "1", "0..1" (zero or one), "0..*" (zero or many), and "1..*" (one or many), indicate the *cardinality participation* in the relationships. For example in the relationship between "LinkedList<Word>" and "Word" classes, the former contains one or more ("1..*") instances of the latter.

It can be seen that "Word" is our basic class, representing naturally a textual word, which can be any sequence of alphanumeric characters (a Java String), excluding obviously the space character. This class is a kind of superclass for the "Sentence" which in turn is also a kind of superclass for the "Text" class.

As it can be seen there is not a direct class derivation among these three classes, since "Sentence" is derived from a linked list of words, and "Text" is derived from a linked list of sentences. The class "CorpusDict" is a dictionary, generated from corpora, mapping word tokens into integer codes, for the sake of computation efficiency.

The "ChunkedSentence" is a specialization from the "Sentence" class to handle chunked sentences and it uses an array of "ChunkMark" instances, which is a class that stores the initial (a) and final (b) chunk word positions, in a sentence, and the chunk type, for example: noun-phrase (NP), and verb-phrase (VP). Sentence chunking and *part-of-speech* tagging may be processed by using one of the two well-known tools, from the "*OpenNLP*"[1] or "*MontyLingua*" (Liu, 2004) packages, as it can be seen in the diagram, by looking at the "ChunkedSentence" class constructors. In this work, the "*OpenNLP*" was exclusively employed, as it seemed a bit more accurate[2], though both packages perform close with almost the same labeling output.

4.5 Final Remarks

In this chapter we described that the goal for automatically generate learning instances for sentence reduction rule induction led us to investigate techniques for aligned paraphrasic corpora construction, without direct human intervention and as efficient and effective as possible. After automatic paraphrase identification and extraction from web news corpora, techniques for paraphrasic sentence alignment were investigated. Algorithms from the field of *Bioinformatics*, used for DNA sequences alignment, were conveniently adapted for word alignment between paraphrasic sentences. In particular the *Needleman-Wunsh* (NW) and *Smith-Waterman* (SW) algorithms were respectively employed, for global and local sentence alignment.Two new issues were introduced. First a function enabling the alignment of lexical similar, words was established, and secondly a new method for choosing between NW and SW during run-time.

[1] http://opennlp.sourceforge.net [October 2010]

[2] Qualitative comparisons of several examples led us to that conclusion.

Equivalently to the "mutation matrices" used in *Bioinformatics* for DNA sequences alignment, we have defined a function establishing the likelihood of lexically similar word alignment - the $costAlign(w_i, w_j)$, presented in Subsection 4.3. It resembles a kind of "word mutation" occurring between the paraphrasic sentences, where "mutations" are more likely to occur between longer words. Our word mutation function is based exclusively in lexical features, however in the future more linguistically complex characteristics may be considered, as fore example synonymy.

We observed that for some paraphrases the local alignment (SW) would be much more appropriate than a full alignment. So, we introduced an efficient method to choose along run-time the right alignment algorithm. This method is based on a pre-analysis of the exclusive lexical links existing between the paraphrase sentences.

The work presented here and in the last chapter allows us to automatically collect paraphrases from *web news stories* and align their words, thereby generating corpora of aligned paraphrase pairs. These data is used to learn sentence reduction rules in our subsequent step. So, the next chapter introduces a number of fundamental theoretical aspects related to the learning system used - an *Inductive Logic Programming* system named *Aleph*. Afterwards in Chapter 6 we described our implementation of the sentence reduction rules learner, based on *Aleph*.

Regards our alignment work we have achieved a scientific contribution published in the *ACL 2007* conference proceedings and presented at the *RTE Workshop* (Cordeiro *et al.*, 2007d).

Chapter 5

Inductive Logic Programming

"*Give me a place to stand and a lever long enough and I will move the world.*"

Archimedes, 250 BC

During the last two decades, remarkable achievements have been reached in the field of *Machine Learning* research, from applications of *Case Base Reasoning*, *Neural Networks* to *Decision Trees* and *Inductive Logic Programming*. In this chapter we introduce the basic knowledge of *Inductive Logic Programming* that we have used in our architecture that involves induction of reduction rules.

Induction can be seen as a process, starting from observations and generating general patterns that underly these observations. Humans are accustomed to construct theories about the surrounding world. We tend to formulate generalizations very easily, from a very few examples, which sometimes leads to erroneous conclusions. But we also tend to specialize or even correct previously induced theories if contrary examples from the real world are seen. Hence, induction is a kind of bottom-up reasoning flowing from the specific towards the general, where successive orders of generalization may be incrementally constructed, as one sees the world passing events.

5.1 Machine Learning

In *Machine Learning*, a theory or an hypothesis is the result obtained from an inductive process by observing a set of examples from a given phenomenon. There are two main learning categories: *supervised* and *unsupervised*. In the first one the learner is supplied with a set of labeled examples called training instances. Such instances are usually represented in the form of an ordered pair as:

$$\langle \overrightarrow{X},\ h(\overrightarrow{X})\rangle \quad \text{where } \overrightarrow{X} = \langle x_1, x_2, ..., x_n\rangle$$

Vector $\overrightarrow{X}$ is the feature vector which contains the values of a given set of n attributes characterizing the concept to be learned in that particular case. The h function represents the hypothesis being learned. In some cases $h(\overrightarrow{X})$ consists in a class value from a discrete set, and we say that we have a *classification* problem. Whenever $h(\overrightarrow{X})$ is a real value we have a *regression* problem. A classical illustrative example of classification learning is the "Play tennis" problem from Quinlan (1986): In this

Table 5.1: Quinlan's "Play Tennis" learning example.

Day	x_1 Outlook	x_2 Temperature	x_3 Humidity	x_4 Windy	$h(\overrightarrow{X})$ Class
1	sunny	hot	high	weak	no
2	sunny	hot	high	strong	no
3	overcast	hot	high	weak	yes
4	rainy	mild	high	weak	yes
5	rainy	cool	normal	weak	yes
6	rainy	cool	normal	strong	no
7	overcast	cool	normal	strong	yes
8	sunny	mild	high	weak	no
9	sunny	cool	normal	weak	yes
10	rainy	mild	normal	weak	yes
11	sunny	mild	normal	strong	yes
12	overcast	mild	high	strong	yes
13	overcast	hot	normal	weak	yes
14	rainy	mild	high	strong	no

case we have four attributes and 14 learning instances. Several machine learning algorithms may be applied to induce the hypothesizes covering the observed dataset. One originally applied was the ID3[1] which produces the hypothesis in the form of a decision tree as shown below:

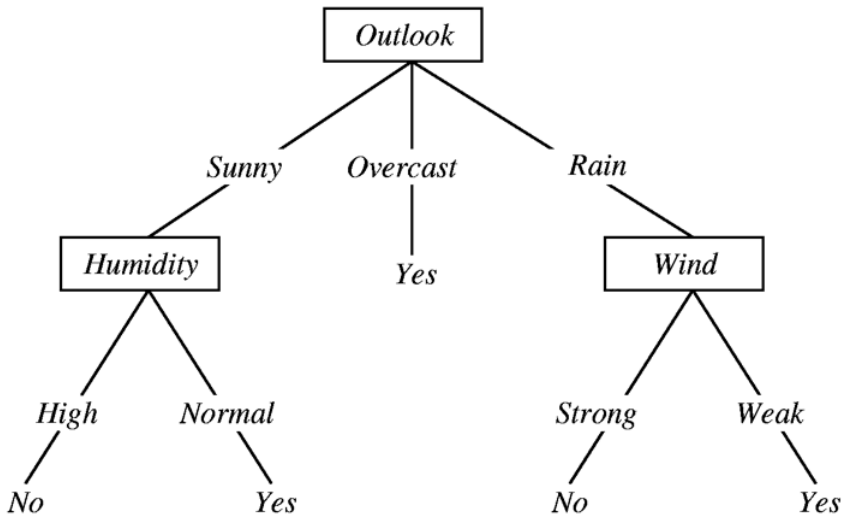

Figure 5.1: A learned decision tree for the *play tennis* problem.

[1]Which stands for "Induction of Decision Tree".

Such a tree may be converted in a set of *if-then* rules by considering each path from the root to the leafs. In this case we would have the following five rules, which represents a theory to explain which are the weather conditions for playing tennis:

$$\textbf{If } outlook = overcast \textbf{ Then } \quad play = yes$$

$$\textbf{If } outlook = sunny \textbf{ And } humidity = high \textbf{ Then } \quad play = no$$

$$\textbf{If } outlook = sunny \textbf{ And } humidity = normal \textbf{ Then } \quad play = yes$$

$$\textbf{If } outlook = rain \textbf{ And } wind = strong \textbf{ Then } \quad play = no$$

$$\textbf{If } outlook = rain \textbf{ And } wind = weak \textbf{ Then } \quad play = yes$$

In supervised learning the training instances are usually labeled by humans, implying necessarily considerable amounts of manual work with all its inherent limitations and imperfections. On the other hand, in *unsupervised* learning there are no labeled training instances. This learning method organizes the "observed" dataset into several main categories being these also discovered throughout the learning process. One concrete example such learning type is *clustering*.

There are also *semi-supervised* learning (Zhu, 2005), where labeled and unlabeled data are combined in the learning process. The approach which we have followed resembles more this learning method. In our system the learning instances are automatically cerated (and labeled) by the system. Thus, we use supervised-like instances without manual intervention.

Exhaustive *machine learning* paradigms and algorithms may be found in wide a set of bibliographic material, from which we highlight here Mitchell (1997) and Witten & Frank (2005). We are now going to motivate and introduce here a more powerful learning paradigm, employed in this work - *Inductive Logic Programming* (ILP).

A great number of *machine learning* tasks are based on what is known as *propositional learning*, the application of the ID3 algorithm to the "playing tennis" problem, previously shown, is just example. In such a learning paradigm, the learning instances are characterized by a set of features, also named as *attributes*, and the learned theory can be described as a set of specific rules from propositional logic (Appendix A.1). In this context a rule is a propositional logic formula formed by two main components: the *antecedent* and the *consequent*. The former is usually formed by the conjunction or disjunction of other several subparts (conditions), while the latter gives the classification. The two parts are connected through the (=>) connective in the form of "*antecedent* => *consequent*", which naturally also represents an "**If** *antecedent* **Then** *consequent*" rule, similar to the examples given for the "playing tennis problem".

However, a different learning paradigm, known as *relational learning*, can be used, in which we have multiple relations representing several entities and the concept being learned is also represented

with a relation. Relations are expressed trough n-ary tupples, similar to n-ary predicates in *First Order Logic* (FOL) (Appendix A.2). For example, continuing to use the "playing tennis" domain we may consider the relations `player` and `game` as follows:

```
player(id, name, weight, height)
   player(101, john, 85, 173)    player(102, mary, 71, 171)
   player(103, anne, 57, 155)    ...

game(num, id1, id2, result)
   game(345, 101, 102, 6:7)   game(347, 101, 103, 7:1)
   game(346, 102, 103, 7:2)   ...
```

The climatic features used in the propositional formulation will also become relations, for example:

```
wind(345, weak)     outlook(345, sunny)      temp(345, hot)    ...
wind(346, strong)   outlook(346, overcast)   temp(346, normal) ...
wind(347, normal)   outlook(347, overcast)   temp(347, mild)   ...
```

The learning concept can be `play(X,Y,G)`, that is, in which conditions two players X, and Y, accept to play a game G. A possible learned rule is:

```
play(X,Y,G) <=
   wind(G, strong)  AND
   player(X, Xname, Xweight, Xheight)  AND
   player(Y, Yname, Yweight, Yheight)  AND
   |Xheight - Yheight| < 20
```

which can be read as "*two players accept to play in a windy day if their height differences is less than 20 cm*".

The type of concept learning in the relational formulation has a much greater expressive power and normally it is very hard, if not impossible in some situations, to reformulate it in propositional learning (Kramer, 2000). In terms of expressive power and knowledge representation, the main difference between these two learning paradigms is the same as the one that exists between *propositional logic* and *first order logic*, briefly presented in appendixes A.1 and A.2, respectively. Thus, in this work we have decided to use a relational learning based approach, known as *Inductive Logic Programming*, to learn sentence reduction rules.

Inductive Logic Programming (ILP) is a mathematical formalization characterizing the inductive reasoning, which is applicable in almost any learning case, from simple every day life situations to complex scientific problems. Our system's third step (fourth module in Figure 1.2) is mainly based on ILP, in particular a computational implementation of it (the *Aleph* system, presented in Subsection 5.4). Since it is a very important step in our work, we have decided to dedicate this chapter to cover some basic ILP aspects. ILP is based on *Logic Programming* which in turn is based on and uses many aspects from *Mathematical Logic*. Thus, we have also decided to include a small set of notes from this discipline in the appendixes, just for the reader who wants quickly remind some concepts mentioned in the text. Among other important things, *Logic* defines rigorously the notion o deduction, which can be seen as the induction's inverse mechanism. In Section 5.3 we describe the main ILP

concepts and in Section 5.4 we present the *Aleph* system. But first we have to present the subject of Logic Programming (Section 5.2), which is the basic language for ILP.

5.2 Logic Programming

Logic Programming (LP) is a *Computer Science* discipline directly based on *First Order Logic* (FOL) and a great deal of aspects and terminology in LP comes from it. LP is even presented as an important FOL subset (Lavrac, 2001). A general FOL description, touching the main relevant aspects of LP, is given in Appendix A.2 for the interested reader. In this section we present basic LP concepts and terminology that are employed in ILP, which is presented in the next section.

ILP produces knowledge in the form of *logic programs*, which are formed by a special type of FOL rules named as clauses. Let us consider the syntactical valid FOL formulas, called *well formed formulas* (*wffs* - Definition 9), then a ***clause*** is a *wff* formed by the disjunction of a set of literals. A ***literal*** is an atomic formula or its negation, such as: $L_1 = p(X_0, a)$, $L_2 = \neg p(X_0, a)$, where a negated literal is also called a ***negative literal***, and a non negated one a ***positive literal***. For example, assuming that we have a set of literals $\{L_1, ..., L_n\}$, either positive or negative, having k variables, being universally quantified, say $X_1, ..., X_k$, then the following expression is a clause:

$$\forall_{X_1}, ..., \forall_{X_k} \quad L_1 \vee ... \vee L_n \tag{5.1}$$

For the sake of simplicity, the universal quantifiers are usually omitted, and so in a clause variables are interpreted as being universally quantified. Since clauses are in fact FOL rules, an implication connective is normally employed in a right-left direction, making a separation between the ***clause head***, at the left hand side, and the ***clause body*** in the right hand side. For example, assuming that we have m positive literals $H_1, ..., H_m$ and n negative literals $\neg B_1, ..., \neg B_n$, then our clause can be written as follows:

$$H_1 \vee ... \vee H_m \;\Leftarrow\; B_1 \wedge ... \wedge B_m \tag{5.2}$$

Variables occurring in the clause head are still universally quantified whether those occurring exclusively in the clause body are existentially quantified, due to the negation operation, since we have:

$$H_1 \vee ... \vee H_m \;\Leftarrow\; B_1 \wedge ... \wedge B_m \quad \equiv \quad H_1 \vee ... \vee H_m \vee \neg B_1 \vee ... \vee \neg B_m \tag{5.3}$$

A clause having at most one positive literal is called a **Horn clause** and a Horn clause with exactly one positive literal is called a **definite clause** or a rule, and is normally expressed as:

$$H \;\Leftarrow\; B_1 \wedge ... \wedge B_m \tag{5.4}$$

where H is obviously the formula's unique positive literal. A **program clause** is a more general rule where negated literals are also admissible in the rule's body. More precisely, a program clause is

a clause of the form of 5.4, where any B_i may be a positive or negative literal. If a given B_i is a negative literal then it is written in the form of "$not\ \hat{B}_i$", where $\hat{B}_i$ is a positive literal.

Program clauses are the "building blocks" for any logic program. A definite clause without a body is called a ***fact***, which are meant to be always true, since it is formally represented as $H \Leftarrow true$. Usually it is written just as "H".

A ***predicate*** definition is a set of clauses having the same head, and a set of clauses is called a ***clausal theory***, representing the conjunction of their clauses. A set of program clauses is called a ***logic program***. For example, in Figure 5.2 it is shown a simple logic program, with only one predicate and two clauses, defining the list membership, that is, what it means for a given element to be member of a list of elements.

```
member(X, [X | Tail]).
member(X, [Y | Tail])  <=  member(X, Tail).
```

Figure 5.2: The "member" predicate, with two clauses and being defined recursively.

The expression "`X | Tail`" represents a list decomposition into its first element, `X`, and the rest of the list represented by the variable `Tail`. In this case the predicate `member(X,L)` assumes a particular semantic, meaning that `X` belongs to the list "`L`". This is a recursive definition, stating that an element `X` belongs to a list of elements ("is a member of") if it is its first element (first clause) or else if `X` is an element belonging to the list tail. Note that in the previous example we are almost using the Prolog notation, in which every clause ends with a stop mark. The only difference here is the "<=" symbol, which in Prolog would be "`:-`".

Another important concept from FOL which is largely employed in ILP is *substitution*, which assigns terms to variables, and may be used to obtain a specific instance of a given formula. This concept is formalized in the following definition:

Definition 3 (Substitution). *Given a wff φ with n variables $X_1, ..., X_n$, a substitution $\theta = \{X_1/t_1, ..., X_n/t_n\}$ defines a variable replacement function, where each variable X_k is substituted by a corresponding term t_k in φ.*

Given a literal P, the result of applying a substitution θ to P is written as $P\theta$. A substitution θ unifies two literals P and Q if we have $P\theta = Q\theta$. For example the substitution $\theta = \{X/a,\ Tail/[b,c]\}$ unifies the literals `member(X,[X|Tail])` and `member(a,[a|[b,c]])`[1]. The `member` term is a also predicate, as explained next, presented as an example in Figure 5.2.

[1] As in *Prolog* the simplified list representation is indeed a binary term, and for example "$[b,c]$" represent a term like: $term(b, trem(c, \emptyset))$, with the *term* name being equal to the full stop (".") character.

5.2.1 Inference Rules

In *Mathematical Logic* there are two main concerns related to *well formed formulae* (wff): *semantic* and *syntax*. In *model theory* we are interested to assign meaning (semantic) to a formula, i.e. to know when it is true or false. We explain this more formally in Appendix A.2, through definitions 10 and 11. Informally, an interpretation is a function that maps any wff φ into a given domain determined by the ground facts in which φ is true. We say that an interpretation $\mathcal{I}$ is a *model* to φ if it assigns the true value to φ. A formula is said to be *satisfiable* if it has a *model*, and *unsatisfiable* otherwise. We say that a formula φ is logically implied by another wff ψ, i.e. $\psi \models \varphi$ if every model for ψ is a model for φ. In that case we also say that ψ *semantically entails* φ.

In *proof theory* the focus is syntax - what kind of formula operation maintains its syntactical correctness? One want to know how to correctly generate new formulae (conclusions) from other formulae (premises). This has to do with *inference rules* or deductive reasoning. We say that a formula φ is deductible from a set of formulae $\mathcal{S}$, expressed as $\mathcal{S} \vdash \varphi$, if there exist a *proof* of φ from $\mathcal{S}$. A proof is a sequence of formula consequences obtained through an inference rule (see Definition 13 from Appendix A.2). if we have $\mathcal{S} \vdash \varphi$ we also say that $\mathcal{S}$ *syntactically entails* φ.

We say that an inference rule is *sound* if whenever we have $\mathcal{S} \vdash \varphi$ we also have $\mathcal{S} \models \varphi$. We say that the inference rule is complete if the previous implication holds to in the opposite direction, that is if $\mathcal{S} \models \varphi$ then $\mathcal{S} \vdash \varphi$. When a set of inference rules is both sound and complete, the two concepts are equivalent and we are sure that deduced formulae from valid formulae are still valid. One such inference rule is *deduction modus ponens*, simply mentioned as **deduction** .

Deduction is one of the Logic Programming corner stones, specially *Prolog*, and the other one is *unification* (Robinson, 1965). To illustrate *deduction* suppose that we have a common sense rule stating that every man having at least one million dollars and a big house is considered a rich man. Then if a particular person, let's say john, fulfills these conditions, we naturally conclude (deduce) that john is a rich man. Schematically we have:

$$
\begin{array}{l}
MillionDollar(X) \wedge BigHouse(X) \Rightarrow IsRich(X) \\
MillionDollar(john) \wedge BigHouse(john) \\
\hline
\therefore\ IsRich(john)
\end{array}
$$

Note that variable "X" was unified[1] with constant "$john$". This mechanism contains a trivial case of syntactic entailment, expressed as:

$$\{\alpha \Rightarrow \beta,\ \alpha\} \ \vdash\ \beta \tag{5.5}$$

where formulae $\alpha \Rightarrow \beta$ and α are respectively the first and second premises and β is the conclusion.

[1] From the unification algorithm (Robinson, 1965).

Furthermore, modus ponens it is a special case of a more general inference rule, which is **resolution**, defined by Robinson (1965). The resolution rule is a valid inference rule that produces a new clause from two clauses having complementary literals. The two original clauses should be in a normal disjunctive form, i.e a sequence of literals connected with the disjunction operator. Two literals p_i and q_j are complimentary if one is the negation of the other one, for example $p_i = \neg q_j$. Schematically, the resolution rule may be drawn as shown in Figure 5.3. The resulting formula is a clause with a

$$
\begin{array}{c}
p_1 \vee p_2 \vee \ldots \vee p_i \vee \ldots \vee p_n \\
q_1 \vee q_2 \vee \ldots \vee q_j \vee \ldots \vee q_m \\
\hline
\therefore \quad p_1 \vee p_2 \vee \ldots \vee p_{i-1} \vee p_{i+1} \vee \ldots \vee p_n \quad \vee \quad q_1 \vee q_2 \vee \ldots \vee q_{j-1} \vee q_{j+1} \vee \ldots \vee q_m
\end{array}
$$

Figure 5.3: The general resolution scheme.

disjunction of their premisses, excepting the complimentary literals p_i and q_j.

5.3 ILP Generals

Although only in 1991 the term *Inductive Logic Programming* (ILP) was defined in (Muggleton, 1991), some concepts and key ideas of this discipline have its roots way back in the 1960s, from the fields of Psychology and Artificial Intelligence (Sammut, 1993). It is a formal scheme of an inductive process through computational logic and can be defined as the intersection between Inductive Learning and Logic Programming.

$$ILP \quad = \quad Inductive\ Learning \bigcap Logic\ Programming$$

As a learning mechanism it provides several advantages over other learning schemes, which are usually based on propositional logic, like for example *decision tree induction*, where the hypothesis is constructed from a set of learning instances represented in a n-dimensional space where for each instance we have $n-1$ feature values and the remainder value is the class or target value.

The task of an ILP system may be defined as the knowledge production from a set of observed learning examples ($\mathcal{E}$), according to the knowledge one already has about some problem domain, named as the background knowledge ($\mathcal{BK}$). The knowledge created is codified through a set of rules which form utterly the best theory or hypothesis which explains the observations and still be coherent with the $\mathcal{BK}$. This theory is also known as the hypothesis (h). Being "$\vdash$" the logic entailment operator, we say that a set of first order logic formulae $\mathcal{S}$ entails a given formula q ($\mathcal{S} \vdash q$) if it can be obtained from $\mathcal{S}$ through natural deduction[1], as defined in propositions A.13 and A.14, in Section A.2.

[1]There are other deduction operators that may be used.

The objective of ILP can be stated in a more formal way as the search for the best hypothesis h from an hypothesis space $\mathcal{H}$ such that:

$$\mathcal{BK} \wedge h \vdash \mathcal{E}^{+} \tag{5.6}$$

$$\mathcal{BK} \wedge h \nvdash \mathcal{E}^{-} \tag{5.7}$$

where $\mathcal{BK}$ is the background knowledge and $\mathcal{E}^{+}$ and $\mathcal{E}^{-}$ are the sets of positive and negative examples respectively.

In order to have a simple and illustrative example, suppose that one wants a system to learn the human concept of *being in love*, through ILP techniques. Of course, this aims only to identify the most elementary conditions for such a human complex relationship, which states that for two humans to be in love they must love each other. In terms of predicate representation we want to induce rules where the head is the $inLove(X,Y)$, meaning that X and Y are in love. The system will be supplied with a small population of 16 humans, 8 male and 8 female, and their affective relationships, which are represented in Figure 5.4.

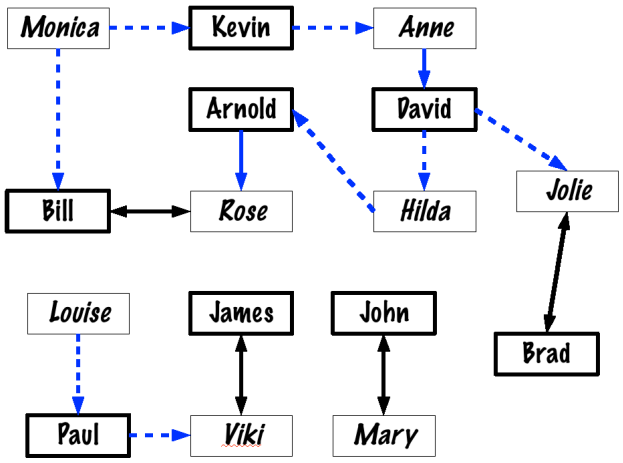

Figure 5.4: "Humans in love!"

The one side dash arrow means that some one loves the other one, however the reverse does not hold. For example, Monica loves Kevin but Kevin does not love her. Naturally the full two side arrows represent love feelings in both directions. This information is represented through FOL predicates and shown in Table 5.3.

With only this information, an ILP system may be able to make a generalization about the $inLove(X,Y)$ concept, and in this case the constructed hypothesis (*h*) would have only one clause shown in 5.8.

$$inLove(X,Y) \Leftarrow love(X,Y) \wedge love(Y,X) \tag{5.8}$$

meaning that X and Y are in love if X loves Y and Y loves X. Note that both conditions 5.6 and 5.7 are

Table 5.2: Relations expressed in FOL.

$\mathcal{BK}$		$\mathcal{E}^+$	$\mathcal{E}^-$
$human(X) \Leftarrow male(X)$		$inLove(brad, jolie)$	$inLove(kevin, anne)$
$human(X) \Leftarrow female(X)$		$inLove(mary, john)$	$inLove(jolie, louise)$
		$inLove(rose, bill)$	$inLove(anne, david)$
$love(jolie, brad)$	$male(paul)$	$inLove(james, viki)$	$inLove(wiki, john)$
$love(brad, jolie)$	$male(james)$		$inLove(hilda, arnold)$
$love(anne, david)$	...		$inLove(bill, anne)$
$love(monica, kevin)$	$female(mary)$		$inLove(rose, arnold)$
$love(arnold, rose)$	$female(viki)$		$inLove(paul, bill)$
...	...		

satisfied therefore the Formula 5.8 represents a complete and consistent hypothesis.

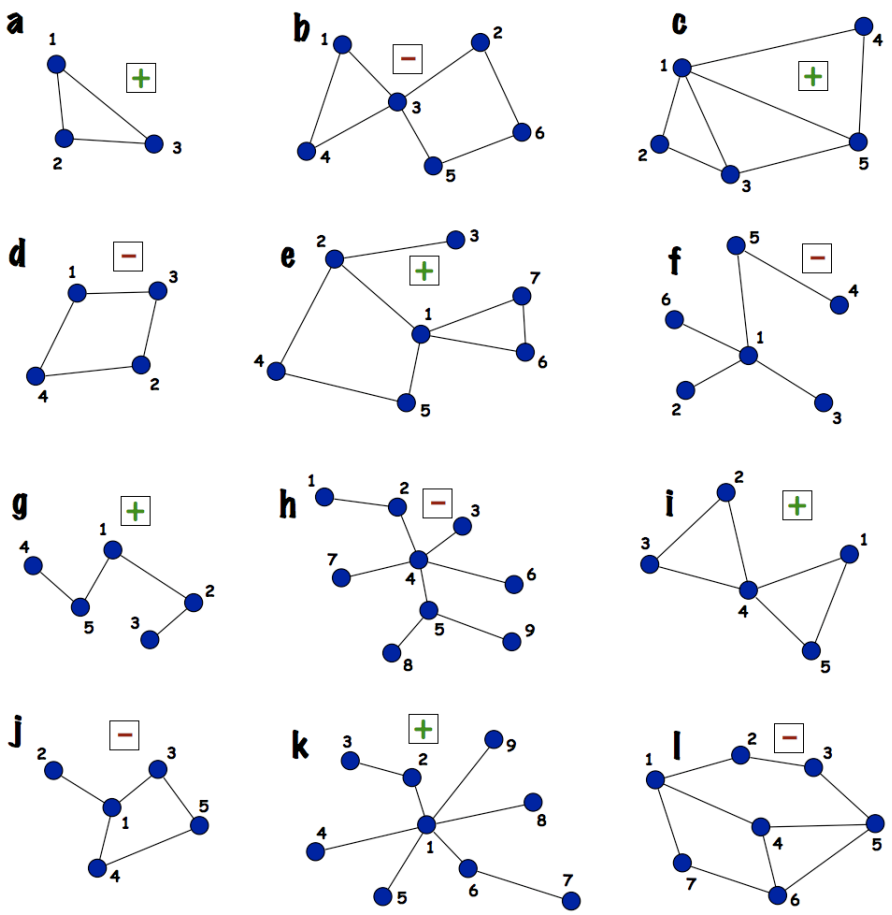

Figure 5.5: Example: "Why are they different?"

Despite the simplicity of the previous example, usually an ILP system is employed to learn more complex and hidden relations, as for example mining relations in relational data bases (Lavrac, 2001).

In the example of Figure 5.5 it is not so obvious to see what is the difference between two graph types, identified as "$+$" and "$-$". The answer will be given in Section 5.4, where the ILP *Aleph* system (Srinivasan, 2000) is described.

5.3.1 The Structured Hypothesis Search Space

The majority of learning tasks may be seen as the searching process for the best possible hypothesis h from an **hypothesis space** $\mathcal{H}$ (Mitchell, 1982) that can be, and usually is, extremely large if not infinite. Fortunately, methods were devised in *machine learning* to overcome these difficulties by exploiting strategies to carry out an intelligent and systematic search through the hypothesis space. One of these techniques employs the θ-subsumption operator, based on term variable substitution, as presented in Definition 3.

Definition 4 (θ-subsumption). *Given two clauses c_1 and c_2, we say that c_1 θ-subsumes c_2, represented as $c_1 \rhd_\theta c_2$, if and only if there exists a substitution θ such that $c_1\theta \subseteq c_2\theta$.*

For example, if we have:

$$c_1 = inLove(X,Y) \Leftarrow love(Y,X)$$
$$c_2 = inLove(brad,jolie) \Leftarrow love(jolie,brad) \wedge female(jolie)$$

then we have $c_1 \rhd_\theta c_2$, with $\theta = \{X/brad, Y/jolie\}$, because if we regard clauses as sets of conjunctions of literals, we have $c_1\theta = \{inLove(brad,jolie),\ \neg love(jolie,brad)\}$, which is a subset of $c_2\theta = \{inLove(brad,jolie),\ \neg love(jolie,brad), female(jolie)\}$.

The θ-subsumption operator ensures the important relation of logical implication in the $\mathcal{H}$ space, as formalized in the next proposition:

Proposition 1. *For any two clauses c_1 and c_2, if $c_1 \rhd_\theta c_2$ then $c_1 \models c_2$.*

In general, the contrary does not hold. For example, in the next case $succ(X)$ is a function which returns the successor of a natural number (ex: $succ(1) = 2$) and $natural(X)$ a predicate expressing that its argument is a natural number. Continuing with our example, we have:

$$c_1 = natural(sucessor(X)) \Leftarrow natural(X)$$
$$c_2 = natural(sucessor(successor(Y))) \Leftarrow natural(Y)$$

It is clear that $c1 \models c_2$, however there is no substitution θ that allows c_1 to θ-*subsume* clause c_2.

From the Gödel theorem (Theorem 6, in Appendix A) and the previous proposition, we may also conclude that if $c_1 \rhd_\theta c_2$ then c_2 is deducible from c_1, i.e $c_1 \vdash c_2$. This means that clause c_1 is more general than clause c_2, and that we have a relation among clauses which gives us the ability to

organize/order the $\mathcal{H}$ space in a way that will facilitate searching through the hypothesis space. To have a clearer view of this we first need to introduce the mathematical notion of a *partial order*.

For a given set $\mathcal{A}$, a partial order is a binary ordering operator ($\sqsubseteq$) with the properties of reflexivity, antisymmetry and transitivity. That is, for any elements $a \in \mathcal{A}$, $b \in \mathcal{A}$, and $c \in \mathcal{A}$ we have:

$$a \sqsubseteq a \quad (reflexive) \tag{5.9}$$

$$a \sqsubseteq b \wedge b \sqsubseteq a \Rightarrow a = b \quad (antisymmetric) \tag{5.10}$$

$$a \sqsubseteq b \wedge b \sqsubseteq c \Rightarrow a \sqsubseteq c \quad (transitive) \tag{5.11}$$

Definition 5 (lattice). *A structure $\langle \mathcal{A}, \vee, \wedge, \sqsubseteq \rangle$ is a lattice if $\sqsubseteq$ is a partial order over the set $\mathcal{A}$, and any pair of elements in $\mathcal{A}$ has a supreme and an infimum, denoted respectively as $a \vee b$ and $a \wedge b$, where $a \in \mathcal{A}$ and $b \in \mathcal{A}$.*

For example, the structure $\langle \mathcal{P}(\mathcal{A}), \cup, \cap, \subseteq \rangle$, where $\mathcal{P}(\mathcal{A})$ is the power set[1] of $\mathcal{A}$ is a lattice. For instance, if $\mathcal{A} = \{x, y, z\}$ we would have:

$$\mathcal{P}(\mathcal{A}) = \{\emptyset, \{x\}, \{y\}, \{z\}, \{x, y\}, \{x, z\}, \{y, z\}, \{x, y, z\}\}$$

and the lattice may be represented with an *Hasse* diagram as in Figure 5.6

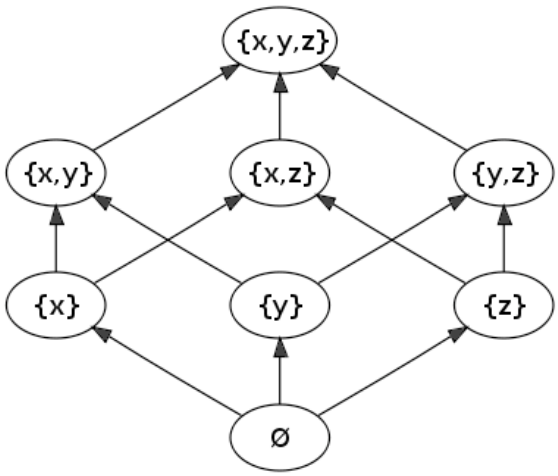

Figure 5.6: A lattice Hasse diagram for $\mathcal{P}(\{x, y, z\})$.

Proposition 2. *Being $\mathcal{H}$ the space of admissible hypotheses which can explain a set of examples $\mathcal{E}$, based on a set of background knowledge clauses $\mathcal{BK}$, then the structure $\langle \mathcal{H}, \vee, \wedge, \rhd_\theta \rangle$ is a lattice.*

The previous proposition asserts that $\vee$ and $\wedge$ exists in $\mathcal{H}$, i.e. any two clauses $c_1 \in \mathcal{H}$ and $c_2 \in \mathcal{H}$ always have a supreme and an infimum clause, which are respectively more general and more specific than c_1 and c_2.

[1] The power set is the set of all subsets of a given set.

The notion of clause pair generalization was initially investigated by Plotkin (1970) as the *least general generalization* (*lgg*). It assumes that if two clauses are true then it is very likely that their *lgg* will also be true. The *lgg* of two terms compares the components in the same predicate position and if they are equal the value will remain in the resulting generalization. If they are not equal, then that position is replaced by a variable. A few examples are shown in Table 5.3.1. Hence the supreme

Table 5.3: Examples of *least general generalizations*.

T_a	T_b	$lgg(T_a, T_b)$
$love(brad, jolie)$	$love(brad, jolie)$	$love(brad, jolie)$
$love(brad, jolie)$	$love(brad, anne)$	$love(brad, Y)$
$love(brad, jolie)$	$love(john, jolie)$	$love(X, jolie)$
$love(john, jolie)$	$love(brad, mary)$	$love(X, Y)$
$love(brad, jolie)$	$dream(brad, jolie)$	Z

operator, which gives the minimum of the majorants, for two clauses c_1 and c_2 in $\langle \mathcal{H}, \vee, \wedge, \rhd_\theta \rangle$ may be defined as the $lgg(c_1, c_2)$, i.e $c_1 \vee c_2 = lgg(c_1, c_2)$.

Since the *lgg* generalization operator does not consider the background knowledge, a *relative least general generalization* (*rlgg*) operator was proposed later by Plotkin (1971), where for two positive examples $e_1 \in \mathcal{E}^+$ and $e_2 \in \mathcal{E}^+$ we have:

$$rlgg(e_1, e_2) \;=\; lgg(e_1 \Leftarrow \mathcal{BK},\; e_2 \Leftarrow \mathcal{BK}) \tag{5.12}$$

and here the background knowledge is already taking into account, in the generalization process. This is more realistic and coherent with the human mental generalization process, which always works with the whole knowledge in a given problem (the background knowledge or world knowledge).

The *lgg* and *rlgg* enable a bottom-up theory construction through successive, though non-deterministic generalizations following the paths in the $\langle \mathcal{H}, \vee, \wedge, \rhd_\theta \rangle$ lattice.

The opposite direction (top-down) is also possible since for any two hypotheses there exist an infimum, which will be a more specific hypothesis. Specialization operators were also investigated and employed in many ILP systems like FOIL (Quinlan, 1990). One of these is based on θ-subsumption and a refinement graph, which is a directed acyclic graph, where the nodes are clauses and the arcs are possible refinement operations that may be followed, representing the replacement of a variable with a term, or an insertion of a new literal in the clause body. The search starts with the most general clause, which becomes more specific after each search step, until certain conditions like *coverage* or *entropy* are satisfied. Part of a refinement graph from an example used by Lavrac & Dzeroski (1994) is shown in Figure 5.7. The objective there is to learn the concept of "daughter(X,Y)", where the $\mathcal{BK}$ set contains facts about being "male", "female", "father", "mother" and a predicate which defines the

concept "progenitor". The search starts with the most general clause "$daughter(X,Y) \Leftarrow$", stating

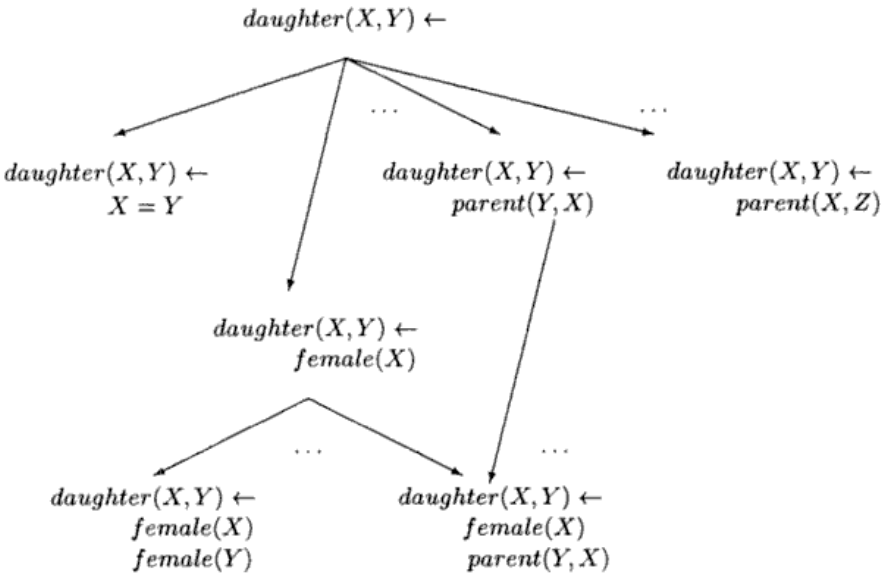

Figure 5.7: Part of the refinement graph, for learning the concept "daughter" (Lavrac & Dzeroski 1994)

that every X is a daughter of Y, which is, of course, too general and heavily inconsistent. Therefore, a search of successive specialization (refinement) steps is undertaken, following a given search method, like *hill-climbing* or the *A* best-first*.

5.3.2 Inverse Resolution

Another generalization method is the *inverse resolution*, introduced by (Muggleton & Butine 1998), consisting of the process of inverting the deductive inference resolution rule (see Figure 5.3). Resolution identifies a pair of literals, one in clause c_1 and the other in c_2, where one is the negation of the other one and then derive a new formula, the resolvent $res(c_1, c_2) = c$, where these two opposites were eliminated and the remaining literals maintained, i.e $c = (c_1 \setminus \{a\} \bigcup (c_2 \setminus \{\neg a\})$[1].

$$
\begin{array}{rll}
c_1 = & \neg a \vee b & = a \Rightarrow b \\
c_2 = & \neg b \vee c & = b \Rightarrow c \\
 & \text{----------} & \text{----------} \\
res(c_1, c_2) = & \neg a \vee c & = a \Rightarrow c
\end{array}
$$

In terms of propositional logic it is easy to devise an inverse resolution operator, also named as inverse entailment, by considering one of the initial clauses, let's say c_1 and the resolvent $res(c_1, c_2)$. It should enable the inference of the other clause, c_2 in this case, through the following two steps:

[1] In this context the "\" and "⋃" operators still represent set difference and union, respectively. The sets are formed by the clause literals.

1. Find a literal A that occurs in c_1, but not in clause c
2. Then construct clause c_2 with the literals of: $(c \setminus (c_1 \setminus \{a\})) \ \bigcup \ \{\neg a\}$

For example if we have:

$$\begin{array}{rl} c_1 = & a \\ c_2 = & ??? \\ & \text{----------} \\ c = res(c_1, c_2) = & b \end{array}$$

then following strictly the operator, we get $c_2 = (c \setminus (c_1 \setminus \{a\})) \ \bigcup \ \{\neg a\}$, and $c_1 \setminus \{a\} = \emptyset$, $c \setminus (c_1 \setminus \{a\})$, and finally $c_2 = b \vee \neg a$, or equivalently: $c_2 = a \Rightarrow b$, and so a rule has been induced.

For the case of FOL formulae the *inverse entailment* operator is based in the same principle, however it contains an extra complexity due to the possible existence of quantified variables. In this situation the resolution works together with *unification* (Robinson, 1965) through an unifying substitution. An illustration with our previous "$inLove(X, Y)$" domain is:

$$\begin{array}{rl} c_1 = & inLove(X, Y) \ \Leftarrow \ love(X, Y) \wedge love(Y, X) \\ c_2 = & love(brad, jolie) \wedge love(jolie, brad) \\ & \text{----------} \\ c = & inLove(brad, jolie) \end{array}$$

and the unifying substitution would be $\theta = \{X/brad, Y/jolie\}$. The resolution rule in FOL is an extension from its propositional counterpart, incorporating the substitution function. Given two FOL clauses c_1 and c_2, with a_1 a literal occurring in c_1 and a_2 a literal from a_2, and let θ be the unifying substitution among the two clauses, such that $a_1\theta = \neg a_2\theta$, then a new clause c will be deduced as in proposition 5.13.

$$c = (c_1 \setminus \{a_1\})\theta \ \bigcup \ (c_2 \setminus \{a_2\})\theta \tag{5.13}$$

Now it is possible to derive the inverse resolution for FOL clauses by algebraic manipulation of proposition 5.13. From Mitchell (1997), the θ unifier can be uniquely factored in θ_1 and θ_2, such that $\theta = \theta_1\theta_2$ and θ_1 contains all substitutions involving variables from c_1 and θ_2 substitutions involving those from c_2. Hence, Proposition 5.13 is equivalent to Proposition 5.14.

$$c \setminus (c_1 \setminus \{a_1\})\theta_1 \ = \ (c_2 \setminus \{a_2\})\theta_2 \tag{5.14}$$

On the other hand, as we have $a_1 \in c_1$ and $\neg a_2 \in c_2$, then $a_1\theta_1 = \neg a_2\theta_2$ or equivalently $a_2 = \neg a_1\theta_1\theta_2^{-1}$ and finally the inverse resolution equation for FOL clauses is shown in proposition 5.15.

$$c_2 = (c \ \setminus \ (c_1 \setminus \{a_1\})\theta_1)\theta_2^{-1} \ \bigcup \ \{\neg a_1\theta_1\theta_2^{-1}\} \tag{5.15}$$

The $\theta_{\bullet}^{-1}$ operator is simply the inverse of a variable/term substitution operator, mapping concrete entities, i.e. terms, into generalized variables. The inverse entailment operation is not deterministic and different choices for literal a_1 yield to different clause inductions (c_2). To give a simple illustration about this mechanism, we will take our previous $inLove(X, Y)$ example and their $\mathcal{BK}$, $\mathcal{E}^+$, $\mathcal{E}^-$ components, where the target is to induce conditions such that $inLove(X, Y)$. Taking the first positive example, from table in Table 5.3, $inLove(brad, jolie)$, and choosing from $\mathcal{BK}$ a convenient c_1 clause, $love(brad, jolie)$, we would have what is schematized in Figure 5.8.

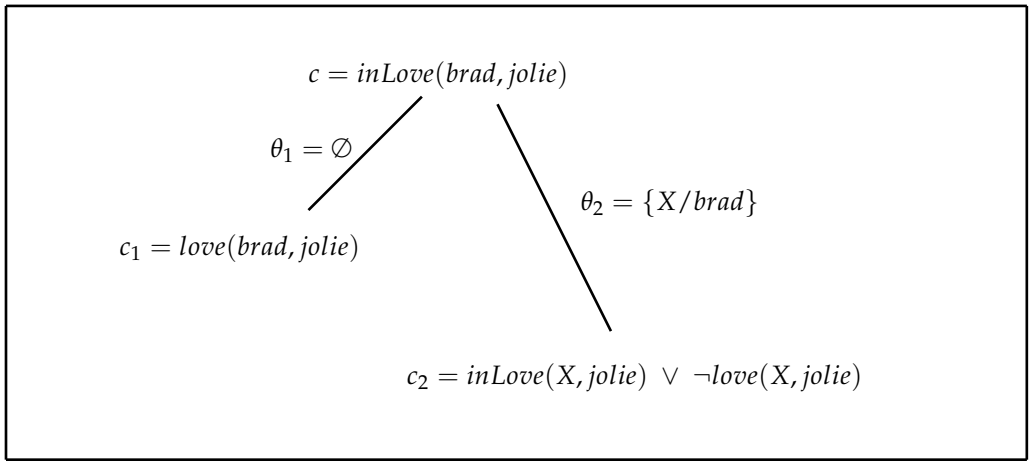

Figure 5.8: An *inverse resolution* first step application for the "In Love" example.

From Equation 5.15 $c_1 = a_1$ and θ_1 without any necessary substitution, hence we get $(c_1 \setminus \{a_1\})\theta_1 = \emptyset$, and

$$\begin{aligned} c_2 &= c\theta_2^{-1} \ \bigcup \ \{\neg a_1 \theta_2^{-1}\} \\ &= inLove(brad, jolie)\ \theta_2^{-1} \ \vee \ \neg love(brad, jolie)\ \theta_2^{-1} \\ &= (inLove(brad, jolie) \ \vee \ \neg love(brad, jolie))\ \theta_2^{-1} \end{aligned}$$

and it is only necessary to choose a θ_2 substitution, which in this case one may think of $\theta_2 = \{X/brad, Y/jolie\}$, however it would not generate the *least general generalization* and the search space would not be systematically traversed (see Subsection 5.3.1), thus only one variable substitution was chosen, as shown in Figure 5.8.

Looking at the generalization achieved so far, it is clear that the fix point is not yet reached, since only one positive example is covered. It only states that someone (X) is in love with *Jolie* if he loves her, but from the background knowledge we know that *David* loves *Jolie* too yet they are not in love. Thus another inverse resolution, step similar to the previous one, will solve the problem and lead the process to the final generalization state. And as result the final rule will be induced, stating that two humans are in love if they love each other (see Figure 5.9).

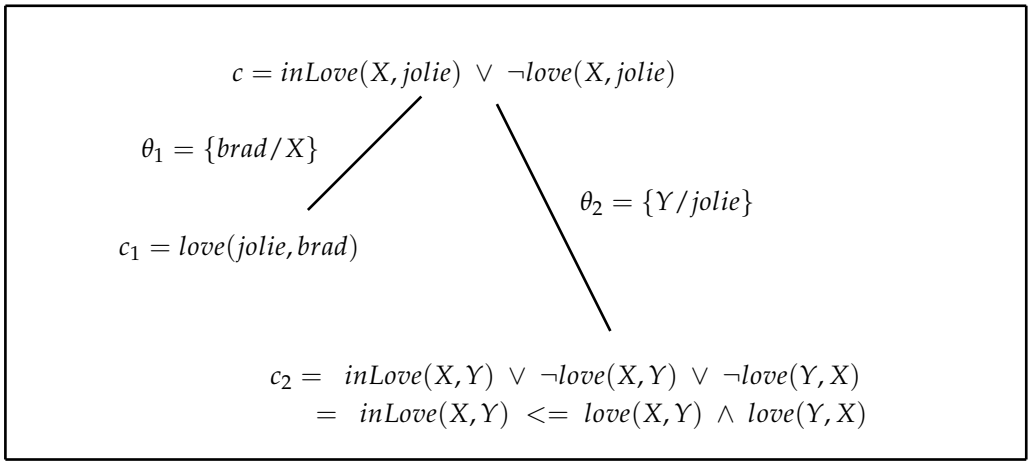

Figure 5.9: An *inverse resolution* second step applied for the "In Love" example (see also Figure 5.8).

The inverse resolution is a mechanism implemented in the kernel of several ILP implementations, as the *Aleph* system. The non-determinism inherent to this logic mechanism requires that some search strategies as well as convenient stop criteria must be set before the induction process starts and each ILP system implements its own criteria. In the next subsection, we present the ILP system we used for sentence reduction rule induction, as well as some particularities and system settings we decided to follow.

5.4 The *Aleph* System

The *Aleph*[1] system (Srinivasan, 2000) is an empirical ILP system, written in *Prolog*, initially designed to be a prototype for exploring ILP ideas and originally named as P-Progol. It has become a quite mature ILP system, used in many research projects, ranging from Biology to NLP areas. In fact, Aleph is the successor of several and "more primitive" ILP systems, like: Progol (Muggleton, 1999), FOIL (Quinlan, 1990), and Indlog (Camacho, 1994), among others, and may be appropriately customized to emulate any of those older systems.

The relationship to *Prolog*, inherent in the *Aleph* system, gave it the capacity to use a conventional and powerful language to represent world knowledge and consequently exploit this in the induction of new relational knowledge. The *Aleph* system contains a set of options providing the user with a vast amount of possible configurations. One may easily and incrementally insert more domain knowledge, choose the direction for rule generation (*bottom-up* or *top-down*), define the evaluation function (*Precision*, *Entropy*, *Laplace*, among others) and the search method (*Hill-Climbing*, *Branch-and-Bound*, *Best-First*, among others). One interesting feature in *Aleph* is the possibility to learn exclusively from positive instances, contrary to what is required by most learning systems. More-

[1] Aleph - **A** Learning **E**ngine for **P**roposing **H**ypothesis

over, there is theoretical research work (Muggleton, 1996) demonstrating that the increase in the learning error tends to be negligible with the absence of negative examples, as the number of learning instances increases. This is a relevant issue in many learning domains, and especially ours, where negative examples are not available, as we will see in the next chapter.

In *Aleph* any knowledge, including the Background Knowledge ($\mathcal{BK}$) is represented using relations, codified through *Prolog* predicates. In the previous subsection, particularly conditions 5.6 and 5.7, we stated that the main objective of induction is to find the best possible hypothesis (h), which in conjunction with $\mathcal{BK}$, entails the maximum number of positive examples $\mathcal{E}^+$, ideally all, and none or the minimum number of negative cases $\mathcal{E}^-$. In such a system, $\mathcal{BK}$, $\mathcal{E}^+$ and $\mathcal{E}^-$ are input sets, while h is the output generated, representing the learned theory that explains the observed data ($\mathcal{E} = \mathcal{E}^+ \cup \mathcal{E}^-$), with respect to the background knowledge ($\mathcal{BK}$). In *Aleph* these three input data/knowledge sets, that is $\mathcal{BK}$, $\mathcal{E}^+$, and $\mathcal{E}^-$, are represented through three files, having respectively ".b", ".p", ".n" extensions.

In the rest of this subsection we give a general overview of the *Aleph* system *modus operandi* and describe some particular features. Some of them were employed in our work in order to satisfy the specific characteristics related to our problem of the induction of sentence reduction rules. The *Aleph* default induction algorithm includes four main steps listed below, which are repeated until no more uncovered positive example is left.

1. **Selection** - Selects a positive example from the training set to be generalized. The algorithm stops if no example is available.

2. **Saturation** - Constructs the non-ground most specific clause that implies the positive example selected in the previous step. This is made through successive application of the *inverse resolution*[1] operator, on the selected example, until all ground terms have been generalized to variables. These substituted variables are reused in the clause body until only the selected positive ground example is covered. This minimum generalized clause is called the *bottom clause*.

3. **Reduction** - The system searches for a "good" clause amongst the variety ranging from the maximally general clause, the *empty* clause, and the maximally specific clause, named as the *bottom* clause[2]. This clause space is a lattice (according do proposition 2), having a partial ordered defined through the θ-subsumption relation (as defined in Definition 4), which ensures a systematically search for a good clause, a least general one covering as far as possible the examples. During this searching process a defined evaluation function is used to measure the quality of the clause. More about this evaluation function is explained later.

[1] As presented in Subsection 5.3.2.

[2] In the previous listing, it is the clause shown in lines 7 to 10.

4. **Cover Removal** - Adds the *best* clause found in step 3 to the induced theory and removes all their covered positive examples from the training set.

More detailed and exhaustive descriptions about all possible *Aleph* configurations, may be obtained from technical documentation in Srinivasan (2000).

To exemplify *Aleph*'s potential to induce useful theories we will use the *graphs* example, presented previously in Figure 5.5. It is not obvious at first sight what the differences are between the two graph classes: "+" and "−". We have 12 graphs labeled with lower case letters (a, b, ..., l), and in each graph the vertices are sequentially marked with numbers: $1, 2, \ldots, n$. Predicate `goodgraph/1` is used to represent the whole set using six positive and six negative cases. Hence we will have two files holding each type, for "+" graphs, stored in file `graphs.p`, in which we have the following six positive examples:

```
goodgraph(a).   goodgraph(c).   goodgraph(e).
goodgraph(g).   goodgraph(i).   goodgraph(k).
```

and for *bad* graphs, file `graphs.n` which will contain the other six negative cases.

```
goodgraph(b).   goodgraph(d).   goodgraph(f).
goodgraph(h).   goodgraph(j).   goodgraph(l).
```

The background information $\mathcal{BK}$ is inserted in a file with the same base name, having a ".b" extension, `graphs.b` in this case. This file contains a set of Prolog predicates, related to the problem in question and with the purpose to guide the induction process. Besides, there are two main directives that can be defined, which establishes the predicates that can be combined by *Aleph* in the induction process. Two mode directives, `modeh/2` and `modeb/2`, can be used. The first one defines what is the concept to be learned. The second one determines which predicates can be tried out during the rule (predicate body) construction process. In our graph set example, we would have the following predicates:

```
:- modeh(1, goodgraph(+gname)).
:- modeb(1, numvertex(+gname, -int)).
:- modeb(1, numedges(+gname, -int)).
:- modeb(1, maxdegree(+gname, -int)).
:- modeb(*, diff(+int, +int, #int)).
:- modeb(*, odd(+int)).
:- modeb(*, even(+int)).
:- modeb(*, prime(+int)).
```

Thus the mode directive (`:- modeh(1, goodgraph(+gname))`) defines that the aim is to generate a predicate definition of `goodgraph(...)`. The first argument indicates the maximum number of times the predicate, indicated in the second argument position, can occur in the rule body, which can be an integer: 1,2,..., or "*", meaning one or more times. Each predicate that can be used in the induced clauses can have arguments represented by symbolic names representing a certain data type. In our example `gname` and `int` respectively stand for "graph name" and an integer value. Each data

type name is preceded by one of the three symbols for input/output mode: the "+" meaning an input parameter, "−" an output parameter, and "#" for a constant. The plus and minus symbols are normally used to bind variables in the induced rules. A general scheme for each rule is `H :- B1, B2, ..., Bn`, with header `H` and the body literals `B1, ..., Bn` which may include variables. An input variable (+) in a given `Bj` must be either an output variable (−) in `H` or an output variable in some `Bi`, where `i<j`. An argument preceded by the constant symbol (#) can only be replaced by a ground atom.

For our graph example and with the previous mode declarations, we are informing *Aleph* that the rule head must be `goodgraph/1` and for the rule body several graph properties could be explored in order to find a rule that will characterize well the training set, ideally covering all positive examples and no negative ones. Therefore the number of graph vertices, edges and graph maximum degree[1] are considered through, respectively, the "`numvertex/2`", "`numedges/2`" and "`maxdegree/2`" predicates. For each one of these, we have the graph name as input parameter and an integer as an output parameter, which in turn is considered for numeric relation discovery through the other predicates: "`diff/3`", "`odd/1`", "`even/1`" and "`prime/1`", in this case. The last three verify whether a given integer is odd, even, and prime number respectively, as suggested by the name. The predicate "`diff(+int,+int,#int)`" is used to discover numeric differences between two feature values. Here we are assuming that the underlying law characterizing the positive graphs may rely on the difference between vertices and edges, like vertices minus edges being always equal to 3. Then this phenomenon could be expressed using the following rule:

```
goodgraph(G) :- numvertex(G,V), numedges(G,E), diff(V,E,3).
```

meaning that a graph `G` is a poisitive graph ("+") if it has `V` vertices, `E` edges and `V-E=3`.

Immediately after the mode declarations, should come the `determination/2` directives, which indicate which are the predicates allowed to be used in the construction of each rule body. In our example we have the following ones:

```
:- determination(goodgraph/1, numvertex/2).
:- determination(goodgraph/1, numedges/2).
:- determination(goodgraph/1, maxdegree/2).
:- determination(goodgraph/1, diff/3).
:- determination(goodgraph/1, odd/1).
:- determination(goodgraph/1, even/1).
:- determination(goodgraph/1, prime/1).
```

Afterwards comes a section including all facts and all the predicate definitions directly or indirectly involved in the rule induction. For example, in our case facts attributing names for graphs are:

```
gname(a).   gname(b).   gname(c).   gname(d).
gname(e).   gname(f).   gname(g).   gname(h).
gname(i).   gname(j).   gname(k).   gname(l).
```

Each graph is defined through a 2-ary predicate `graph/2`, with the first argument containing the

[1] The maximum number of connections (edges) among the graph's vertices.

graph name and the second argument a list of all graph edges. Each edge is represented by a term connecting two graph nodes, in the form of "A−B", where "A" and "B" represent vertex connections. Thus the twelve graphs shown in Figure 5.5 are represented as:

```
graph(a, [1-2, 2-3, 3-1]).                                  (+)
graph(b, [1-3, 1-4, 4-3, 3-2, 3-5, 5-6, 2-6]).              (-)
graph(c, [1-2, 1-3, 1-4, 1-5, 4-5, 5-3, 3-2]).              (+)
graph(d, [1-3, 3-2, 2-4, 4-1]).                             (-)
graph(e, [1-2, 1-7, 1-6, 1-5, 7-6, 4-5, 4-2, 2-3]).         (+)
graph(f, [1-5, 1-6, 1-2, 1-3, 4-5]).                        (-)
graph(g, [1-2, 2-3, 1-5, 5-4]).                             (+)
graph(h, [1-2, 2-4, 4-3, 4-7, 4-5, 4-6, 5-8, 5-9]).         (-)
graph(i, [1-4, 1-5, 5-4, 4-2, 4-3, 2-3]).                   (+)
graph(j, [1-2, 1-3, 1-4, 3-5, 4-5]).                        (-)
graph(k, [1-2, 3-2, 1-9, 1-8, 1-6, 6-7, 1-5, 1-4]).         (+)
graph(l, [1-2, 2-3, 3-5, 4-1, 4-5, 4-6, 6-5, 6-7, 7-1]).    (-)
```

The remainder of the file contains the definitions for the predicates declared in the mod and determination directives and any other auxiliar predicates used by them. The complete `goodgraph.b` file may be consulted in Listing C.1, in Appendix C. This completes the *Aleph* background knowledge definitions for our illustrative "good graph" example. Starting *Aleph* and running the default inducer "`induce`", after loading the problem files (`goodgraph.*`), through the "`read_all(goodgraph)`" command, only one exact rule will be induced, which covers all positive examples and no negatives.

```
[Rule 1] [Pos cover = 6 Neg cover = 0]
goodgraph(A) :-
   numvertex(A,B), odd(B), maxdegree(A,C), even(C).
```

Thus, according to this, a "good graph" has an odd number of vertices and a maximum degree which is an even number. The execution will also print some statistical informations. First it shows the confusion matrix and afterwards more details concerning how the theory was generated.

```
          Actual
       +        -
     + 6        0        6
Pred
     - 0        6        6

       6        6        12

Accuracy = 1.0
[Training set summary] [[6,0,0,6]]
[time taken] [0.011]
[total clauses constructed] [43]
```

An *Aleph* induction execution produces a trace, listing clause search along with their number of positive and negative examples covered. An excerpt from this listing is shown below accompanied by our commentary. The process starts with:

```
1     ?- induce.
2  [select example] [1]
3  [sat] [1]
4  [goodgraph(a)]
```

Then it shows the following bottom clause that was generated:

```
5  [bottom clause]
6  goodgraph(A) :-
7     numvertex(A,B), numedges(A,B), maxdegree(A,C), diff(C,C,0),
8     diff(C,B,1), diff(B,C,-1), diff(B,B,0), odd(B),
9     even(C), prime(C), prime(B).
10 [literals] [12]
11 [saturation time] [0.002]
```

The search for the right clause is initiated.

```
12 goodgraph(A).
13 [6/6]
14 goodgraph(A) :-
15    numvertex(A,B).
16 [6/6]
17 goodgraph(A) :-
18    numedges(A,B).
19 [6/6]
20 goodgraph(A) :-
21    maxdegree(A,B).
22 [6/6]
23 goodgraph(A) :-
24    numvertex(A,B), diff(B,B,0).
25 [6/6]
26 goodgraph(A) :-
27    numvertex(A,B), odd(B).
28 [6/3]
29 ...
```

The last clause shown includes a subset of literals of the final clause shown earlier. The information indicates that it covers all six positive examples and three negative ones. As it still covers some negative examples, this serves as input for further specialization by adding more literals.

The default *Aleph* execution mode, described earlier, can be changed, adapted and configured for many specific needs, for a wide range of learning problems, varying from *decision trees* to *abductive reasoning*. These system settings are made through the `set/2` *Aleph* built-in predicate, which can be inserted in front of the mode declarations (`modeh/2` and all `modeb/2`), as many times as needed, one for each parameter to be configured. For our *good graph* example two system parameters were set:

```
:- set(depth, 100).
:- set(clauselength, 10).
```

The first one specifies the upper bound on the proof depth in the process of the theorem-proving, while the second setting specifies the maximum clause length in terms of literals admissible for the induced rules.

One setting permits to determine the search strategy to be followed by *Aleph* in the clause-by-clause search. The default is to prefer shorter clauses and this may be explicitly defined as:

```
:- set(search, S).
```

with S variable set to "bf". Other search strategies are supported, like for example a heuristic *best-first* (heuristic), or an iterative beam search (ibs), among others which may be consulted in *Aleph* technical report (Srinivasan, 2000).

Another setting defines the evaluation function for evaluating the quality of the theory that is being induced. This is similar to, for example what happens with the ID3 algorithm for *decision tree* learning, where *entropy* and *information gain* is used to guide the step-by-step tree construction. In *Aleph* any evaluation function takes into account the number of positive (P) and negative (N) examples covered at a given moment in the induction process. The default evaluation function is coverage, calculated as $P - N$. The setting of an evaluation function is made through:

```
:- set(evalfn, EvFunc)
```

where EvFunc may be accuracy as $\frac{P}{P+N}$, compression as $P - N - L + 1$, where L represents the number of literals in the clause, coverage as $P - N$, entropy as $p \cdot log(p) + q \cdot log(q)$ where $q = 1 - p$ and $p = \frac{P}{P+N}$, gini as $2p(1-p)$ and $p = \frac{P}{P+N}$, laplace as $p = \frac{P+1}{P+N+2}$, posonly (learning exlusively from positive examples), user, among others (Srinivasan, 2000). This last one (i.e. user) is user driven, meaning that the evaluation function is specified by the user through the *Aleph*'s built-in cost/3 predicate, which has the following syntax:

```
cost(Clause, ClauseLabel, Cost) :- Body
```

where ClauseLabel is an input three element list [P,N,L] specifying the number of positive and negative examples covered by the Clause and L represents the clause length. The Cost parameter is the output cost value for that clause. The cost/3 predicate gives the user the possibility to specify his own evaluation function to be used during the induction process. In the next chapter (see in particular Subsection 6.2 and Figure 6.10) we present our defined cost function for the induction process.

5.5 Final Remarks

In this chapter we have addressed a number of fundamental aspects involved in *Inductive Logic Programming*. We started with a brief introduction on Machine Learning (Section 5.1), mentioning propositional and relational learning, and remarking the knowledge representation power of the latter. Then, in Section 5.2, we introduce the field of *Logic Programming* as the basic language for ILP, mentioning the *inference* mechanisms as well. In Section 5.3 we have presented several ILP key aspects, including a general description, the search in an ILP hypothesis space, and *inverse resolution*. Afterwards, in Section 5.4, we present *Aleph*, the ILP implementation which we have worked with. A number of times, throughout this chapter, we can find references to *Mathematical Logic* concepts. These have been included in the Appendix A.

In the next chapter we focus on the application of this ILP paradigm on our problem of learning

sentence reduction rules. One can find the particular engineerings we have implemented, using *Aleph* and working with the data gathered from the previous steps - a corpus of aligned paraphrases, automatically extracted from *web news stories*.

Chapter 6

Induction of Sentence Reduction Rules

"If I have seen further it is by standing on the shoulders of giants."

Sir Isaac Newton, 1676

In the previous chapter we have given a brief overview of *Inductive Logic Programming* and the *Aleph* system. This system has been used in our approach to the problem of sentence reduction. In this chapter we describe the specifications and main settings used for this task.

In chapters 3 and 4, we have described the method for constructing a corpus of word-aligned paraphrases, using a complete automatic procedure. First, paraphrase sentence pairs were collected from comparable news stories corpora and afterwards an alignment process was carried out to align the words of each paraphrase pair, following a dynamic strategy that chooses between global or local alignment.

By examining aligned paraphrase corpus we perceived that there are different sentence, or phrase, transformation patterns. For example, we can get common aligned sentence segments[1], but certain sentence portions get transformed into different sequences, or may even be omitted in the other sentence. Therefore the subsequent natural idea was to investigate certain "regions" from these aligned paraphrases, with the aim of providing learning instances for the induction of sentence reduction rules. In our case, we have decided to use the *Aleph* system as it revealed more powerful than other existing tools from the area of propositional learning. *Aleph* enables the induction of relational rules (clauses), which is one of its main advantages. Hence our objective at this stage was to decide which kind of aligned sentence portions we should select and conveniently transform them into learning instances. This task is detailed in the following subsection.

6.1 Bubble Extraction

We note that the corpus of aligned paraphrases, contained several aligned sentence portions, worth to be explored. These could serve in investigating of the existence of certain sentence transformation

[1] Both segments with exactly the same words.

regularities or patterns. Therefore, we have decided to focus our attention on one of such type of aligned portions of a sentence pair, that we named as *bubbles*. To give a more precise comprehension of what a *bubble* is, we introduce some definitions and provide some schematic representations of these entities.

Definition 1 (*Pair-sub-segment*). A pair-sub-segment is constituted of two aligned word sequences, extracted from an aligned sentence pair.

```
A B S G T _ _ _ G A B R A T A T _ P .
A _ S G T A B A G A C R B T _ _ V P .
```

Figure 6.1: Two word-aligned sentences. Each word is represented with a letter.

For example, the aligned sequence pair in Figure 6.1 gives rise to many examples of pair-sub-segments, from which we show just seven possibilities in Figure 6.3. A special type of pair-sub-segments is when both word sequences are exactly the same, as specified by this definition:

Definition 2 (*EQsegment*). An EQsegment is a pair-sub-segment having exactly the same words in each sequence, written in the exact same order.

For example, in the alignment from Figure 6.1 there are six EQsegments, which are:

```
(1)  A       (2)  S G T     (3)  G A
     A            S G T          G A

(4)  R       (5)  T         (6)  P
     R            T              P
```

Figure 6.2: The only six EQsegments from the alignment of Figure 6.1.

Now we are able to define a bubble as a special kind of a pair-sub-segment, as in Definition 3.

Definition 3 (*Bubble*). A *Bubble* is a pair-sub-segment from a word aligned sentence pair, where two heterogeneous word sequences are aligned, and are delimited by a left and right contexts of EQsegments, with at least one equal and aligned word in each context.

```
(1)  A B S G T       (2)  _ _ _       (3)  P
     A _ S G T            A B A            P

(4)  T _ _ _ G       (5)  T A T _ P   (6)  A T A
     T A B A G            T _ _ V P        B T _

(7)  S G T _ _ _ G A
     S G T A B A G A
```

Figure 6.3: Seven examples of pair-sub-segments containing four bubbles.

From the previous definition, one may look at a *bubble* as a transformation of a sentence segment, taking place within a given context. Therefore one may divide a bubble into three main components: *left context* (L), *right context* (R), and the *segment transformation* or *kernel*, which may be represented by X. The next figure shows these three components, for the four bubbles present in Figure 6.3.

```
      L  |    X    |   R                L  |    X    |  R
    -----|---------|------          -------|---------|-------
(1)  A   |   B S   |  G T      (5)     T   |  A T _  |  P
     A   |   _ T   |  G T              T   |  _ _ V  |  P
    ---------------|----           -------|---------|-------
(4)  T   |  _ _ _  |  G        (7)  S G T  |  _ _ _  |  G A
     T   |  A B A  |  G            S G T  |  A B A  |  G A
```

Figure 6.4: The *Bubble* main components: L, X, and R.

As it can easily be seen, each kernel alignment forms an instance of a transformation of a sentence portion, within a given context, determined by the left and right segments. Hence bubble `(1)` provides evidence for the equivalence between sequence `[B,S]` and sequence `[T]`, when the left context is `[A]` and the right one `[G,T]`. Whenever the X sequences have different lengths, we use the term transformation instead of equivalence, i.e. we interpret it as a transformation from the longer sequence to the shorter one. The longer sequence is still labeled with the X letter and for the shorter one we use letter Y. So in the previous example, we have a transformation from X = `[B,S]` into Y = `[T]`, within the mentioned contexts. Therefore, one may represent a *bubble* as 3-tuple (a 3-ary relation), as shown below:

$$bubble \equiv bub(L,\ X \xrightarrow{transf} Y,\ R) \tag{6.1}$$

The bubble shown above respects a general pattern which can be specialized by substituting its variables by terms. So, for the three bubbles shown in Figure 6.4 we obtain the following specialized term representations:

$$bubble_{(1)} \equiv bub([A],\ [B,S] \xrightarrow{transf} [T],\ [G,T])$$
$$bubble_{(4)} \equiv bub([T],\ [A,B,A] \xrightarrow{transf} [],\ [G])$$
$$bubble_{(5)} \equiv bub([T],\ [A,T] \xrightarrow{transf} [V],\ [P])$$

Figure 6.5: The *bubble* term representation.

Having the main objective of learning sentence reduction rules, we focus our attention on this structure - the *bubbles* - selected from the corpus of aligned sentences. We noticed that there exist other pair-sub-segment types worth to be explored, however, due to the complexity revealed of those structures, we decided to first concentrate our efforts towards this structure. In fact during this

research we decided to work only with specific types of bubbles - those where the middle region of one pair aligned sub-sequence is aligned with a void segment ($X \xrightarrow{transf} \emptyset$). In the future, more general transformations could be investigated, including transformations of the form $X \xrightarrow{transf} Y$, where $Y \neq \emptyset$ and $|X| > |Y|$, with $|\bullet|$ being the segment size in terms of the number of words contained.

In the following Figure 6.6, we show some examples of bubbles extracted from real data - the corpus of aligned paraphrases, created from web news stories:

```
(1)  the situation here in chicago with the workers
     the situation ____ in chicago with the workers

(2)  obama talks exclusively with tom brokaw on meet
     obama talks ___________ with tom brokaw on meet

(3)  Ball and Arnaz were divorced in 1960
     Ball and Arnaz ____ divorced in 1960

(4)  america is in the exact same seat as sweigert and
     america is in ___ _____ same seat as sweigert and

(5)  after a while at the regents park gym , the president
     after a while at ___ _______ ____ gym , the president
```

Figure 6.6: Examples of extracted bubbles

To extract a *bubble*, from a word aligned paraphrase, left and right contexts of equally aligned words are required, according to Definition 3, and the probability of such extraction depends on these contexts sizes as well as the size of the middle region aligned with the empty sequence. We noticed that not all bubbles lead to useful rules and that performance improved when we further restricted our attention to bubbles where the surrounding context is sufficiently large when compared with the middle segment. Therefore, to decide whether a pair-sub-segment is a *bubble* the system computes the segment sizes, according to the condition in Equation 6.2. If this condition is verified then a bubble is extracted, giving rise to a new learning instance.

$$|L| + |R| \geq |X| \tag{6.2}$$

For example, in the first example from Figure 6.6, we would have: $2 + 5 > 1$. In the last example we have $4 + 4 > 3$. So on both cases this justifies the extraction of those bubbles. This condition prevents the extraction of the bubbles shown in Figure 6.7. We note that in both cases the size of the contexts is not large enough.

```
(6)  To the horror of their fans , Miss Ball and Arnaz ...
     __ ___ ______ __ _____ ____ _ ____ Ball and Arnaz ...

(7)  ... vote __ ___ _______ ____ __ _____ __ friday .
     ... vote on the amended bill as early as friday .
```

Figure 6.7: Examples of two rejected bubbles

With condition 6.2 we can define a normalized metric to calculate the value of a bubble, in order to quantitatively compare different bubbles. Thus we define $val_\beta(bubble)$ as follows:

$$val_\beta(\mathbf{bubble}) = val_\beta\left(\mathbf{bubble}_{(L,\, X \xrightarrow{transf} Y,\, R)}\right) = 1 - \frac{|X|}{|L| + |R| + \frac{1}{2}} \tag{6.3}$$

where $|X|$, $|L|$, and $|R|$ are respectively the middle, left, and right segment sizes, in terms of the number of words contained. This function has values in the $[0,1]$ interval as long as the condition 6.2 is preserved. If it is violated then we will have $val_\beta(\mathbf{bubble}) < 0$, which is the case for the examples in Figure 6.7 where $val_\beta(\mathbf{bub_6}) = 1 - \frac{8}{0+3+1/2} \approx -1.286$ and $val_\beta(\mathbf{bub_7}) = 1 - \frac{7}{1+1+1/2} \approx -1.799$. Table 6.1 shows the bubble values for those shown previously in Figure 6.6.

Table 6.1: Bubble values for the previous five examples.

bub:	(1)	(2)	(3)	(4)	(5)
$val_\beta(\mathbf{bub})$:	0.867	0.867	0.846	0.765	0.647

Following this methodology, we obtained a huge set of instances in which relevant sentence transformations occur. To give an idea about the amount of data involved, from a set of 30 days web news stories, which corresponds to approximately 133.5 MB of raw text collected, the system identified $596\,678$ paraphrases that were selected and subsequently aligned, and from this set $143\,761$ bubbles were extracted. These bubbles are then conveniently adapted to serve as learning instances for the *Aleph* system, which will then "discover" sentence segment transformation patterns and codify them as rules (clauses) that can be be applied in future for sentence simplifications.

6.2 System Execution

We will now detail a little bit the implementation and execution of the third module from the system's scheme in Figure 1.2, comprehending the automatic generation of learning instances and the execution of the induction process. First we present the main conceptual steps in this module, with algorithms 3 and 4. From the set of aligned paraphrases, stored in an XML file (line 1), the next execution towards the rule induction process consists in the extraction of bubbles and their transformation in *Aleph* learning instances (line 6). This gives rise to the *Aleph's* "`*.f`" file of positive examples (being generated in line 7).

6.2.1 Data Preparation

In terms of implementation, the bubble extraction process from a paraphrase list is executed through the java class "`ExtractBubbles`", contained in the `Factory.jar` package, as follows:

Algorithm 3 Bubble extraction.

1: $apraphlist \leftarrow LoadAlignedParaphrases(xmlfile)$
2: $bublist \leftarrow \emptyset$
3: **for** $aparaph \in apraphlist$ **do**
4: $\quad bubbles \leftarrow extractBubbles(aparaph)$
5: $\quad$ **for** $bub \in bubbles$ **do**
6: $\quad\quad instance \leftarrow learningInstance(bub)$
7: $\quad\quad$ *saveInAlephFfile*$(instance)$
8: $\quad\quad bublist \leftarrow bublist \cup \{bub\}$
9: $\quad$ **end for**
10: **end for**
11: $saveInFile(filename, bublist)$

Algorithm 4 Running the induction process.

1: $bublist \leftarrow loadFile(filename) \vee loadLists(directory)$
2: $template \leftarrow readAlephTemplate()$
3: $generateAlephFiles(template, bublists)$
4: $startInduction()$
5: $saveRules()$

```
java -Xms32m -Xmx1024m ExtractBubbles -inp pac.xml -fdata lbub.dat
```

In particular, the `-inp` parameter specifies the input file containing the aligned paraphrases[1], which here is `pac.xml` and the `-fdata` parameter indicates the name of the output binary file in which the list of extracted bubbles will be stored: `lbub.dat`, in this case[2]. This file will hold a binary instance of an `ListXBubble` object, which contains a list of bubbles (`XBubble` class), that were populated during the bubble extraction procedure, previously mentioned (Algorithm 3). Afterwards the *Aleph* instances and all its associated files[3] necessary to start an induction execution can be generated by running the following command:

```
java hultig.sumo.ListXBubble -aleph dir
```

This will be done by executing a special mode (defined through the `-aleph` parameter) of the `ListXBubble` class, which is also used to store a list of bubbles. In the previous execution command we are requesting to search the current directory for `*.dat` files containing binary lists of bubbles, previously extracted from aligned paraphrases. It corresponds to take the "$bublist \leftarrow loadLists(directory)$" branch in line 1, from Algorithm 4. A single directory may contain several bub-

[1] Related to algorithm's line 1.
[2] Related to algorithm's line 11.
[3] In our case files "*.f" and "*.b".

ble lists, obtained from different bubble extraction execution moments. In this case all the existing bubble lists will be gathered into a single list, before generating the *Aleph's* learning files.

For every bubble a learning instance is inserted in the `*.b` file, which is simply a Prolog term, similar to the 3-ary term shown previously in formula 6.1 and Figure 6.5. For example, the list of bubbles from Figure 6.6, is transformed in the five learning instances shown in Figure 6.8, listed in the same order. Indeed, each bubble is represented by a 5-ary term (`bub/5`) where the first argument is a sequential

```
bub(1, t(1,0),
    [situation/nn/np, the/dt/np],
    [here/rb/advp] ---> [],
    [in/in/pp, chicago/nn/np, with/in/pp]
).

bub(2, t(1,0),
    [talks/nns/np, obama/nn/np],
    [exclusively/rb/advp] ---> [],
    [with/in/pp, tom/nn/np, brokaw/nn/np]
).

bub(3, t(1,0),
    [arnaz/nnp/np, andcc/np, ball/nnp/np],
    [were/vbd/vp] ---> [],
    [divorced/vbn/vp, in/in/pp, 1960/cd/np]
).

bub(4, t(2,0),
    [in/in/pp, is/vbz/vp, america/nn/np],
    [the/dt/np, exact/jj/np] ---> [],
    [same/jj/np, seat/nn/np, as/in/pp]
).

bub(5, t(3,0),
    [at/in/pp, while/nn/np, a/dt/np],
    [the/dt/np, regents/nns/np, park/nn/np] ---> [],
    [gym/nn/np, ','/punct/punct, the/dt/np]
).
```

Figure 6.8: Bubbles converted in *Aleph* positive learning instances.

index. The second argument is a 2-ary term (`t/2`) indicating the kernel size transformation. For instance the term `t(2,0)` means that two words were in effect dropped (substituted by zero words). This second argument is related to the fourth one which represents such transformation through a 2-ary term, in the "`X ---> Y`" format, where `X` and `Y` are the lists of tagged words from the kernel transformation. Term `t(m,n)` requires that `m = lenght(X)` and `n = lenght(Y)`. Finally the third and fifth `bub/5` arguments are respectively the left and right contexts, which are also represented using lists of tagged words. For the sake of convenience of algorithmic processing, the left context list (third argument) is represented in a reversed order. These are the raw data to be used by the learning system. Afterwards, the execution of the last shown command will also generate the rest of the necessary *Aleph* configurations and preparations for starting an induction execution. Besides the bubble instances included in the `*.b` file, we also need the header set configurations and all the

auxiliary predicates necessary to the induction execution. This complementary material is generated from a pre-defined template file, since it is common to any inductive run.

The first part in the header section (see Figure 6.9) includes all the modes, determinations and types involved, according to the description and examples given in Section 5.4.

```
%-----------------------------------------------------------------------
% AUTOMATICALLY GENERATED BY:... "$PROG_NAME"
% INPUT FILE:................... "$INPUT_FILE"
% MOMENT:....................... $MOMENT
%-----------------------------------------------------------------------
%
% ALEPH PARAMETERS
:- set(minpos, 5).
:- set(verbosity, 1).
:- set(evalfn, user).

%-----------------------------------------------------------------------
% MODE DECLARATIONS
%-----------------------------------------------------------------------
:- modeh(1, rule(+bub)).

:- modeb(1, transfdim(+bub, n(#nat,#nat))).
:- modeb(3, chunk(+bub, #side, #chk)).
:- modeb(*, inx(+bub, #side, #k, #tword)).

:- determination(rule/1, transfdim/2).
:- determination(rule/1, chunk/3).
:- determination(rule/1, inx/4).

%-----------------------------------------------------------------------
% TYPE DEFINITIONS
%-----------------------------------------------------------------------
side(left).       side(right).
side(center:x).   side(center:y).

k(1).  k(2).   k(3).

chk(np).    chk(undefined).     chk(np).    chk(vp).
chk(pp).    chk(prt).           chk(advp).  chk(multi).

nat(X) :- entre(X,0,20).
```

Figure 6.9: The *.b *Aleph* template header.

Predicate `rule/1` is our goal head predicate and `transfdim/2`, `chunk/3`, and `inx/4` the names of the predicates that can be used in the rule body construction. The first one, `transfdim/2`, tries to generalize over the dimension transformation, related with the `t/2` term included inside the bubble terms, as shown previously. The predicate `chunk/2` seeks regularities over the chunk types and `inx/4` scans for lexical and syntactical (POS tags) regularities. The "Type Definitions" block clarifies the domains of the predicate arguments just mentioned, which are briefly described in the following list:

- `nat` - defined as a natural number between 0 and 20.
- `side` - the segment type, among the four possibilities: `left` and `right` contexts and the bubble's center `X` and `Y` components.
- `k` - a numeric value restricted just to the $\{1, 2, 3\}$ set, which are the allowed segment positions for scanning lexical and syntactical regularities.
- `bub` - the bubble's sequential index, also used as the `rule/1` index connector.
- `tword` - represents either a word or a part-of-speech tag.

An example of an induced rule, expressed in the *Aleph*'s *Prolog* language, is shown below:

```
rule(A) :- inx(A,left,1,the), chunk(A,center:x,np), inx(A,right,1,pos(nn)).
```

Note that this rule almost corresponds to the fifth rule presented in Figure 1.5. The only difference is that there the rule explicitly states that the central X segment must have size equal to 2. As it can be seen here, the first "`inx`" conditions states that the word "the" must occur in the first position of the left context, while the second "`inx`" condition says that the first position of the right context must contain a noun. The rule second condition imposes that the central X segment must be a noun phrase chunk.

```
cost(_, [P,_,L], Cost) :-
    value_num_literals(L, ValueL),
    Cost is P/$DATA_SIZE * ValueL
    .

value_num_literals(1, 0.10). %     |
value_num_literals(2, 0.25). % 1.0 -                   _
value_num_literals(3, 0.50). %     |                   _      _
value_num_literals(4, 1.00). %     |               _   _      _     .
value_num_literals(5, 0.60). %     |         _     _   _      _     _     .
value_num_literals(6, 0.40). %     |   _     _     _   _      _     _     _
value_num_literals(7, 0.20). %   ---------------------------------------------->
value_num_literals(_, 0.00). %         1     2     3     4     5     6     7
```

Figure 6.10: Our rule evaluation function, defined in the *.b template file.

In the file general settings the minimum positive coverage defined is equal to five "`set(minpos,5)`", which means that any induced rule must have at least five learning instances supporting it. The "`set(verbosity,0))`" setting defines the verbosity level, which in this case is switched to minimum verbosity, meaning that the output screen printings during the induction process should be minimalist. The third template general setting, "`set(evalfn,user)`", establishes that rule evaluation is made through a specially defined function ("user defined"), instead of using *Aleph*'s predefined evaluation. Usually this function is defined through the `cost/3` predicate, which in our case is shown Figure 6.10.

After several experiences related to our learning task, we have decided to implement our own rule evaluation function, which combines *positive coverage* with *rule length*, defined in terms of the number of body literals involved. The symbols `P` and `L` are input variables representing respectively the number of positive examples covered and the number of rule literals. The variable `Cost` represents the output value of the rule that is being evaluated. The number of literals (`L`) is defined with the `value_num_literals` predicate and as can be seen in Figure 6.10 we have decided to give higher preference (i.e. higher `L` value) to rules having three to five literals, since we find this well suited for our problem, where we aim at rules with at least one condition for each context segment, plus one for the center X segment. To even ensure this last statement, for any inducted rule, we have also defined *Aleph* constraints as shown in Figure 6.11. The first constraint rejects any learned

```
false :-
    hypothesis(rule(_), Body, _),
    num_literais(Body, N),
    N < 2
    .

false :-
    hypothesis(rule(_), Body, _),
    count_restr_zones(Body, NL, NX, NY, NR),
    not_valid(NL, NX, NY, NR)
    .

not_valid( 0,  0,  0,  0).  %----> any zone without constraints.
not_valid( _,  0,  _,  _).  %----> only in surrounding contexts.
not_valid( 0,  _,  _,  _).  %----> left context is free.
not_valid( _,  _,  _,  0).  %----> right context is free.
```

Figure 6.11: Special *Aleph* constraints included in our *.b template file.

rule with less than two literals in the body, and the second one ensures that any induced rule will have at least one literal ruling over each relevant bubble segment: left, right and center X. The "`count_restr_zones/5`" predicate, takes as input the body of a rule being evaluated, and outputs the number of literals found for each segment. Afterwards the "`not_valid/4`" imposes the desired restrictions. The complete *Aleph* background knowledge file template (`*.b`), with all its components may be consulted in Appendix C, in listings C.2, C.3, and C.4.

Both the rule evaluation function and the defined constraints, help to guide the induction process toward obtaining more relevant rules with better characteristics. Here it remains to mention that the `ListXBubble` command execution, shown previously, also generates the *Aleph* file of positive examples "`*.f`", which will just contain a list of `rule/1` terms, where their numeric argument is exactly the bubble index value, defined in the `bub/5` term.

6.2.2 Exemplifying the Process of Induction

We include here tracing excerpts showing how the system proceeds in inducing sentence reduction rules, accompanied by some comments in order to complement the description in the previous subsection.

The induction process is started on the command line by first invoking the *Aleph* program, which is a program written in *Prolog*. In our case we have been working with its *Yap* (Santos Costa *et al.*, 2002) implementation ("`yaleph`").

```
[john@JPCMacBook#0 2010-11-08 15:13:03] /a/news@google/test3days/m
$ yaleph
% Restoring file /usr/local/lib/Yap/startup
YAP version Yap-5.1.2
 % reconsulting /Users/john/bin/aleph.yap...

A L E P H
Version 5
Last modified: Sun Jun  4 10:51:31 UTC 2006

Manual: http://www.comlab.ox.ac.uk/oucl/groups/machlearn/Aleph/index.html

 % reconsulted /Users/john/bin/aleph.yap in module user, 102 msec 1127808 bytes
   ?-
```

After launching *Aleph* we have to load the learning input files (`*.b` and `*.f`) through the command "`read_all(+BaseFileName)`". In our case two additional files will be consulted (`*.lgt`, and `*.yap`), containing several auxiliary *Prolog* predicates. It is convenient to start *Aleph* with a folder containing these files.

```
   ?- read_all(lxbub).
 % reconsulting /a/news@google/test3days/m/lxbub.b...
  % reconsulting /a/news@google/test3days/m/lxbub.yap...
   % reconsulting /lib/prolog/utils.lgt...
   % reconsulted /lib/prolog/utils.lgt in module user, 2 msec 10176 bytes
   % reconsulting /lib/prolog/penpostags.lgt...
   % reconsulted /lib/prolog/penpostags.lgt in module user, 1 msec 4400 bytes
  % reconsulted /a/news@google/test3days/m/lxbub.yap in module user, 5 msec 35136 bytes
  % reconsulting /a/news@google/test3days/m/lxbub.lgt...
  % reconsulted /a/news@google/test3days/m/lxbub.lgt in module user, 992 msec 9804104 bytes
  % reconsulting /usr/local/share/Yap/system.yap...
   % reconsulting /usr/local/share/Yap/lists.yap...
   % reconsulted /usr/local/share/Yap/lists.yap in module lists, 3 msec 22768 bytes
  % reconsulted /usr/local/share/Yap/system.yap in module system, 10 msec 82912 bytes


DATA SET SIZE: 13140
 % reconsulted /a/news@google/test3days/m/lxbub.b in module user, 1011 msec 9940192 bytes
[consulting pos examples] [lxbub.f]
[cannot open] [lxbub.n]
yes
   ?-
```

Note that in the previous listing, in line 20, we have been informed that the system could not find the corresponding file with negative examples. This is naturally in agreement with our previous statement. At least for now we use just positive learning instances. Afterwards we start the induction process by invoking the *Aleph*'s "`induce`" command, which starts a *greedy cover removal search* (Srinivasan, 2000), meaning that for any *best clause* found the covered examples are removed.

```
1      ?- induce.
2   [select example] [1]
3   [sat] [1]
4   [rule(0)]
5
6   [bottom clause]
7   rule(A) :-
8      transfdim(A,n(39,0)), chunk(A,left,np), chunk(A,right,vp), chunk(A,center:x,multi),
9      inx(A,left,1,he), inx(A,left,1,pos(prp)), inx(A,right,1,has), inx(A,right,1,pos(vbz)),
10     inx(A,right,2,heard), inx(A,right,2,pos(vbn)), inx(A,right,3,so), inx(A,right,3,pos(rb)),
11     inx(A,center:x,1,proved), inx(A,center:x,1,pos(vbd)), inx(A,center:x,2,that),
12     inx(A,center:x,2,pos(dt)), inx(A,center:x,3,age), inx(A,center:x,3,pos(nn)).
13  [literals] [19]
14  [saturation time] [0.00800000000000001]
15  [reduce]
16  [best label so far] [[1,0,2,-inf]/0]
17  rule(A).
18  [constraint violated]
19  [13140/0]
20  rule(A) :-
21     transfdim(A,n(39,0)).
22  [constraint violated]
23  ... ... ...
24  ... ... ...
```

In this listing we can identify the initial bottom clause constructed (lines 6 to 12). The induction starts, by searching for generalizations (least general generalization) which entails this bottom clause. It tries to cover the training data as much as possible, without violating the predefined constraints. For example, in lines 18 and 22 we can see that the system indicated constraint violation for the two initial rules proposed, one in line 18, and the other one in lines 20 and 21.

The search for a good rule set will last a while, depending on the size of the training data. These times are of the order of tens of minutes. The time complexity analysis is reported in Section 6.4. During this search the system continues to output several items of information, including for example *good clauses* found, as shown in the following excerpt:

```
1   ... ... ...
2   [good clause]
3   rule(A) :-
4      chunk(A,left,np), chunk(A,right,vp), inx(A,center:x,1,pos(nn)).
5   [user defined cost] [0.0240487062404871]
6   [clause label] [[316,0,4,-0.0240487062404871]]
7   [clauses constructed] [317]
8   ... ... ...
```

A *good clause* is any clause with utility above a certain predefined minimum score (Srinivasan, 2000). It must also satisfying other parameters as well, like for example the minimum examples covered, which in our case was set to 5 (`:-set(minpos, 5).`). A *best clause* is a *good clause* having an higher score. For the final theory, clauses are chosen from the set of *best clauses* found. The `[clause label]` information, in line 6, reports the value of 4 parameters for that clause, in the form of `[[P,N,L,Val]]`, where `P` and `N` are respectively the number of positive and negative examples covered, `L` is the number o literals in the clause, and `Val` is the clause value.

In the end, the set of rules forming the theory is listed and saved in a file. The next excerpt shows an induction process ending:

```
1   ... ... ...
2   [theory]
3
4   [Rule 1] [Pos cover = 5 Neg cover = 0]
5   rule(A) :-
6      chunk(A,left,np), inx(A,right,1,pos(vbz)), inx(A,center:x,3,pos(nn)).
7   ... ... ...
8   [Rule 2426] [Pos cover = 520 Neg cover = 0]
9   rule(A) :-
10     chunk(A,left,pp), chunk(A,right,np), inx(A,center:x,1,pos(dt)).
11
12  [Rule 2436] [Pos cover = 7 Neg cover = 0]
13  rule(A) :-
14     chunk(A,right,np), inx(A,left,2,pos(nns)), inx(A,center:x,1,but).
15
16  [Rule 2441] [Pos cover = 5 Neg cover = 0]
17  rule(A) :-
18     chunk(A,left,punct), inx(A,right,1,police), inx(A,center:x,1,and).
19
20  [Training set performance]
21                Actual
22            +            -
23       + 12873           0          12873
24  Pred
25       -  267            0           267
26
27         13140           0          13140
28
29  Accuracy = 0.979680365296804
30  [Training set summary] [[12873,0,267,0]]
```

For any learned theory's rule, the positive and negative coverage are printed. In our case, the latter is naturally always zero. Just for ilustration, the rule 2426 (lines 8 to 10) covers 520 examples and states that if we have an *propositional phrase* (pp), followed by some segment starting with a *determiner* (pos(dt)), and this one is followed by a noun phrase (np), then there is strong evidence that the middle segment can be eliminated. One of the examples used for learning this rule is shown below:

```
bub(3883, t(3,0),
    [between/in/pp],
    [the/dt/np, start/nn/np, of/in/pp] ---> [],
    [november/nn/np, '2006'/cd/np, and/cc/undefined]
    ).
```

The induction listing finishes by showing the confusion matrix for the generated rule set. We can see that the theory covers 12873 examples, letting out 267 cases.

In this section we have presented some implementation issues, avoiding many rather specific details, some of them can be consulted in appendixes B, C, and D. The next section presents some of the learned rules and reflect on their quality.

6.3 Learned Rules and their Applications

The final output of our system is a set of sentence reduction rules, or more precisely, rules that can be used to drop some sentence portions, normally having low information value. It is desirable that our learned rules do not cut out relevant sentence portions and that our obtained sentence still maintain its principal meaning as well as a syntactically acceptable structure. Initially, in Section 1.3 we have provided a small set of such induced rules as an illustrative example. However, the

whole set is obviously much larger and it is important to stress out how can we apply such rule set for practical purposes and subsequently make evaluations.

The work endeavored throughout this research project is far from being at a complete, final and perfect stage, and by the contrary we look at it more as being the beginning of something very promising and worth following, where these were just the first steps. As the research become more and more developed and in order to avoid being lost inside many complex decision spaces, we found ourselves in a position where we had to maintain simplicity at most, or at least decide to confine our research aims to a more takable horizon. Therefore, after having decided to work only with *bubbles*, another complexity confinement was made by deciding to work only with those having middle size equal to one ($|X| = 1$), two ($|X| = 2$) or three ($|X| = 3$). One reason for that, was that the majority of the extracted *bubbles* (83.46%) have their middle sizes within this range, as can be seen in Figure 6.12. We have also decided to induce separate rules for each *bubble* type. Consequently a specific set of rules was learned for each size type. The generated rule set was quite large. For example, in total 5806 reduction rules were induced from a set of 37271 "$|X| = 2$" bubbles.

Some examples of sentence reduction rules are provided in Table 6.2. The eliminated sentence portion is marked in bold and the rule syntax follows the same as shown and explained previously in Section 1.3.

However, the system is not perfect as it can lead to erroneous sentence reduction rules, by eliminating wrong sentence portions, either by eliminating some key elements, or producing syntactically incorrect sentences. Table 6.3 show six erroneous situations of reduction rules applied to sentences. However, as our results show, the majority[1] of rule applications produce good or acceptable results. Here *acceptable* means that semantic key terms were preserved and that no major syntactical errors

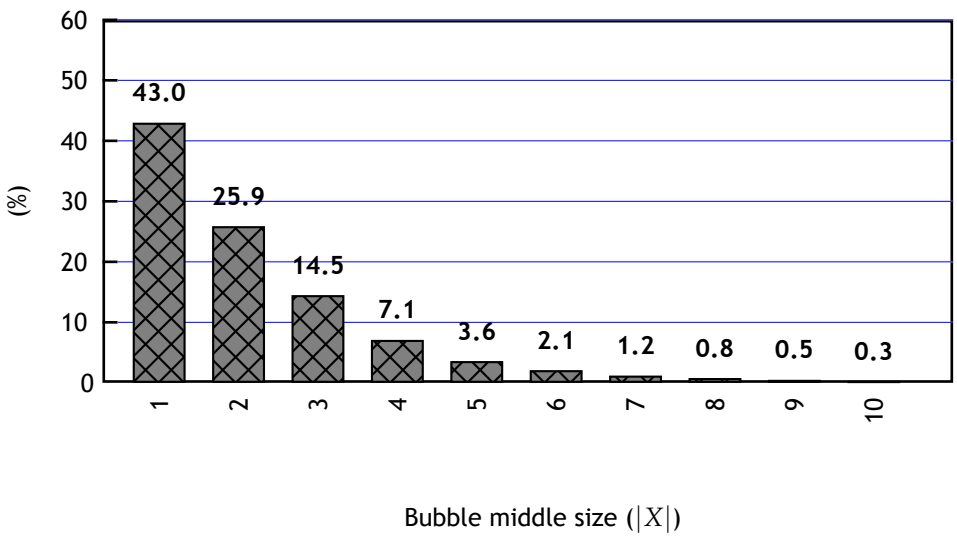

Figure 6.12: Bubble middle size distribution for a set of 143761 extracted bubbles.

[1] An $\hat{F_{0.5}}$ value of 73.01% (Section 7.3.1, table 7.10)

Table 6.2: Some examples of good sentence reduction rules showing the eliminated sentence portions in each case.

N	Sentence	Reduction Rule
1	*I want to move **now** to international affairs, the war on terror.*	$L_c = \texttt{VP} \wedge X_1 = \texttt{RB} \wedge R_1 = \texttt{to} \wedge \lvert X \rvert = 1$
2	*Andrea Mitchell is **also** a very good reporter but she's a lousy ...*	$L_c = \texttt{VP} \wedge X_1 = \texttt{RB} \wedge R_3 = \texttt{JJ} \wedge \lvert X \rvert = 1$
3	*My comment has **everything** to do with the way the media is covering Obama and nothing to do with racism.*	$L_c = \texttt{VP} \wedge X_1 = \texttt{NN} \wedge R_1 = \texttt{to} \wedge \lvert X \rvert = 1$
4	*Wasn't his entire campaign **therefore** more a product and function of greed than any before it?*	$L_1 = \texttt{NN} \wedge X_1 = \texttt{RB} \wedge R_3 = \texttt{NN} \wedge \lvert X \rvert = 1$
5	*With **the financial** crisis still worsening, obama said his main goal is finding a remedy for the economic woes.*	$L_c = \texttt{PP} \wedge X_c = \texttt{PP} \wedge R_1 = \texttt{NN} \wedge \lvert X \rvert = 2$
6	*Obama continued to say that the **domestic automotive** industry must put an even bigger emphasis than in the past on developing fuel-efficient vehicles and hybrid vehicles.*	$L_1 = \texttt{the} \wedge X_c = \texttt{NP} \wedge R_1 = \texttt{NN} \wedge \lvert X \rvert = 2$
7	*He rallied support for an auto bailout and the **massive economic** stimulus he announced this weekend.*	$L_1 = \texttt{the} \wedge X_c = \texttt{NP} \wedge R_1 = \texttt{NN} \wedge \lvert X \rvert = 2$
8	*This is the **kind of** behaviour any five-year-old child could see through, and habitually does.*	$L_c = \texttt{NP} \wedge X_2 = \texttt{of} \wedge R_c = \texttt{NP} \wedge \lvert X \rvert = 2$
9	*Complaining **that and washington** pols have done little for people facing foreclosure or struggling to get loans and jobs.*	$L_c = \texttt{VP} \wedge X_1 = \texttt{that} \wedge R_c = \texttt{NP} \wedge \lvert X \rvert = 3$
10	*Despite **the nation's** massive debt, Obama said he won't be focusing on building a balanced budget at the start of his administration.*	$L_c = \texttt{NP} \wedge X_1 = \texttt{the} \wedge R_c = \texttt{NP} \wedge \lvert X \rvert = 3$
11	*As **a first time** user, your comment has been submitted for review.*	$L_c = \texttt{PP} \wedge X_1 = \texttt{a} \wedge R_c = \texttt{NP} \wedge \lvert X \rvert = 3$
12	*Congressional leaders say **the emerging stimulus** program could cost between $400 billion and $700 billion.*	$L_c = \texttt{VP} \wedge X_1 = \texttt{the} \wedge R_1 = \texttt{NN} \wedge \lvert X \rvert = 3$

were introduced in the final sentence.

At this stage, a practical safety procedure is suggested, in order to minimize the inappropriateness of rule application to certain sentences that would result in syntactical errors and sentence discontinuities, as can be seen from the examples shown in Table 6.3. In order to achieve that, we applied *part-of-speech n-gram* models. First we have created the models from a large corpus of near 1 GB of text. This corpus was transformed into a "corpus" of *part-of-speech* terms like for example:

```
"(...) NNP VBZ RB VBG IN NNP IN NNS WDT MD VB IN DT NN IN DT
JJ NN IN NNP NN NN . RBR DT NN NN NNP NNP NNP VBD NNS TO VB
NNP IN DT NNP CC NNP NNP NN TO NNS VBN IN NNP . (...)"
```

Table 6.3: Six examples of bad sentence reduction rule applications.

N	Sentence	Reduction Rule
1	*I want to move **now** to international affairs, the war on terror.*	$L_c = \texttt{VP} \wedge X_1 = \texttt{RB} \wedge R_1 = \texttt{to} \wedge \lvert X \rvert = 1$
2	*Andrea Mitchell is **also** a very good reporter but she's a lousy ...*	$L_c = \texttt{VP} \wedge X_1 = \texttt{RB} \wedge R_3 = \texttt{JJ} \wedge \lvert X \rvert = 1$
3	*My comment has **everything** to do with the way the media is covering Obama and nothing to do with racism.*	$L_c = \texttt{VP} \wedge X_1 = \texttt{NN} \wedge R_1 = \texttt{to} \wedge \lvert X \rvert = 1$
4	*Wasn't his entire campaign **therefore** more a product and function of greed than any before it?*	$L_1 = \texttt{NN} \wedge X_1 = \texttt{RB} \wedge R_3 = \texttt{NN} \wedge \lvert X \rvert = 1$
5	*With **the financial** crisis still worsening, obama said his main goal is finding a remedy for the economic woes.*	$L_c = \texttt{PP} \wedge X_c = \texttt{PP} \wedge R_1 = \texttt{NN} \wedge \lvert X \rvert = 1$
6	*Obama continued to say that the **domestic automotive** industry must put an even bigger emphasis than in the past on developing fuel-efficient vehicles and hybrid vehicles.*	$L_1 = \texttt{the} \wedge X_c = \texttt{NP} \wedge R_1 = \texttt{NN} \wedge \lvert X \rvert = 2$

This excerpt of POS tags corresponds to the text block shown below:

> "(...) *Apple is also working with Intel on microprocessors that will serve as the heart of a new generation of Macintosh computer hardware. Earlier this year Apple CEO Steve Jobs announced plans to change Macs from the IBM and Motorola-manufactured PowerPC architecture to chips made by Intel.* (...)"

Afterwards, we employ the CMU-Toolkit (Clarkson & Rosenfeld, 1997) to compute 2-gram and 4-gram models and then use these models to ensure that the reduced sentence still maintains a "reasonable expectable POS sequence", meaning that we still have a likely syntactical sequence of words in the resulting sentence. To explain the method used, we first present it through Algorithm 5, which also presents the main document summarization procedure.

Function "summarizeSentences" (line 4) contains the main conceptual summarization algorithm we have implemented. It receives a document and for each one of its sentences tries to apply a reduction rule, previously learned (line 9). If the rule (r) match the sentence (s) being considered, a secondary test is performed in order to ensure that the resulting sentence (s') will have a likely sequence of POS tags (line 11). If so then the original sentence is updated to its reduced version (line 12), and at the end it is concatenated[1] to the summary being produced (line 16). Note that more than one rule may be applied to a given document sentence.

[1]The $\oplus$ operator represents string concatenation.

Algorithm 5 Summarization loop with model based safety procedure for avoiding bad rule application.

```
1:  rules ← LearnReductionRules(corpora)
2:  model ← loadNGramModel()
3:
4:  function summarizeSentences (document) : summary
5:  summary ← ∅
6:  sentences ← ListOfSentences(document)
7:  for s ∈ sentences do
8:     for ρ ∈ rules do
9:        if match(ρ, s) then
10:          s′ ← applyRule(ρ, s)
11:          if POSlikely(s′, model) then
12:             s ← s′
13:          end if
14:       end if
15:    end for
16:    summary ← summary ⊕ s
17: end for
18: end function
```

To explain the execution of the $POSlikely(s', model)$ test (line 11), consider a general sentence scheme, tagged as shown below, where w stands for *word* and t for POS tag:

$$s \;=\; w_1/t_1 \;\; w_2/t_2 \;\; w_3/t_3 \;\; [w_4/t_4 \;\; w_5/t_5 \;\; w_6/t_6] \;\; w_7/t_7 \;\; w_8/t_8$$

Let us assume that there exist some reduction rule ρ specifying conditions such that the middle bracketed sequence in s can be eliminated. According to the algorithm, this means that we would have:

$$s' \;=\; w_1/t_1 \;\; w_2/t_2 \;\; w_3/t_3 \;\; w_7/t_7 \;\; w_8/t_8$$

and s will be updated with s' only if the "$POSlikely$" test is true. Since this test is based on POS tag sequences, we must compare the likelihood of sequences $\langle t_1, t_2, t_3, t_4, t_5, t_6, t_7, t_8 \rangle$ and $\langle t_1, t_2, t_3, t_7, t_8 \rangle$ in a balanced/normalized way, i.e. the comparison should be independent from the whole sequence lengths. Therefore the middle sequence boundaries are considered for calculation. For example if a 2-gram test is performed then we compare the likelihood of $\langle t_3, t_4 \rangle$ and $\langle t_6, t_7 \rangle$, with the one obtained from $\langle t_3, t_7 \rangle$, which would be a resulting subsequence if the rule is applied. In this case, and weighting equally the middle sequence boundaries we would have the following test:

$$P\{\langle t_3, t_7 \rangle\} \geq \frac{\sqrt{P\{\langle t_3, t_4 \rangle\} * P\{\langle t_6, t_7 \rangle\}}}{2} \tag{6.4}$$

where "$P\{X\}$" stands for "the probability of X". The hypothesis here is that if the probability of the resulting POS sequence ($\langle t_3, t_7 \rangle$) is relatively low, then it might e better to ignore this rule. So, if the test from condition 6.4 holds then the rule is applied and the original sentence is updated to its new reduced version. A similar calculation can be made for 3-grams and 4-grams and then the probability values would be combined to make the best decision. In fact a 4-gram model was one of our first experiments.

However, the results reported in Subsection 7.3.1, were obtained from a sample of a differently generated dataset. In this case, instead of trying to apply all possible rules to a sentence, we focused on just applying the best rule matching the sentence, if at least one exists. So, in this case the previously shown algorithm (5) has become slightly different: Here ρ_{best} and q_{best} represents

Algorithm 6 Summarization loop choosing the "best" rule to apply.

```
rules ← LearnReductionRules(corpora)
model ← loadNGramModel()

function summarizeSentences (document) : summary
summary ← ∅
sentences ← ListOfSentences(document)
for s ∈ sentences do
    ρ_best ← ∅,   q_best ← 0
    for ρ ∈ rules do
        if match(ρ, s) then
            q ← ruleQuality(ρ, s)
            if q > q_best then
                ρ_best ← ρ,   q_best ← q
            end if
        end if
    end for
    if ρ_best ≠ ∅ then
        s′ ← applyRule(ρ_best, s)
        if POSlikely(s′, model) then
            s ← s′
        end if
    end if
    summary ← summary ⊕ s
end for
end function
```

naturally the "best" rule for sentence "s" and its "quality". So, the question that remains is how do we

calculate the rule quality (line 12 in Algorithm 6)? We consider a combination of three parameters, where one is a 2-gram model ratio, as shown in condition 6.4.

$$p = \frac{P\{\langle t_i, t_{i+n+1}\rangle\}}{\sqrt{P\{\langle t_i, t_{i+1}\rangle\} * P\{\langle t_{i+n}, t_{i+n+1}\rangle\}}} \tag{6.5}$$

The second parameter for rule quality calculation is the number of words eliminated by the rule, lets say n, to comply with the notation of Equation 6.5. The third parameter is the rule coverage, represented by "c". Each induced rule generated by *Aleph* has a coverage value associated with it, indicating the number of positive learning instances covered by the rule. Hence, the final rule quality is calculated through a weighted geometrical mean involving these three parameters:

$$ruleQuality(\rho, s) = e^{\frac{1}{2}\cdot\log(p)+\frac{1}{4}\log(n)+\frac{1}{4}\log(c)} \tag{6.6}$$

Thus, more importance is given to the n-gram model parameter (p) than the other two. This formula and its parameter weights were the result of several experiments carried out, where values were gradually tuned according to the obtained results.

6.4 Induction Time Complexity

In the earlier[1] days of ILP, the computation time spent by the existing systems was a serious difficulty and a critical bottleneck, disabling its implementation on real life problems. However, nowadays these time efficiency issues have been overcome, through more efficient implementation techniques and the steady growing of computational power, opening a wide range of application possibilities, for many problems, ranging from Biology to Natural Language Processing. The graph in Figure 6.13, shows that even with considerable big datasets, our *Aleph* based learning system evidences acceptable feasible computation times. These computation measures were taken from a machine with an *Intel Core 2 Duo* processor, working at 1.83Ghz, having 2GB of RAM, and running a POSIX[2] operating system.

To give an idea about the size of an induced rule set, and taking as an example the learned rules with $|X| = 2$, these were learned from a dataset containing 37271 $t(2,0)$ bubbles, and in the final 5806 sentence reduction rules were produced, through the induction process which took 53 minutes.

6.5 Final Remarks

In this chapter we focus on the work related with our last system's module - the induction of sentence reduction rules. We describe the learning data preparation used, which includes the selection of special structures from the aligned paraphrases called *bubbles*. These are transformed into learning instances that are subsequently used for rule induction in an ILP based learning system - *Aleph*.

[1] In the 1990-2000 decade.

[2] In our case the *Mac OS X*.

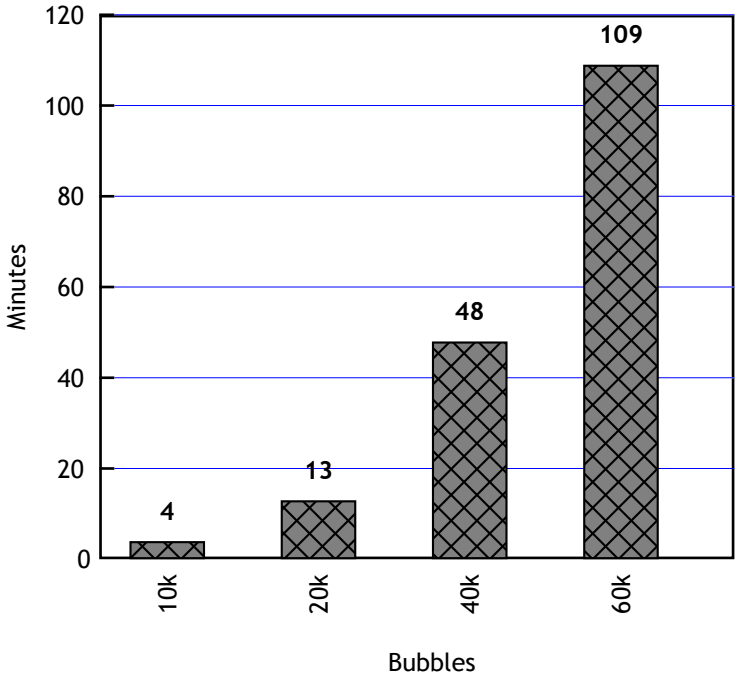

Figure 6.13: Time spent during the induction process, for datasets with size expressed in thousands of bubbles.

Although we have only considered *bubbles* with $|X| \leq 3$, a sentence may have a compression length greater than this value, since several compression rules may be applied to a single sentence. Even compositional rule application is likely. One or more rules are initially applied to a sentence, transforming it into a new (simpler) sentence where other reduction rules can fire and continue the sentence reduction process until no more rules match the sentence. An experiment we have carried out shows that for a collection of learned rules and a collection of sentences targeted for reduction, we have on average three rules firing in the first level for each sentence.

For the future, we have a large set of directions to follow, within this particular issue. We let here some ideas. Due to its complexity, we have not yet worked with bubbles where $X \xrightarrow{transf} Y$, with $Y \neq \emptyset$, and $|X| > |Y|$. These are likely to be relevant to induce rewriting rules. We also want to explore other structures, obtained from the alignments, that we may call "extremes", in which one of the paraphrasic sentences initial or final segment is aligned with an empty sequence. We also think that a better chunker, capable of identifying *subject-object* relations will generate better rules and improve correctness.

The work described in this chapter allow us to have another scientific contribution, published in proceedings of the *ACL 2009* conference (Cordeiro *et al.*, 2009). It was presented in workshop specially dedicated to text generation and summarization, entitled *Workshop on Language Generation and Summarisation* (*UCNLG+Sum*).

Chapter 7

Results

"Everything that can be counted does not necessarily count; everything that counts cannot necessarily be counted."

Albert Einstein

This chapter gathers a set of results and main conclusions, from various experiments conducted throughout the research work of this thesis. The main research subjects: paraphrase extraction, paraphrase word alignment, and sentence reduction, were reported in chapters 3, 4, and 6 respectively. In this chapter, sections 7.1, 7.2, and 7.3 discuss the corresponding experiments carried out and provide the respective results obtained.

7.1 Paraphrase Extraction

In this work we have carried out an investigation concerning how to identify paraphrases. A set of already known and used functions was tested and new functions proposed. Therefore an experimental comparative study included nine functions in conjunction with three paraphrase corpora described as follows.

Two standard corpora were used for comparative tests of different metrics: the *Microsoft Research Paraphrase Corpus* (Dolan *et al.*, 2004) and a paraphrase corpus supplied by Daniel Marcu that has been used in many other experiments, namely for *sentence compression* (Knight & Marcu, 2002; Nguyen *et al.*, 2004). By making some adaptations in these two corpora, we have obtained three new paraphrase corpora to serve as a benchmark for our comparative experimentations. In the following subsections we give a more detailed description concerning the method used in this process.

7.1.1 The Microsoft Paraphrase Corpus

In 2005 Microsoft researchers (Dolan *et al.*, 2004) published the first freely available paraphrase corpus containing a total of 5801 sentences pairs, with 3900 annotated as "semantically equivalent" or true paraphrases and the remaining 1901 with the opposite label, stored as negative paraphrase

examples. Throughout the rest of our text we will refer to this corpus as *MSRPC*[1].

This corpus was constructed by selecting paraphrases from massive parallel news sources using a semi-automatic process, where first a shallow sentence similarity function was employed as well as a heuristic permitting to pair initial sentences from different news stories as paraphrases. The shallow function employed was the well-known *Sentence Edit Distance* (see Section 3.2.1). Afterwards human

1. The sentences have different content: prototypical example.
2. The sentences share content of the same event, but lacking details.
3. One cannot determine if sentences refer to the same event.
4. Shared content but different rhetorical structure.
5. The sentences refer to the same event but details different emphasis.

Figure 7.1: Guidelines used by human judges in the *MSRPC* paraphrase corpus construction to determine equivalent pairs.

judges were included in the process to analyze and rate each automatically extracted pair. Three human judges classified independently each pair as being "equivalent" (paraphrases) or "not equivalent" pairs. To help them in the decision process, specially in the more difficult cases, a set of five principles or guidelines was elaborated beforehand, stating essentially that a sentence pair should not be marked as a paraphrase if one condition from the list in Figure 7.1 holds. These guidelines were oriented towards the extraction of symmetrical paraphrases (see Section 3.3) and therefore do not quite comply with our broader view of the types of paraphrases that should be considered. Since our main research objective here is not paraphrasing, but sentence reduction or simplification, we are also interested in sentence pairs of different generality or information content. In general such pairs provide information about sentence reduction transformation that we wanted to explore. As explained in Section 3.3, we are specially interested in *asymmetrical paraphrases* and so some of the guidelines formulated for the *MSRPC* corpus construction conflict with this objective, in particular the second and fifth directives from the list in Figure 7.1. For instance, although sentences **(a)** and **(b)** below would be rejected as a paraphrase according to the *MSRPC* construction guidelines, they form a relevant sentence pair for our research purposes.

(a) *Researchers have identified a genetic pilot light for puberty in both mice and humans.*

(b) *The discovery of a gene that appears to be a key regulator of puberty in humans and mice could lead to new infertility treatments and contraceptives.*

Figure 7.2: An asymmetrical paraphrase pair likely to be rejected according to the *MSRPC* construction guideline.

[1] An abbreviation of *Microsoft Research Paraphrase Corpus*

Hence it can be expected that our proposed new functions for asymetrical paraphrase identification (see Section 3.3) would yield a certain amount of disagreement over some *MSRPC* negative examples, by identifying them as positive, i.e. true asymmetrical paraphrase cases. On the other hand, it increases the number of false positives and this phenomenon was in fact experimentally confirmed (see Table 7.3 in Section 7.1.5).

7.1.2 The Knight and Marcu Corpus

The corpus used by Knight & Marcu (2002) in their *sentence compression* work contains 1087 sentence pairs, where for each pair one sentence is a compressed or summarized version of the other one. Similarly to what we did for the Microsoft Paraphrase Corpus we also labeled this one and referred to it as the *KMC* corpus. It was manually crafted by selecting related sentences from a collection of $\langle Text, Summary \rangle$ pairs, one sentence from each category the expanded one from the *Text* and its compressed version from the *Summary*. This corpus was constructed to serve as a *supervised learning* data set. Such a construction is very laborious and time consuming and even if the corpus has a significant size it may always be incomplete in terms of linguistic diversity. By the time we started this research and even during it, this was the only existing sentence reduction corpus available. Despite its richer content, it was clearly not sufficient for our sentence reduction rule induction objective, since we aimed at following an unsupervised approach and preferred to rely on massive data gathered automatically, in order to capture a wide and relevant number of rules. However, even if *KMC* is not our primary data source it is still very relevant to test paraphrase extraction functions, consisting in a golden[1] asymmetrical paraphrase test set. Furthermore, it helps balancing the overall paraphrase test set, since on one hand we have the *MSRPC* containing exclusively symmetrical pairs, and on the other hand the *KMC* corpus with only asymmetrical paraphrase types.

These two corpora were conveniently adapted and combined in order to provide relevant test sets for the paraphrase identification functions.

7.1.3 The Evaluation Paraphrase Corpora

One major limitation with the *KMC* corpus is that it only contains positive examples and therefore it should not be taken as such for evaluation. Indeed, it is necessary to add an equal number of negative examples, pairs of sentences which are not paraphrases, in order to obtain balanced evaluation set for the paraphrase detection. Even the *MSRPC* corpus is fairly unbalanced, having only 1901 negative examples compared to the 3900 positive ones. To perform a fair evaluation, we decided to expand both corpora by adding negative examples, randomly selected from *web news texts*, in a sufficient number to balance the corpora, and so as to end up with the same number of positive and negative examples. Finally we also decided to create a third corpus, which is a combination of both, *MSRPC*

[1] The *KMC* corpus was manually selected.

and *KMC*, to which we added a new set of negative cases. In Figure 7.3, an example of a "random negative pair" is depicted in order to clarify this point. It is evident that the previous example is in fact

(a) *Running back Julius Jones is expected to return after missing the last three games with a sprained left ankle.*

(b) *Economists closely watch the performance of consumer spending since it accounts for two-thirds of total economic activity.*

Figure 7.3: An example of a negative paraphrase pair, randomly selected from *Web News Texts*

a negative paraphrase case with sentences having nothing in common. On the other hand, one may expect that among a set of news stories talking about a given event it is likely to encounter sentence pairs that are almost equal (quasi-equal), even coming from different texts. Such pairs are naturally irrelevant for our research purposes (see the discussion in the beginning of Section 3.3). Hence, our objective is that our proposed paraphrase identification functions should be immune to these kind of pairs, by deciding to reject them. Therefore in order to test this feature one may consider to include a low percentage of such negative cases, like the one shown in Figure 7.4. In Subsection 7.1.7, we

(a) *"I don't think my age matters during competitions", said Nadal.*

(b) *I don't think my age matters during the competitions, said Nadal.*

Figure 7.4: An example of a quasi-equal negative paraphrase pair.

present a discussion and some evaluation based on this issue of adding at random selected sentence pairs consisting of quasi-equal sentences as negative examples. We show that it does not affect the performance of our method, as the performance differences between symmetrical paraphrase (SP) and asymmetrical paraphrase (AP) function types are not affected by the existence of negative random pairs.

7.1.3.1 The $MSRPC \cup X^{-}_{1999}$ Corpus

This new derived corpus contains the original *MSRPC* collection of 5801 pairs (3900 positives and 1901 negatives) plus 1999 extra negative examples (symbolized by X^{-}_{1999}), selected from web news stories. So we end up with 3900 positive pairs and 3900 negatives.

7.1.3.2 The $KMC \cup X^{-}_{1087}$ Corpus

From the *KMC* corpus we derived a new corpus that contains its 1087 positive pairs plus a set of negative pairs, in equal number, selected from *web news stories*. We named this new corpus as $KMC \cup X^{-}_{1087}$, where the X^{-}_{1087} represents the extra negative paraphrase examples, resulting once again in a balanced test set with an equal number of positive and negative cases.

7.1.3.3 The $MSRPC^{+} \cup KMC \cup X^{-}_{4987}$ Corpus

Finally, we decided to build a larger corpus that gathers the positive *MSRPC* part, with 3900 symmetrical positive examples and 1087 positive asymmetrical paraphrase pairs from the *KMC* corpus, summing up to a total of 4987 positive pairs, of which 21.8% are asymmetrical. To balance the corpus, an equal number of negative pairs was added, obtained in the same fashion as described previously for the other two corpora. We labeled this larger corpus as $MSRPC^{+} \cup KMC \cup X^{-}_{4987}$. In this corpus, we decided to exclude the *MSRPC* negative pairs, according to what is discussed in Subsection 7.1.1.

The data components needed to reconstruct these corpora are available in the following website: http://www.di.ubi.pt/∼jpaulo/competence/[1]. The *MSRPC* corpus may be obtained directly from its source[2]. The *KMC* corpus was not publicly available by the time we have conducted our experiences and it was directly provided by Knight & Marcu (2002). Therefore, we only provide the negative examples that were used in our tests which were joined with the positive sets to derive the three main test sets: X^{-}_{1999} joined with *MSRPC*, the X^{-}_{1087} set joined with *KMC*, and the X^{-}_{4987} set for the third one. These data will enable any future reader to reconstruct each paraphrase corpus described, and make comparative experiments.

7.1.4 How to Classify a Paraphrase?

As in many machine learning problems, in paraphrase identification/extraction a system makes decisions taking into account certain parameters, which are called *thresholds* and are pre-set by someone: the user, the programmer, the researcher or even others. In our paraphrase extraction task, which involves a classification problem, the evaluation of the performance for any paraphrase identification function, is dependent of a given predefined threshold θ, which conditions the classification. That is, for a given sentence pair the system computes a proximity or dissimilarity value, which determines whether the pair is or is not a paraphrase. For example, assuming that we are evaluating the function $paraph(.,.)$, which calculates the likelihood that any two sentences S_a and S_b, are a paraphrase pair, and also assuming that $\theta = 0.6$, then the pair $\langle S_a, S_b \rangle$ would be classified as a paraphrase if and only if we had $paraph(S_a, S_b) > 0.6$.

Thresholds are parameters that make the process of a fair evaluation more difficult. Indeed the best possible threshold parameter should be determined first for a given function. However, this is not always the case and, very often, wrong experimental set-up is used.

In order to avoid this mistake, and since we are comparing the performance of our proposed paraphrase identification functions with other already existing ones, our evaluation process, does not

[1] Link available on October 2010.

[2] http://research.microsoft.com/nlp/ [October 2010].

pre-define any threshold. Instead, the evaluation procedure will first automatically set the best threshold function value for a given training corpus. Afterwards the function is tested in the test corpus. We made this by using *10-fold cross validation*. For each run, we divide a given paraphrasic corpus in two parts: $\frac{9}{10}$ of data for training and $\frac{1}{10}$ for testing. Therefore any function is tested with its best threshold, the one that maximizes the function performance in the training corpora. This is equivalent to stating that the paraphrase extraction functions are compared at their maximum possible performance.

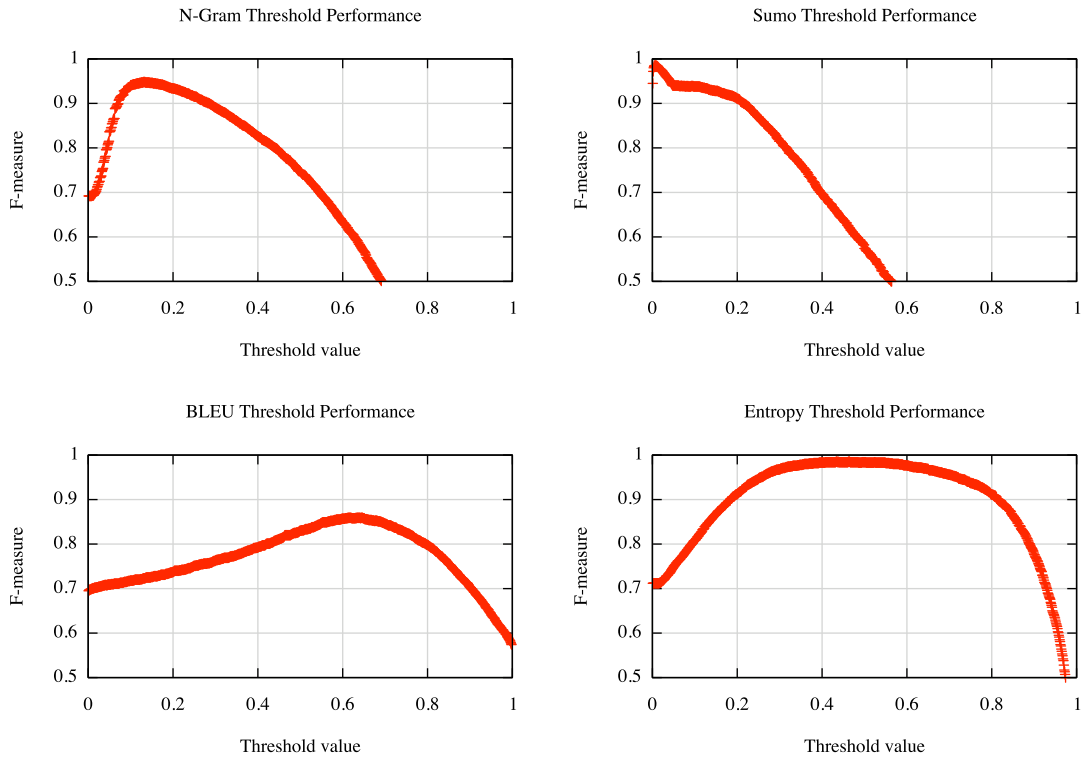

Figure 7.5: Threshold performance curves for 4 paraphrase identification functions, tested on the $MSRPC^{+} \cup KMC \cup X^{-}_{4987}$ corpus.

The pre-scanning of the best threshold is a typical problem of function maximization or optimization (Polak, 1971), and in order to efficiently find a good approximation to the global maximum, a *golden section search* method was employed (Anita, 2002), which progressively divides and narrows a function interval towards the location of a function extreme. It is guaranteed to converge to the global maximum, if the function is unimodal. We noticed that the performance function graphs are smooth curves and almost all reveal the unimodal characteristic[1], that is a kind of inverted concavity with a unique global maximum point. In Figure 7.5 we present four of such curves, for the N-Gram, BLEU, Sumo and Entropy paraphrase classifiers. These four graphs were produced through an exhaustive point drawing where nearly 1000 threshold values were sequentially taken in the $]0,1[$ interval and their respective classifier performance computed for a specific corpus, which in this case was the

[1] At least all that we have tested and reported here.

Table 7.1: Threshold means and standard deviations, obtained using a 10-fold cross validation procedure.

thresholds	$\mathbf{MSRPC} \cup \mathbf{X}^{-}_{1999}$	$\mathbf{KMC} \cup \mathbf{X}^{-}_{1087}$	$\mathbf{MSRPC}^{+} \cup \mathbf{KMC} \cup \mathbf{X}^{-}_{4987}$
Edit	17.003 ± 0.0000	18.800 ± 1.1353	17.000 ± 0.0000
Sim_o	0.1668 ± 0.0000	0.2096 ± 0.0138	0.1298 ± 0.0008
Sim_{exo}	0.5000 ± 0.0000	0.7205 ± 0.0382	0.5000 ± 0.0000
Bleu	0.6887 ± 0.0015	0.0204 ± 0.0051	0.6369 ± 0.0077
Sumo	0.0718 ± 0.0033	0.0049 ± 0.0010	0.0070 ± 0.0001
Trignom	0.2889 ± 0.0803	0.2685 ± 0.0000	0.3421 ± 0.0000
Parabolic	0.5091 ± 0.0146	0.3242 ± 0.0001	0.3255 ± 0.0048
Entropy	0.6166 ± 0.0065	0.3851 ± 0.0011	0.4455 ± 0.0319
Gaussian	0.5250 ± 0.0095	0.3670 ± 0.0006	0.3986 ± 0.0105

largest one used - the $MSRPC^{+} \cup KMC \cup X^{-}_{4987}$ corpus.

Given this, the *golden section search* algorithm was used to quickly find the low error approximation for the best function threshold, on the training corpus. Afterwards a test was run with that threshold in the test corpus. To provide evidence that we are computing a good approximation of the global optimum, we show in Table 7.1 the means and standard deviations of all function thresholds, obtained in each corpus. Each threshold mean is obtaining by averaging the ten results coming from a 10-fold cross validation. As already mentioned, a given paraphrasic corpus is divided in ten folds, where nine of them are used to tune the best threshold through the *golden section search* method. Then the paraphrase extraction function is tested with that threshold, on the left out fold.

The obtained thresholds (Table 7.1) and their variation range is almost negligible, supporting the fact that for these functions the *golden section search* method provides a good approximation of the global optimum.

7.1.5 Experiments and Results

In order to make a comparative evaluation of all paraphrase identification functions, a 10-fold cross-validation (cv) method was followed for each function and each paraphrase test set. The performance was judged using the well known *F-Measure*[1] and *Accuracy*, as shown in equations 7.3 and 7.4. As

[1]Which is also known as F_β.

usual, *precision* and *recall* were calculated as shown in Equation 7.1 and 7.2

$$precision = \frac{TP}{TP + FP} \tag{7.1}$$

$$recall = \frac{TP}{TP + FN} \tag{7.2}$$

where TP, FP, and FN are the *True Positive*, *False Positive* and *False Negative* values respectively, obtained from the *confusion matrix*. In a classification problem this matrix confronts the predicted

Table 7.2: The *confusion matrix* for a binary classification problem.

	Positive	***Negative***
Positive$_{predict}$	*TP*	*FP*
Negative$_{predict}$	*FN*	*TN*

outcomes with the actual classifications obtained by the system. Usually the rows refer to the predictions and the columns to the actual values. The general structure of a confusion matrix for a binary classification problem is shown in Table 7.2.

$$F_\beta = \frac{(1 + \beta^2) * precision * recall}{\beta^2 * precision + recall} \tag{7.3}$$

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{7.4}$$

We took the F_β evaluation measure with $\beta = 1$ meaning that precision and recall are equally weighted:

$$F_1 = \frac{2 * precision * recall}{precision + recall} \tag{7.5}$$

The following tables 7.3 and 7.4, provide a summary of the *F-Measure* and the *Accuracy* values obtained for each function and each paraphrase test set. We have three test sets and nine functions, with five of these specially defined to identify symmetrical type paraphrases (***Edit***, Sim_a, Sim_{exo}, ***Bleu***.). The $Edit$ function is the sentence edit distance, while Sim_o and Sim_{exo} are the simple ant the exclusive N-gram overlap functions, respectively. Detailed description about these functions are given in Section 3.2. The three paraphrase corpora cover a wide range of paraphrase types, from a "pure" symmetrical type ($MSRPC \cup X^-_{1999}$) to an asymmetrical type ($KMC \cup X^-_{1087}$), and also a larger corpus including both types ($MSRPC^+ \cup KMC \cup X^-_{4987}$).

Table 7.3: F_1 evaluation results obtained (in %).

Type	Function	$\mathbf{MSRPC} \cup \mathbf{X}^-_{1999}$	$\mathbf{KMC} \cup \mathbf{X}^-_{1087}$	$\mathbf{MSRPC}^+ \cup \mathbf{KMC} \cup \mathbf{X}^-_{4987}$
SP	*Edit*	74.42	71.06	80.97
	Sim_o	78.94	95.34	94.72
	Sim_{exo}	76.54	91.06	86.27
	$Bleu$	78.72	69.26	85.86
	average:	**77.16**	**81.68**	**86.96**
AP	*Sumo*	80.93	98.29	98.52
	Trignom	77.12	61.40	87.02
	Parabolic	80.18	97.55	98.40
	Entropy	80.25	97.50	98.35
	Gaussian	80.20	97.56	98.37
	average:	**79.74**	**90.46**	**96.13**

Table 7.4: $Accuracy$ obtained (in %).

Type	Function	$\mathbf{MSRPC} \cup \mathbf{X}^-_{1999}$	$\mathbf{KMC} \cup \mathbf{X}^-_{1087}$	$\mathbf{MSRPC}^+ \cup \mathbf{KMC} \cup \mathbf{X}^-_{4987}$
SP	*Edit*	67.68	69.26	79.27
	Sim_o	73.85	95.16	94.48
	Sim_{exo}	70.51	90.36	84.49
	$Bleu$	74.00	57.88	86.18
	average:	**71.51**	**78.17**	**86.11**
AP	*Sumo*	78.18	98.29	98.52
	Trignom	71.63	69.22	87.99
	Parabolic	75.74	97.58	98.39
	Entropy	75.88	97.24	98.34
	Gaussian	75.92	97.56	98.37
	average:	**75.47**	**91.98**	**96.32**

The results show that, in general, the AP-functions outperform the SP-functions on all paraphrase corpora, except for the trigonometric function (defined in 3.10), which does not behave very well and is worse than the majority of the AP-functions on asymmetrical corpora $KMC \cup X^-_{1087}$ and $MSRPC^+ \cup KMC \cup X^-_{4987}$. For instance, on the largest corpus $MSRPC^+ \cup KMC \cup X^-_{4987}$, the AP-functions correctly classified, on average, 96.32% out of all (9974) sentence pairs, which include both positive or negative examples, being near 10% better than the SP-functions.

As we expected, the AP-functions performed better on corpora containing asymmetrical pairs, like

in $KMC \cup X^{-}_{1087}$ and $MSRPC^{+} \cup KMC \cup X^{-}_{4987}$, revealing an *Accuracy* difference of 12.02%, by considering the average performance of each type (SP and AP) on both corpora and an equivalent *F-Measure* difference of 8.98%. Nevertheless, even in the $MSRPC \cup X^{-}_{1999}$ corpus, which is exclusively symmetrical, the performance of the AP-functions is surprisingly better than that of the SP-functions, revealing an average differential of 2.58% in terms of *F-Measure* and 3.96% in terms of *Accuracy*.

Within the SP-function class, the simple word N-gram overlap (sim_o) and the exclusive LCP N-gram overlap (sim_{exo}) metrics always obtain the first and the second place respectively, whereas the *Bleu* metric and the *Edit Distance* obtain the worst results on all paraphrase corpora. Within the AP-function class our proposed *Sumo* function obtained the best results. Note that the performance differences among the functions in this class is very small, with one exception, which is the trigonometric function.

7.1.6 The (SP type) Bleu Function - A Special Case

By looking at the *Bleu* function, in Subsection 3.2.2.2, it is clear that this function should indeed be interpreted as a set of functions that depends on the upper sum limit (N) chosen. It was set to 4 in our experiments, following indications in literature, as for example Barzilay & Lee (2003). They propose to use $N = 4$ for overlap n-grams, that is consider n-grams with length 1, 2, 3 and 4. By choosing different values for the N limit, a question arises wheter this would imply any significant difference in performance. We have carried out such tests and observed that for values greater than 4 the performance starts to decrease rapidly. It seems that the main reason for this lays in the fact that the *Bleu* function makes a product of N factors (see Equation 3.6). So if just one of these is a near zero then the overall function performance will be affected. As shown in Table 7.5, when N decreases from 4 to 1 the *Bleu* performance tends to improve. We note that in these experiments

Table 7.5: The $BLEU$ results (in %), with N decreasing.

F_β	$\mathbf{MSRPC \cup X^{-}_{1999}}$	$\mathbf{KMC \cup X^{-}_{1087}}$	$\mathbf{MSRPC^{+} \cup KMC \cup X^{-}_{4987}}$
N=4	76.62	69.32	84.68
N=3	77.82	69.33	87.86
N=2	78.77	68.88	90.18
N=1	79.39	77.45	91.15

the best performance was achieved for $N = 1$, when only unigram lexical links were counted. We followed this strategy for our proposed AP-functions, such as $Sumo$, which relies only on the number of words from each sentence and the number of exclusive lexical links (see Figure 3.5). Therefore counting 2-gram, 3-gram, ..., n-gram overlaps would not bring any benefit, but only increase the computational cost.

7.1.7 The Influence of Random Negative Pairs

In Section 7.1.3 we described a method to artificially add sentence pairs, randomly picked from *Web News Texts* as negative paraphrase pairs, to balance[1] the corpora. We also mentioned that it is likely that quasi-equal pairs could be inserted too as negative examples, depending on from where we obtained our texts. This last option may naturally be criticized, since these quasi-equal pairs can be seen as symmetrical paraphrases artificially labeled as negatives, and for such pairs the SP-functions would likely return *true*, whereas the AP-functions would give *false* as an answer. This type of examples will be counted as *True Negatives* by the AP-functions, but as *False Positives* by the SP-functions, and so one may say that the evaluation is biased towards AP-functions. However, our intent was simply to have a test corpora resembling, as near as possible, to real data, where such functions would be applied to identify and select instances that facilitates the construction of asymmetrical paraphrase corpora. On the other hand, these quasi-equal pairs were inserted randomly with very low probability, and at the end their number in the corpus was less than 1%.

Indeed, to acknowledge this situation and confirm that the performance differences are independent of the presence of quasi-equal pairs, we performed another experiment with a corpus similar to the $MSRPC^{+} \cup KMC \cup X^{-}_{4987}$, but without such pairs. We named this new corpus here as the "BigPure". Tables 7.6 and 7.7 present the results for both paraphrase identification function types, which confirm our claim that the performance difference between AP and SP type functions is independent of the presence of a small amount of quasi-equal pairs.

Table 7.6: SP-functions on a corpus without quasi-equal negative pairs

BigPure	edit	sim_o	sim_{exo}	bleu
F-measure	80.97%	94.72%	86.25%	85.86%
Accuracy	79.27%	94.48%	84.49%	86.18%

Table 7.7: AP-functions on a corpus without quasi-equal negative pairs

BigPure	Sumo	Trignom	Parabolic	Entropy	Gauss
F-measure	98.51%	87.02%	98.40%	98.35%	98.37%
Accuracy	98.50%	87.99%	98.40%	98.35%	98.37%

For this last experiment, the magnitude of the test data ensures us that at least with 99% statistical confidence[2] (1% significance) we have $Accuracy_{(AP\text{-}functions)} > Accuracy_{(SP\text{-}functions)}$ and $F\text{-}Measure_{(AP\text{-}functions)} > F\text{-}Measure_{(SP\text{-}functions)}$.

[1] Equal number of positive and negative examples.

[2] By making a statistical test for Accuracies: $H_0 : accuracy_{ap} = accuracy_{sp}$ against $H_1 : accuracy_{ap} > accuracy_{sp}$, and a similar one for the F-Measure.

7.2 Paraphrase Alignment

In this section we report the results obtained for the alignment algorithms presented in Chapter 4. Algorithms for word alignment between paraphrasic sentences were conveniently adapted from *Bioinformatics* and a new method to dynamically choose between a global and local alignment were also proposed. In order to assess the accuracy of these alignment methods for our domain, we designed a test involving human manual evaluation.

7.2.1 Paraphrase Corpora

Automatic evaluation of sequence alignment requires a gold standard data test set. As far as we are aware of, such data is unavailable for our type of problem, since we are proposing a new alignment method, specially adapted to our specific domain of word alignment between sentences. Although sequence alignment is not our main research line, it is part from the whole process, aim to automatically learn sentence reduction rules from text. We decided to carry out a manual evaluation of a sample of aligned paraphrases in order to get a quantitative measure of the quality of the alignments.

We tested our alignment method by using two sets of aligned paraphrases obtained from different text sources. One was obtained from the raw text material we are working with - *web news stories*. A random sample of 200 extracted aligned paraphrases were gathered as one test set and for representation convenience we will refer to it as the WNS. This dataset represents the type of alignments obtained from our working data, from which mainly asymmetrical paraphrases were extracted. However, in certain cases some symmetrical pairs could also be extracted. Therefore we also considered the possibility to test our alignment method in a different dataset, with exclusively asymmetrical paraphrase pairs. For that purpose we decided to use texts from the series of the *Document Understanding Conference* (DUC). The DUC series are competitions holding once a year which aim is to compare the performance of different sentence reduction systems on a wide range of specific tasks. After each competition, a scientific conference takes place where the main comparisons and conclusions are reported. Thus it was relatively easy to select a corpus containing solely asymmetrical paraphrase pairs from the DUC corpora. We decided to pick the DUC 2002 dataset of text pairs, where each pair is formed by a text and its corresponding summary. From these text pairs, 405 asymmetrical paraphrases were extracted and aligned. A random sample of 220 pairs[1] was then chosen to constitute the second evaluation test set. This test set is referred here simply as the DUC. Both for the WNS and the DUC, paraphrases were extracted using the *Sumo* function (see Subsection 3.3.1) and their words aligned using the proposed methodology described in Chapter 4. A difference between the WNS and the DUC is that in the latter dynamic alignment was used, while in the former only global alignment was performed.

[1] Composed of 200 globally aligned and 20 locally aligned.

7.2.2 Quality of Paraphrase Alignment

For each dataset, selected as previously described, a human judge evaluated the quality of each paraphrase alignment using four labels, two positive ones and two negative ones: *excellent*, *good*, *flawed*, and *bad*. Label *excellent* was assigned to alignments without any error at all. These are the perfect cases which are most useful for the subsequent rule induction step. Below are three examples of aligned pairs that were marked with this label (alignments 1, 2, and 3).

Excellent:

```
(1)  ____ He has helped the Police Department make hundreds of arrests and has " apprehended "
     Nero __ ___ helped the police __________ make hundreds of arrests and ___ _ apprehend   _

     more than 50 felons .
     more than 50 felons .

(2)  _____ ____ _______ __ ___ ___ ___ " That's the lowest pressure ever ________ measured
     Winds were clocked at 175 mph and _ ______ the lowest pressure ever recorded ________

     in the Western Hemisphere ___ ________ __ ___ ______ , " Zimmer said .
     in the Western Hemisphere was measured at its center _ _ ______ ____ .

(3)  kenya ____ has also claimed that the ________ ___ arms are destined for its ___ military .
     kenya says ___ ____ _______ ____ the weaponry was ____ ___ ________ for its own military .
```

The *good* label is used for alignments with at most two misalignments without having major implications on the overall quality of the alignment. Below we show two examples of such alignments (alignments 4 and 5).

Good:

```
(4)  fossett , 63 , vanished in his ___ ____ ____ _____ __ ____ airplane after taking off
     fossett _ __ _ ________ __ has not been seen since he took ________ _____ ______ off

     from a private air strip    in northern nevada on september 3 , 2007 .
     from a private ___ airstrip in ________ nevada in september _ _ 2007 .

(5)  Checkpoint Charlie , the Berlin Wall border post ______ ______ __ that symbolized the
     Checkpoint Charlie , the ______ ____ ______ most famous symbol of ____ __________ the

     Cold War , was ______ hoisted into history today .
     Cold War _ was lifted _______ into history today .
```

The negative label *flawed* is assigned to those cases where several misalignments were made. Despite the fact that good matches are present, the amount of wrong word alignments has an overall negative effect. Two examples of such cases are shown below, in alignments 6 and 7.

Flawed:

```
(6)  _______ __ ____ _ With ___ tropical storm force winds extending 250 miles north and
     Massive in size , it   had ________ storm _____ winds extending 250 miles north and
```

```
     200 miles south of the hurricane's center , Gilbert also was one of the largest .
     200 miles south of ___ ___________ ______ _ _______ ____ its ___ __ eye _______ .

(7)  honda _____ _____ _____ _______ __ _______ ___ _______ ______ ____ _ reported a
     honda motor co.'s sales dropped 24 percent and general motors corp . reported a

     ____ _______ 24 percent drop .
     15.8 decline __ _______ ____ .
```

And finally the second negative label *bad* is obviously assigned to completely erroneous alignments, as exemplified with the next two examples (alignments 8, and 9).

Bad:

```
(8)  The __ best America could do was six medals , its worst Winter _______ Games
     The US ____ _______ _____ __ had ___ ______ _ its worst Winter Olympic _____

     showing in 52 years ___ ____ _____ .
     showing in 52 years the 1988 Games .

(9)  there is so far no sign of the single-engine aircraft in which fossett , 63 , ________
     _____ __ __ mr  __ ____ __ ___ _____________ ________ __ _____ fossett , 63 , vanished

     __ _________ ____ ____ _____ __ _ ____ ______ ____ took off from ____________ a nevada
     in september last year while on a solo flight that took off from neighbouring _ nevada

     ranch in september 2007 .
     _____ __ _________ ____ .
```

Even here some tokens got well aligned. However, the overall quality of the alignment is poor and misleading for our next step of induction of sentence reduction rules. It is noteworthy that we are comparing alignment qualities in a deeper and more semantic level than comparisons based on our previously defined lexical "$algval(A_{2\times n})$" function (Equation 4.9). For instance this function seems to misevaluate some of these harder examples (Table 7.8), where in fact we have relatively less words aligned, yet still making sense as alignments between paraphrases (alignments 2 and 3). On

Table 7.8: The $algval(A_{2\times n})$ values for the previous nine examples shown.

ID:	$algval(A_{2\times n})$	ID:	$algval(A_{2\times n})$	ID:	$algval(A_{2\times n})$
1_{excel}	0.847	4_{good}	0.643	7_{flaw}	0.429
2_{excel}	0.588	5_{good}	0.743	8_{bad}	0.638
3_{excel}	0.581	6_{flaw}	0.676	9_{bad}	0.495

the other hand, there are situations where a great number of words are aligned with few yet critical errors for the overall alignment semantics (specially in alignments 6 and 8).

In the case of the DUC corpus, dynamic alignment was used yielding several local alignments. We asked the judge to classify these alignments as *adequate* or *inadequate*. We were interested in finding out whether the decision concerning the choice of the local alignment algorithm was adequate,

whenever such choice was made by the system. The results obtained are shown in Table 7.9. The

Table 7.9: Accuracy obtained in the alignments .

	Global				Local
Dataset	*excellent*	*good*	*flawed*	*bad*	*adequate*
DUC	67%	22%	7%	4%	80%
200+20	134	44	14	8	16/20
WNS	46%	28%	9%	17%	-
200	92	56	18	34	-
DUC ∪ WNS	56.5%	25.0%	8.0%	10.5%	80%
420	226	100	32	42	16/20

last column contains information concerning the adequacy of local alignments. There are 16 adequate ones from a total of 20 local alignments. As it can be seen the results from each dataset are related, being better on DUC data. We suppose that the main reason for this is that WNS data are somewhat more noisy, containing more erroneous cases, including errors coming from the previous steps, like for example sentences wrongly divided. On the other hand, with respect to DUC data, the text has been manually pre-selected, ensuring higher quality at the beginning of the process. The overall performance (DUC ∪ WNS) of the alignment method is quite satisfactory, since it achieved 81.5% of positive (*excellent* & *good*) alignment classifications. This provided us with confidence about the usage of these alignment techniques and the corresponding alignments obtained to be used in the next step, where *machine learning* techniques for sentence reduction rule induction were explored.

7.3 Application of Reduction Rules

The automatic evaluation of a system is always the best way to perform it, due to its objectivity and scalability. However, in many cases such evaluation is unfeasible for several practical reasons, like for example the unavailability of data or the huge difficulty to prepare it appropriately. In supervised learning, manually labeled test sets are used to evaluate the systems. However, these tend to be relatively small, affecting adversely the statistical significance of the obtained results. For example in Knight & Marcu (2002) a set of 1035 sentences was used to train their sentence compression system and only 32 sentences were used to test it, which is indeed a quite small test set. It is also important to propose many different evaluation techniques. In order to assess well the performance of our methodology on large datasets, we propose a set of qualitative and quantitative evaluations based on three different measures: estimated precision ($\widehat{precision}$), estimated recall ($\widehat{recall}$), and *syntactic N-gram simplification* ($ng\sigma$). These parameters are explained next together with the evaluations

carried out.

7.3.1 Evaluation Measures

The first evaluation measure carried out was based on human judgments of the quality of the application of sentence reduction rules. For a random sample of compressions, resulted from the application of sentence reduction rules, two human judges rated each rule application with an integer value ranging from 1 up to 5, where 1 means a total erroneous rule application, and 5 a perfect rule application. So, averaging human ratings, we have obtained a qualitative evaluation expressed quantitatively in the $[1,5]$ interval, which we call *correctness*. The *weighed Cohen's kappa coefficient* (Cohen, 1968) for inter-rater agreement obtained was equal to 64.89%, which means "substantial agreement", according to the guidelines of Landis & Koch (1977). The ratings were aggregated in three main categories: *negative* (1, 2), *neutral* (3), and *positive* (4, 5).

Precision and *recall* are evaluation measures normally employed in supervised learning, where for a given test set we know exactly how many elements belong to each class. These two measures are calculated from a *confusion matrix* as described previously in Subsection 7.1.5. In our case of sentence reduction rule application, we are evaluating an unsupervised approach where *precision* and *recall* can not be exactly calculated. Nevertheless, we decided to use an approximation of these measures. *Precision* is a measure of exactness calculating the percentage of correct positive classifications made from the total of instances positively classified by the system. For example in *information retrieval precision* is the number of relevant documents obtained from the total number of documents retrieved. In our specific problem here we approximate precision by normalizing correctness as proposed in Equation 7.6, where n is the number of sentences for which a rule is applied.

$$precision \approx \widehat{precision} = \frac{\sum_{k=1}^{n} rate(reduction_k)}{5 \cdot n} = \frac{correctness}{5} \tag{7.6}$$

The *recall* measure is a measure of completeness, calculating the percentage of positive classifications effectively made from the total of all positive classifications that the system could have made. In the example of *information retrieval*, this is the number of relevant documents found divided by the number of relevant ones existing in the collection. In order to have an approximation for *recall* too, we considered the number of sentences for which at least one reduction rule was applied, in a given text sentence collection being used for evaluation. This number is divided by the collection size. Let us assume that we are using a collection of m sentences $\mathcal{S} = \{s_1, ..., s_m\}$ for evaluating a given ruleset $\mathcal{R}$. Then recall is approximated as in Equation 7.7.

$$recall \approx \widehat{recall} = \frac{\#\, reductions_{\mathcal{R}}(\mathcal{S})}{m} \tag{7.7}$$

where "$\#\, reductions_{\mathcal{R}}(\mathcal{S})$" is the number of reduced sentences obtained from $\mathcal{S}$ through $\mathcal{R}$.

The $F\text{-}Measure$ (Equation 7.3) combines *precision* and *recall* in a single evaluation metric. By varying the β parameter, we are able to increase the importance of *precision* ($\beta < 1$) or *recall* ($\beta > 1$). For our $F\text{-}Measure$ approximation we decided to give more relevance to $\widehat{precision}$ than $\widehat{recall}$. We think that our defined $\widehat{recall}$ is a less accurate approximation than $\widehat{precision}$, because not all $\mathcal{S}$ sentences are eligible to be summarized. We may already have a subset of quite simple sentences contained in $\mathcal{S}$, for which any further simplification is no more possible without wiping out necessary fundamental components/information. Therefore, we decided to use an $F\text{-}Measure$ approximation with $\beta = 0.5$ ($F_{0.5}$):

$$F_{0.5} \approx \hat{F_{0.5}} = \frac{1.25 * \widehat{precision} * \widehat{recall}}{0.25 * \widehat{precision} + \widehat{recall}} \tag{7.8}$$

The *syntactic n-gram simplification* methodology is an automatic extrinsic test performed in order to perceive how much a given set of sentence reduction rules would simplify sentences in terms of syntactical complexity. The answer is not obvious at first sight and smaller sentences are not necessarily simpler, because even smaller sentences can contain more improbable syntactical sequences. In terms of syntactical analysis we are just working with shallow techniques, in particular here we are only considering the *part-of-speech tags* (POS)[1]. To compute the syntactical complexity of a sentence, we used a 4-gram model and computed a normalized[2] sequence probability as defined in Equation 7.9 where $\vec{W} = [t_1, t_2, ..., t_m]$ represents the POS sentence sequence, with size m of any sentence of length m

$$complex(\vec{W}) = \left(\prod_{k=n}^{m} P\{t_k \mid t_{k-1}, ..., t_{k-n}\} \right)^{\frac{1}{m-n+1}} \tag{7.9}$$

where $P\{t_k \mid t_{k-1}, ..., t_{k-n}\}$ means the conditional probability of tag t_k given the tag sequence $t_{k-1}, ..., t_{k-n}$. For example, for the sentence:

"The/`DT` passage/`NN` of/`IN` the/`DT` bill/`NN` by/`IN` an/`DT` overwhelming/`JJ` majority/`NN`."

we will have $\vec{W_1}$ = `[DT,NN,IN,DT,NN,IN,DT,JJ,NN]`, and assuming a simplified version of this sentence to be:

"The/`DT` passage/`NN` of/`IN` the/`DT` bill/`NN` by/`IN` majority/`NN`."

its corresponding $\vec{W_2}$ sequence would be equal to `[DT,NN,IN,DT,NN,IN,NN]`. As such, we can compare the POS sequence complexities of $\vec{W_1}$ and $\vec{W_2}$ through Funtion 7.9 and take note if $complex(\vec{W_2}) < complex(\vec{W_1})$ or not. It is naturally expected that simplified sentences will result in less complex POS sequences, i.e. with lower $P(\vec{W})$ value, meaning naturally a more likely sequence. This computation is based on a n-gram model of POS sequences and the n variable in Function 7.9 corresponds exactly to the n-gram used, in our case $n = 4$. The parameters of the POS 4-gram model were trained over a corpus of *web news stories* with approximately 1GB text size. In our particular example $complex(\vec{W_2})$

[1]For POS tagging the *Penn Treebank Tag Set* was employed.

[2]Because it is raised to the inverse power of $m - n + 1$, which is the number of factors being multiplied.

would be equal to

$$\Big(P(\texttt{DT}|\texttt{DT},\texttt{NN},\texttt{IN}) * P(\texttt{NN}|\texttt{NN},\texttt{IN},\texttt{DT}) * P(\texttt{IN}|\texttt{IN},\texttt{DT},\texttt{NN}) * P(\texttt{NN}|\texttt{DT},\texttt{NN},\texttt{IN})\Big)^{\frac{1}{4}}.$$

As usual in these cases, the log probability is taken to avoid round-offs to zero, since calculating the exact values is irrelevant, and the most important is to compare the complexity of different sequences. Thus instead of applying directly Function 7.9, we propose its log transformation as in Function 7.10.

$$Lcomplex(\vec{W}) = \frac{1}{m-n+1} * \sum_{k=n}^{m} Log(P\{t_k \mid t_{k-1}, ..., t_{k-n}\}) \tag{7.10}$$

The third evaluation is qualitative. We measured the quality of the learned rules when applied to sentence reduction. The objective is to assess how correct the application of the reduction rules is. This evaluation was made through manual cross annotation for a statistically representative random sample of compressed sentences. Two human judges evaluate the adequacy and *correctness* of each reduction rule to a given sentence segment, in a scale from 1 to 5, where 1 means that it is absolutely incorrect and inadequate, and 5 that the reduction rule fits perfectly to the situation (sentence) being analyzed.

7.3.2 Evaluation Results

In the evaluation, a sample of 300 sentences was randomly chosen, where at least one compression rule had been applied. This evaluation set was subdivided into three subsets, where 100 instances came from rules with $|X| = 1$ (**BD1**), 100 from rules with $|X| = 2$ (**BD2**), and the other 100 from rules with $|X| = 3$ (**BD3**). Another random sample, also with 100 cases was chosen to evaluate our baseline (**BL**), which consists in the direct application of bubbles to make compressions, meaning that no learning process is performed at all, but instead, we store the complete bubble set as if they were rules, in a similar way as Nguyen *et al.* (2004), using them directly for sentence compressions. For any sentence, if there exists a bubble match then simplification is made in an identical format as specified by that bubble.

Table 7.10 shows the comparative results for *correctness*, $\widehat{precision}$, $\widehat{recall}$, F_1, and *n-gram simplification*, for all datasets. The *precision* is simply a normalized *correctness*, obtained from dividing it by five which is the maximum correctness mark. The percentage values for *n-gram simplification* are the proportion of counted test cases where $P\{reduced(\vec{W})\} \geq P\{\vec{W}\}$ is verified.

Table 7.10 provides evidence of the improvement achieved with the induced rules in comparison with the baseline **BL**, on each measure. Considering the three experiences, **BD1**, **BD2**, and **BD3**, as a unique evaluation run, we can see the obtained results in the last line of the table (**BD**). For each evaluation parameter the average results on these three datasets are shown. Comparing these values with the baseline (**BL**) we can see improvements on every evaluation parameter.

Table 7.10: Results with four evaluation parameters.

	correctness	$\widehat{precision}$	$\widehat{recall}$	$\hat{F}_{0.5}$	$ng\sigma$
BL	2.97	59.40%	8.65%	27.33%	47.39%
BD1	3.43	68.60%	32.67%	56.23%	89.33%
BD2	3.52	70.40%	85.72%	73.01%	90.03%
BD3	3.49	69.80%	26.86%	52.89%	89.23%
BD	3.48	69.60%	48.42%	60.71%	89.53%

Moreover, best results overall are obtained for **BD2** with 70.4% $\widehat{precision}$, 85.72% $\widehat{recall}$, which gives an $\hat{F}_{0.5}$ value of 73.01%. For the *N-gram simplification* parameter ($ng\sigma$) we still obtain the best result of 90.03%, on **BD2**. This means that we can expect a reduction of two words with high quality for a great number of sentences. Some examples of compressed sentences using our induced rules can be found in tables 6.2 and 6.3.

Chapter 8

Conclusion and Future Directions

"*There is nothing as to begin, to see how hard it is to conclude.*"

Victor Hugo

Sentence Reduction continues to be an active research topic, where several relevant contributions have been achieved. The pursuit of better sentence simplification systems, resembling more to what humans do has motivated many to follow this line of research. We see Logic, and especially ILP systems, as useful strategies to achieve further advances, since they are able to represent better the human knowledge. The work presented here describes the first research step, with recourse to powerful computing tools, in which knowledge in the form of rules was obtained from a huge set of *news articles*, electronically available nowadays.

The core of our system, as schematized earlier in Figure 1.2, consists of an ILP learning system able to induce relational rules from learning instances. Obviously, as any leaning system, it depends on the amount and quality of learning instances supplied. Therefore we spent a considerable effort on the issue of data preparation, aiming to automate this process. Despite the difficulties and challenges of such automation, it was worthwhile and yielded the possibility to obtain relatively large amounts of learning examples, enabling a more reliable rule induction.

This chapter concludes our work, giving a final overview about what we did (Section 8.1), highlighting the innovative aspects proposed, as well as related scientific contributions. We conclude this chapter by pointing out to some relevant future directions worth to be explored.

8.1 Conclusions

After several years of research while preparing this thesis, we have finally reached the point where it is important to look back and observe the directions we followed, as well as look forward, to see how to propose possible future research issues. A research work is certainly never linear and direct, but includes many slopes, curves, and valleys, involving a combination of methodic work, sometimes with unpredictable outcomes, leading many times to dead ends. In several cases one must stop and backtrack to choose new directions, and our work was no exception.

In our quest for a more effective and autonomous process for simplifying sentences, we have designed a system constituted by several independent and interconnected modules, organized in a pipeline fashion. On one side *web news stories* enter and on the other side we obtain the sentence reduction rules, as it was initially schematized in Figure 1.2. Related *web news stories* are automatically collected on a daily basis, and from such corpora, specific functions are used to automatically identify and extract paraphrases, which is our first module (Chapter 3). In our second module (Chapter 4), classical *Bioinformatics* algorithms for DNA sequence alignment are conveniently adapted for word alignments of paraphrase sentence pairs. These sentence alignments are important for exploring related dissimilarities between paraphrasic sentences, enabling the learning of sentence equivalencies and transformational rules. In our third module, specific sentence aligned sequences, named *bubbles*, are automatically extracted from the set of aligned paraphrase pairs, and converted into learning instances, which are then used for the induction of sentence reduction rules. This is done in our fourth module (Chapter 6), by using an *Inductive Logic Programming* system (*Aleph*). Finally our fifth module is responsible for applying induced rules to new sentences, included in new web texts (Section 6.3).

Comparing our work to previous ones in the field of Sentence Reduction/Compression described in Chapter 2, it can be noted that our work follows an innovative approach to the problem of sentence simplification, by including several new techniques and much more automation, which provide our system with the capability to process large amounts of real text. Contrarily to several relevant approaches in this area, which are exclusively based on linguistic knowledge (Clarke & Lapata, 2006; Jing, 2000; Jing & McKeown, 2000), our work is almost completely independent of it. The only resources used are a POS tagger and a shallow syntactical analyzer from the OpenNLP[1] project. In particular, both exploit *machine learning* techniques, which means that they can be adapted to new languages, by providing convenient training examples to generate new language models. Besides these two linguistic tools, our system is automatic and does not require any human intervention, which is relevant for a possible migration to other languages, not requiring specific linguistic knowledge or resources.

Other approaches found in the literature that exploit *machine learning*, use *supervised learning* (Knight & Marcu, 2002; Turner & Charniak, 2005; Witbrock & Mittal, 1999), or a combination of this with linguistic knowledge (predefined rules), as in Vandeghinste & Pan (2004). However, we saw a great difficulty in using *supervised learning* for our problem, since we are dealing with rich linguistic varieties, containing many complex cases. Even with a considerable amount of manually crafted training examples, such systems are likely to fail to capture the whole phenomena by only inducing a relatively small number of reduction rules. For example, the Knight & Marcu (2002) system was trained with a very small dataset, and so in this case the learned rules when used in new unseen

[1] URL: http://opennlp.sourceforge.net/ [November 2010]

documents may only apply to relatively few sentences. On the other hand, the migration of such a system to other languages would require a huge amount of human labour, to provide new training sets. Our implemented system is based on *unsupervised learning* techniques, and the training examples are automatically mined and constructed from web texts. Thus, a language migration would not imply any major difficulty, provided that texts were electronically available for that language.

One other tendency in the literature is inspired by *automatic machine translation* methods, where the main procedure consists essentially of observing a large number of translation cases, named as *templates*, and "memorizing" them, so that they can be applied in the future (Nguyen *et al.*, 2004). However, this approach acquires rules ("sentence simplification templates") lacking in generalization power. This is again due to the reduced size of the learning set, since it is again based on *supervised learning*. While in *automatic machine translation*, high volumes of parallel texts are electronically available, this is not the case for sentence reduction, where these texts must be directly supplied by humans, which is a serious bottleneck. Our system does not require this work, since a kind of simplification templates, named *bubbles*, are automatically extracted in a very large number. Then an induction process, with the recourse to *Inductive Logic Programming*, is carried out, yielding a generalized set of sentence reduction rules.

With respect to our main scientific achievements, this work resulted into five relevant publications, one in a scientific journal and the others in the proceedings of important scientific conferences. The first article (Cordeiro *et al.*, 2007a) is concerned with our initial proposed function - the *Sumo* function - for paraphrase identification in corpora (Subsection 3.3.1). The second published article (Cordeiro *et al.*, 2007d) discusses word alignments in paraphrasic sentences, described in Chapter 4. Our third publication (Cordeiro *et al.*, 2007c) starts with our *Sumo* function and goes a bit further, by proposing a new family of functions sharing several common characteristics appropriate for asymmetrical paraphrase identification in corpora. This work contains also a comparative study with the conventional existing functions for shallow symmetrical paraphrase identification (section 3.3). Our last publication (Cordeiro *et al.*, 2009) is related to the induction process using *Inductive Logic Programming* for learning sentence reduction rules, presented in Chapter 6. Our second and fourth publications were made at workshops of the *Association for Computational Linguistics* conferences, and this last one in a specific workshop about language generation and summarization. Several adapted/improved versions of some of the techniques presented in this thesis have already been incorporated in new research tasks beyond the scope of this thesis. This has produced a recent publication in the proceedings of the COLING's 2010 conference (Grigonyté *et al.*, 2010). A more detailed list of these articles is given in Section 1.4.

8.2 Future Trends

Any PhD research work is certainly never definitely concluded, since many research issues, related to the initial problem still remain open and even new ones are unfold. Therefore, it is important to look into the future and consider what seems worthy to be continued. There are research opportunities in each one of our three main research lines: *extraction*, *alignment* and *induction*. Several suggestions have been already made in the final sections of the corresponding chapters.

Paraphrase Extraction. For paraphrase extraction deeper linguistic techniques could be experimented with. However, these would necessarily require significantly more computation time, since huge amounts of texts need to be processed during paraphrase mining. The complexity is $O(n^2)$ in the number of processed sentences. However, as results indicate, the performance could not be much improved, since the performance achieved with the proposed lexically-based techniques already reads a satisfactory level in terms of *F-Measure* for the AP-functions (Table 7.7). A shallower technique that can be experimented consists in the introduction of *multiword units* (Dias *et al.*, 2000) for enhancing the quality of the paraphrases extracted. In this case the paraphrase identification function can be adapted to account for different word/multiword lengths, where longer overlapping terms would get proportionally heavier weights in the final calculation of the sentence pair connection strength. In fact, we have recently made a first experiment with this for two specific domains (computer security and medic domain). This was included in the publication (Grigonyté *et al.*, 2010).

Paraphrase Alignment. In terms of paraphrase word alignments, we see several issues worth exploring, such as trying new word alignment functions, thereby introducing some semantic relations, or by trying to improve the dynamic method that chooses at run-time between global and local alignment. Another possibility is to start dividing longer sentences into their main thematic components, in particular long sentences having several conjunctions. In these cases, one may split the initial sentence and work with each part separately. We think that this could decrease significantly the alignment errors reported in Subsection 7.2.2. Another idea worth experimenting, either for paraphrase extraction or alignment, is to use a thesaurus and work with synonyms. We may have terms representing synonymy classes of words or even multiwords. For example, the word *speak* may represent "spoke", "talk", "chat", and "speak up", among other possibilities.

Rule Induction. The induction component of our system is the one which contains much "unexplored territory", and a possibility exists of using different paraphrasic text structures, besides *bubbles*. We can observe that *bubbles* are more appropriate to induce a kind of local sentence simplification, i.e. small sentence segments having at most five or six words. In fact, *bubbles* with larger kernels are hard to find in corpora, implying fewer data is available for induction and consequently less accurate theories are learned. By looking at the aligned paraphrase data it is possible to observe certain regularities in longer aligned segments, different from *bubbles*, at the extremes ends - at

the beginning and at the end of sentences. Such structures are more complex but could probably yield richer and more effective reduction rules. So far, we have only worked with $X \xrightarrow{transf} \emptyset$ type *bubbles* (see Section 6.1), while more general $X \xrightarrow{transf} Y$ types could also be experimented with. That is, instead of only looking at conditions in which a sentence segment can be dropped, we may also investigate those in which it can be substituted by an equivalent shorter segment.

Active Learning. Another possibility to improve the learned rules is to use *active learning* to incrementally refine the induction in subsequent learning cycles. We can for example manually collect cases of negative rule applications to sentences[1] and then use them as negative feedback to be added to the learner, that initiates a new learning cycle. As mentioned, we do not use negative examples in our first learning cycle. However, we think that the inclusion of negative evidence in the learner would yield more specialized and accurate rules.

Rule Application. In terms of rule application, we see a newly open question that needs to be addressed in the future: *what is the best strategy to apply rules to sentences?* In Algorithm 6, we have presented the general procedure for rule application, in which a rule is only applied if the resulting reduction complies with a syntactical statistical model previously trained. Indeed, we only applied one rule - the most promising one, based on Function 6.6. However, in many cases there are several rules applicable to a given sentence. Furthermore, we can also have compositional rule applicability. That is, the application of one rule can transform the sentence into another one for which new rules, which were not applicable before, become applicable. With a sufficiently large rule set this poses the problem of finding the best solution in a *search space*.

Finally, we think that our work can be combined with the so called *edmundson paradigm* (Subsection 1.1.1) in which a text summary is created solely by selecting a number of relevant sentences. Combining this extractive approach with ours of sentence reduction can improve the final quality of summary, aiding more naturally the user's required compression level as well as the abstractiveness of the generated summaries.

[1] As those shown on Table 6.3.

Appendix A

Mathematical Logic

The discipline of *Mathematical Logic* is the direct foundation for many other scientific disciplines and methodologies, including *Mathematics*, *Logic Programming* (LP), and *Inductive Logic Programming* (ILP). This last one was a key element used in our work, as shown throughout chapters 5 and 6, where there are some logic concepts and terminations employed. Therefore we have decided to include here some relevant notes from *Mathematical Logic*, namely *Propositional Logic* (PL) and *First Order Logic*(FOL), which, among other things, defines rigorously the notion o deduction, which can be seen as the induction "inverse mechanism".

A.1 Propositional Logic

Propositional Logic (PL) can be seen as a simpler version of *First Order Logic* (FOL), presented in the next section. It gives a synthesized preview for several topics extended in FOL. The main fundamental difference between PL and FOL consists in the absence or presence of *variables*. In PL we do not have variables and the theory is much more simpler. The presence of these entities requires the existence of other elements, in particular the quantifiers. The paradigm of FOL contains a wider range of possibilities that have to be proved. The theoretical frame set of PL is much more simpler. In this section we present a quick overview of PL since some concepts are further developed in the next section of FOL. We do not pretend to have here a complete formalization of the field but present only the key elements and main theoretical results. To dive into a more formally and complete presentation several well known text books on *Mathematical Logical* are available, for example Mendelson (1997) and Enderton (2001).

The whole idea in *Propositional Logic* is to know in which conditions their expressions are *true* or *false*, and how to ensure the validity (true) of new expressions derived from already known valid ones (the concept of *deduction*). PL is a language with syntactical rules, in which expressions are constructed and evaluated into one of their two logical values - *true* or *false*.

The language of PL, $\mathcal{L}_{PL}$, is formed by *parenthesis* "(" and ")", the *negation symbol* "$\neg$" as the unique unary operator, a set of *binary operators* and an infinite set of *propositional symbols* represented here

with uppercase letters, e.g. A, B, P, Q, P_1, P_2, etc. Each *propositional symbol* may assume only two possible values - *true* or *false*. The four principal *binary operators* used in any PL expression are $\wedge$ (conjunction), $\vee$ (disjunction), $\Rightarrow$ (implication) and $\Leftrightarrow$ (equivalence). There are other operators but they can be reduced to a combination of these four. Furthermore, any $\mathcal{L}_{PL}$ expression can be replaced by an equivalent one that uses combinations of only the $\neg$ and the $\wedge$ operators. However, it is usual to work with the aforementioned four operators as they simplify expression readings.

We can construct syntactical correct expressions in $\mathcal{L}_{PL}$, named as **propositions** , by following a small set of procedures, as explained next. First, any *propositional symbol* is syntactically correct in $\mathcal{L}_{PL}$, as well as their negation. Second, any expression constructed from two correct expressions through a *binary operator* result in a correct expression. For example, $P_1 \Rightarrow P_2$ is correct because P_1 and P_2 are correct (propositional symbols). Parenthesis can be found in PL expressions in a similar form as in elementary algebra to change operation priorities. Naturally, for any opening parenthesis there must exist only one corresponding closing parenthesis. Here are some PL correct expressions (propositions):

$$P_1 \Rightarrow (P_2 \wedge P_3) \tag{A.1}$$

$$P_1 \wedge P_2 \Rightarrow P_1 \tag{A.2}$$

$$\neg(P_1 \wedge P_2) \Leftrightarrow (\neg P_1 \vee \neg P_2) \tag{A.3}$$

We name the set of all syntactical correct expressions in $\mathcal{L}_{PL}$, as $\mathcal{S}_{PL}$.

One of the main objectives in logic is to know in which conditions a given proposition is true, also named as *valid*, and also in which conditions a set of formulae gives rise to a new formula, defined as the *consequent*. For example, P_2 is clearly a consequent of $P_1 \wedge P_2$, since whenever the latter is true, so it is the former. This concept can be rigorously defined by using a *true assignment* function, also known as a *valuation function*, which considers all possible (true, false) combinations, for every propositional symbol in the formula and computes the overall *true* value.

Definition 6 (valuation). *A valuation is a function v from the set of all correct expressions $\mathcal{S}_{PL}$ into the $\{true, flase\}$ boolean set, such that for every $\alpha \in \mathcal{S}_{PL}$ and $\beta \in \mathcal{S}_{PL}$ we have:*

- $v(\alpha) \neq v(\neg\alpha)$, *i.e. if one is true the other must be false.*
- $v(\alpha \vee \beta) = true$ *iff* $v(\alpha) = true$ *or* $v(\beta) = true$
- $v(\alpha \wedge \beta) = true$ *iff* $v(\alpha) = true$ *and* $v(\beta) = true$
- $v(\alpha \Rightarrow \beta) = false$ *iff* $v(\alpha) = true$ *and* $v(\beta) = false$
- $v(\alpha \Leftrightarrow \beta) = true$ *iff* $v(\alpha) = v(\beta)$

Table A.1: A true table for the formula $A \Rightarrow (B \wedge C)$, for all possible combination of values assumed by their propositional symbols.

A	B	C	α = "$A \Rightarrow (B \wedge C)$"
true	true	true	true
true	true	false	true
true	false	true	true
true	false	false	true
false	true	true	true
false	true	false	false
false	false	true	false
false	false	false	false

An example of a *true table* for the proposition $A \Rightarrow (B \wedge C)$ is shown in table A.1.

Whenever a proposition is true for all the possible combinations of their propositional symbol values we say that we have a **tautology**. That is, for $\alpha \in \mathcal{S}_{PL}$ we have $v(\alpha) = true$, and it is written as "$\models \alpha$". Formulae A.2 and A.3 are two examples of tautologies.

Now we can define an important issue in PL, which is the meaning of the statement "a set of propositions gives rise to a new proposition". This was already mentioned previously and can be defined as a **logical implication**.

Definition 7 (Logical implication). *A proposition $\alpha \in \mathcal{S}_{PL}$ is said to imply logically another proposition $\beta \in \mathcal{S}_{PL}$, typed as $\alpha \models \beta$, if and only if $\alpha \Rightarrow \beta$ is a tautology: $\models \alpha \Rightarrow \beta$. A set of propositions $\{\alpha_1, ..., \alpha_n\}$ from $\mathcal{S}_{PL}$ logically implies another proposition β if and only if $\models \alpha_1 \wedge ... \wedge \alpha_n \Rightarrow \beta$.*

Another notion related to the previous definition is the notion of **logical equivalence** between two well formed formulae, which holds whenever one implies logically the other one and vice-versa, as stated in definition 8

Definition 8 (Logical equivalence). *Any proposition $\alpha \in \mathcal{S}_{PL}$ is said to be logically equivalent to another proposition $\beta \in \mathcal{S}_{PL}$, typed as $\alpha \equiv \beta$, if and only if $\models \alpha \Rightarrow \beta$ and $\models \beta \Rightarrow \alpha$.*

In PL it is important to know whether a proposition β is implied by a set of other propositions, lets say Σ, according to definition 7. However, assuming that β might be a quite long expression and Σ can contain many propositions, it may be very difficult to prove that $\Sigma \models \beta$. Hopefully, there exist an equivalent way to prove this by using **natural deduction**, which is based on **modus ponens**, presented in section A.2, in A.13 and A.14. As in FOL, in PL there are theoretical results ensuring us that deduction and logical implication are in fact equivalent, though in FOL it involves a more complex theoretical formulation, as it is shown in the following section.

A.2 First Order Logic

The *Inductive Logic Programming* (ILP) learning paradigm, used in our work and presented in section 5.3, is based on *Logic Programming* which in turn is based on *Mathematical Logic*. One big subject in this scientific fundamental discipline, directly related to the ILP theoretical frameset, is *First Order Logic* (FOL). In order to provide some explanations about the ILP operationality and foundation, several necessary FOL terminology and related propositions will be introduced here. We do not pretend to have here a complete and exhaustive FOL theoretical chapter and it is assumed that the reader has already been in contact with this subject. It is included here as a terminological complement for the issues presented in chapters 5 and 6. Good bibliographic material about *Mathematical Logic* can easily be found, two good references for an interested reader are Mendelson (1997) and Enderton (2001).

FOL is an extension of the *propositional calculus*, in which the concept of *variable* and *relationship* is introduced. The $\mathcal{L}$ language alphabet is made of logic **symbols** and **parameters**. In the first class we have *punctuation*, *connectives*, and *variables*. The second one contains *quantifiers*, *constants*, *predicate symbols*, and *functional symbols*. These two last ones are alphanumeric sequences representing the names for predicates and functions. All these elements are synthetically described and exemplified in table A.2.

Table A.2: The $\mathcal{L}$ language fundamental components.

Type	Description	Examples
punctuation	The comma, left and right parenthesis.	"," "(" ")"
connectives	Unary and binary operators for connecting other elements.	$\neg$, $\vee$, $\wedge$, $\Rightarrow$, $\Leftrightarrow$
variables	An alphanumeric sequence representing a range of possible values in the universe.	X, Y, $Velocity$, $Temperature$
quantifiers	The universal and the existential quantifiers.	$\forall$, $\exists$
constants	An alphanumeric sequence representing concrete entities in the universe.	a, b, 7, 23, π, new_york, $john$
predicates	An alphanumeric sequence followed by an n-ary tuple representing a relation.	$less(X,3)$, $prime(7)$, $parent(john, albert)$.
functions	An alphanumeric sequence followed by an n-ary tuple representing a transformation.	$square(3)$, $sin(X)$, $weight(john)$.

The constants are also named as ***atoms***, and some authors consider them as 0-ary functions (Enderton, 2001). Following a *Logic Programming* convention, we use names starting in upper case for representing variables while constants, function, and predicate names always start in lower case.

In FOL the *universe* concept does not refer to any set in particular. It is purely an abstraction, a meta-concept representing any possible universe. An instantiation of the *universe* to a concrete entity is made through an *interpretation* (definition 10). For example, such interpretation may instantiate

the *universe* into the set of all natural numbers ($\mathbb{N}$), or even into a set like "*all the sentences written in English*".

Predicates and functions define respectively relations and transformations for the elements in the *universe*. For example, if our *universe* is interpreted as $\mathbb{N}$, then the shown examples of predicate "$less(X,3)$", and function "$square(3)$" may represent respectively an order relation ($X < 3$) and the arithmetic square function (X^2), in this case applied to the constant 3.

In FOL terms represent the nouns and pronouns of the $\mathcal{L}$ language. A **term** can be either a constant, a variable, or be obtained from these two by applying zero or more times functional composition, like for example: $square(X)$, $maximum(weight(john), weight(mary))$[1].

Finally, another fundamental concept in FOL is an **atomic formula**, which is the most basic well formed expression in the $\mathcal{L}$ language. An atomic formula has the form of $P(t_1, t_2, ..., t_n)$, where "*P*" is an n-ary predicate symbol and for any i, t_i is a term. Examples of atomic formulae are $parent(anne, john)$ and $less(0, square(X))$.

Similarly to what happens in any *human natural language*, a number of syntactical rules must be obeyed to correctly construct valid expressions in $\mathcal{L}$. After presenting the FOL main fundamental entities, we have now all the necessary elements to define the set of well formed expressions, in the $\mathcal{L}$ language.

Definition 9 (*Well Formed Formula* (wff)). *In a FOL language $\mathcal{L}$ a well formed formula (wff) is either an atomic formula or the composition of atomic formulae by applying connectives and quantifiers, i.e. if we know that P and Q are well formed formulae then $\neg P$, $\forall_X P$, $\exists_Y P$, and $P \odot Q$, with $\odot \in \{\vee, \wedge, \Rightarrow, \Leftrightarrow\}$, are also well formed formulae.*

The set of all wffs is designated here as $\mathcal{F}$. A few examples of *wffs* are:

$$less(0,3) \wedge less(3,0) \tag{A.4}$$

$$mother(anne, mary) \tag{A.5}$$

$$\forall_X \exists_Y \; divide(Y,X) \Rightarrow \neg prime(X) \tag{A.6}$$

$$\forall_X \forall_Y \forall_Z \;\; less(X,Y) \wedge less(Y,Z) \Rightarrow less(X,Z) \tag{A.7}$$

In FOL validity and proof is more difficult to verify than in PL, due to a number of more complex entities like variables, terms, and quantifiers. To define the validity of a given formula $\phi \in \mathcal{F}$ we need first to introduce the *interpretation* and *model* concepts. From here on until the rest of this section we present several theoretical key points, to be used for ensuring validation and proof in FOL.

[1] Assuming naturally that $maximum$ and $weight$ are respectively 2-ary and 1-ary functional symbols.

In general the FOL universe and all its fundamental components are considered in a pure abstract form. We designate this meta-universe as $\mathcal{L}_{Universe}$. However, frequently it is important, for illustrative and practical reasons, to specify a particular universe where we are able to give meaning (interpret) to quantifiers, variables, predicates, and functions. Therefore an *interpretation* is a mapping from this abstract meta-universe into a concrete universe. This is what is defined next, in definition 10.

Definition 10 (*Interpretation*). *In FOL an interpretation $\mathcal{I}$ is a mapping from the $\mathcal{L}_{Universe}$ to a particular set $|\mathcal{I}|$ referred simply as "the universe", giving a concrete meaning to any variable, predicate and function, such that:*

- $\mathcal{I}(\forall) = |\mathcal{I}|$
- *For any n-ary predicate P, $P_{\mathcal{I}}$ is an n-ary relation with each component being a member of $|\mathcal{I}|$, i.e we have $P_{\mathcal{I}} \subseteq |\mathcal{I}|^n$.*
- *For any n-ary function f, $\mathcal{I}(f) = f_{\mathcal{I}}$ is an n-ary operation in $|\mathcal{I}|$, i.e we have $f_{\mathcal{I}} : |\mathcal{I}|^n \rightarrow |\mathcal{I}|$*
- *Each constant symbol c will be a member of the universe, i.e $c_{\mathcal{I}} \in |\mathcal{I}|$*

Note that $|\mathcal{I}|$ represents a *set*, possibly infinite and uncountable, and that the assertion $\mathcal{I}(\forall) = |\mathcal{I}|$ specifies the universal quantifier range, meaning that when we have for example "$\forall_X$" in a formula it really means "$\forall_{X \in |\mathcal{I}|}$". In order to illustrate this definition, let us consider the following *wff* about which in general (in the $\mathcal{L}_{Universe}$) we can not say much about it:

$$\forall_X \forall_Y \forall_Z \;\; P(X,Y) \wedge P(Y,Z) \;\Rightarrow\; P(X,Z) \tag{A.8}$$

However, if we have an interpretation $\mathcal{I}$ defined as:

- $\mathcal{I}(\forall) = |\mathcal{I}| = \mathbb{N}$, the set of all natural numbers.
- $P_{\mathcal{I}} =$ "$\leq$", the partial order in $\mathbb{N}$, meaning that for $m \in \mathbb{N}$ and $n \in \mathbb{N}$, the pair $\langle m,n \rangle \in P_{\mathcal{I}}$, also typed as $P_{\mathcal{I}}(m,n)$, iff $m \leq n$.
- Having a function f interpreted as the sucessor function, i.e $f_{\mathcal{I}} = S$ such that $S(n) = n+1$ and $S(0) = 1$.
- Having a constant c interpreted as $c_{\mathcal{I}} = 0$

We can see that the formula A.8 has now a particular clear meaning with this interpretation, representing the transitivity property of the partial order relation ($\leq$) in $\mathbb{N}$. We can say that this formula is true for that interpretation, or that the chosen interpretation satisfies the formula. We could also easily think about another different interpretation that would not satisfy formula A.8. Whenever a formula ϕ is satisfied through an interpretation $\mathcal{I}$ it is said to be a **model** for ϕ, typed as $\models_{\mathcal{I}} \phi$.

In formula A.8 all variables are quantified, however, we may have formulae containing variables that are not under the scope of any quantifier. These are called **free variables**. For example, in formula A.9, V_0 is a free variable.

$$\forall_{V_1} \;\; P(V_0, V_1) \tag{A.9}$$

A *wff* without any free variable is called a **sentence**. An interpretation must also address free variables in formulae. This requires the additional definition of a function that maps all free variables to concrete entities in the interpreted universe, otherwise the formula would remain ambiguous. Thus, being $\mathcal{V}$ the set of all variables we define s as a function mapping $\mathcal{V}$ into the interpreted universe, that is $s : \mathcal{V} \rightarrow |\mathcal{I}|$. For example, if we define $s(V_k) = k$ ($k \in \mathbb{N}$), then we are able to complete the interpretation shown in the previous example, in order to give meaning to formula A.9. It will now state that $\forall_{V_1} \;\; P(s(V_0), V_1)$ or equivalently that $\forall_{V_1} \;\; 0 \leq V_1$ which is true in $\mathbb{N}$. The formula is valid for that interpretation, which includes the definition of s. In this case we type $\models_{\mathcal{I}} \phi \; [s]$. There are further technical details involved in the definition of s that we will ignore here. They may be consulted in Enderton (2001).

We can now present a definition of *logical implication* for FOL formulae, equivalent to definition 7, made for PL.

Definition 11 (Logical implication). *A set of wffs Γ logically implies another wff ϕ, written as $\Gamma \models \phi$, iff for any interpretation $\mathcal{I}$ and any function $s : \mathcal{V} \rightarrow |\mathcal{I}|$, such that $\mathcal{I}$ and s satisfies every member of Γ, they also satisfies ϕ.*

Note that when Γ contains only *sentences* the reference to the s function is not necessary, as stated by the following corollary.

Corollary 1. *A set of sentences Σ implies logically a sentence σ, written as $\Sigma \models \sigma$, iff every model of Σ is also a model of σ. A sentence σ is valid iff it is true in every interpretation and in this case it is written as $\models \sigma$.*

In FOL *validity* is the equivalent concept of *tautology* in PL. Naturally, for a *wff* with free variables saying that $\models \phi$ means un indeed that $\models \phi \; [s]$ for any function s.

Definition 11 ensures us a theoretical method to know whether we have or not logical implication and validity in FOL. However, it would be unpractical to verify it in a given case, since we have to check it for every interpretation! Hopefully logicians from the twentieth century provided us with systematic methods to verify validity and logical implication. In the reminder of this section we present a number of results allowing us to verify $\Sigma \models \sigma$, without having to check every interpretation. One first relevant theorem is the *Compactness theorem* ensuring that we may search for a proof in a set of finite sentences and it will at most be finitely long.

Theorem 4 (Compactness). *We have $\Gamma \models \phi$ if and only if there exists a finite set of formulas $\Gamma_0 \subseteq \Gamma$ such that $\Gamma_0 \models \phi$. In particular a set Σ of sentences has a model iff every finite subset of Σ has a model.*

In FOL we often consider a special kind of *wffs* called ***axioms***, which are assuming to be valid *a priori*, without having to prove them. They represent the kind of knowledge that are widely accepted to be true in a given theoretical field, being the foundation elements upon which a theory is constructed. For example, for the natural numbers there are the *Peano Axioms* from which we show the first three of them, establishing the reflexive, symmetric and transitive properties for the "equal" predicate.

$$\forall_X \; X = X \tag{A.10}$$

$$\forall_X \forall_Y \; X = Y \Rightarrow Y = X \tag{A.11}$$

$$\forall_X \forall_Y \forall_Z \;\; X = Y \wedge Y = Z \;\; \Rightarrow \;\; X = Z \tag{A.12}$$

Being FOL also a theory, it also contains their own axioms, named here as the set of ***logical axioms*** and represented by the letter Λ. In fact we have rather a set with six axiom families, shown in definition 12.

Definition 12 (*Logical axioms*). *In FOL, a wff is a logical axiom if it complies with one of the following cases:*

1. *Any FOL generalization of a PL tautology is an axiom.*
2. $\forall_X \; \varphi \Rightarrow \varphi_t^X$, *where term t is substitutable for variable X in formula* φ.
3. $\forall_X \; (\alpha \Rightarrow \beta) \;\; \Rightarrow \;\; (\forall_X \alpha \Rightarrow \forall_X \beta)$, *where X is a variable and* α *and* β *are wffs*
4. $\varphi \;\; \Rightarrow \;\; \forall_X \varphi$, *where variable* X *does not occur free in formula* φ
5. $X = X$
6. $X = Y \;\; \Rightarrow \;\; (\varphi_X \Rightarrow \varphi_Y)$, *where* φ_X *is an atomic formula and* φ_Y *is obtained from* φ_X *by replacing* X *in zero or more places by* Y.

The majority of these axioms are quite obvious and for the rest of them a more detailed description is made in Enderton (2001). For example, the first axiom states that any FOL formula with the same structure as a propositional logic tautology is a logical axiom. The "same structure" means that it was obtained from a PL formula just by replacing its propositional symbols. For example, formula $P_1 \Rightarrow (P_2 \Rightarrow P_1)$ is a PL tautology, and so the following three formulae are logical axioms in FOL.

- $less(0,3) \;\; \Rightarrow \;\; (less(1,0) \Rightarrow less(0,3))$
- $\exists_X prime(X) \;\; \Rightarrow \;\; (\forall_Y natural(Y) \Rightarrow \exists_X prime(X))$
- $\exists_A creator(A) \Rightarrow (\forall_H \forall_T human(H) \wedge theory(T) \wedge believe(H,T) \Rightarrow \exists_A creator(A))$

These formulae were obtained from the same PL tautology, and besides calling them logical axioms in FOL, we still call them tautologies to.

One of the most hardwired and relevant mechanism in FOL, originally conceived and employed by the classical greek philosophers and rigorously formalized by the twentieth century logicians, is ***deduction***. In general, deduction states that if we know that a given rule is true and we also know that its preconditions (*premises*) are satisfied, then we may conclude its consequent. A classical example is the well known syllogism: "All men are mortal", "Socrates is a man", then "Socrates is mortal". That is, given formulae α and β, if we know that $\alpha \Rightarrow \beta$ and α are true then we conclude that β is also true. This mechanism is also known as ***modus ponens***, the main inference rule used in FOL. In a more schematic form, it can be represented as in proposition A.13.

$$\frac{\alpha \quad \wedge \quad \alpha \Rightarrow \beta}{\therefore \beta} \tag{A.13}$$

or just as:

$$\frac{\alpha, \quad \alpha \Rightarrow \beta}{\beta} \tag{A.14}$$

The following definition defines deduction in a more formal way.

Definition 13. *A deduction of a wff φ from a set of wffs Γ is a sequence $[\phi_0, ..., \phi_{n-1}, \varphi]$ off wffs such that for each $k < n$, we have either:*

- $\phi_k \in \Lambda \cup \Gamma$
- *ϕ_k is obtained by modus ponens from two earlier formulae in the sequence, i.e there are $i < k$ and $j < k$, such that ϕ_i and ϕ_j are in the sequence and ϕ_j is equal to $\phi_i \Rightarrow \phi_k$.*

We say that φ is deduced from Γ, written as $\Gamma \vdash \varphi$ if there exists a deduction of φ from the formulae in $\Lambda \cup \Gamma$. A formula deduced from Γ is called a ***theorem*** of Γ. To exemplify a deduction, suppose that we have:

$$\Gamma = \{\forall_X(\alpha \Rightarrow \beta),\ \neg\alpha \Rightarrow \rho,\ \neg\rho\}$$

with α, β, and ρ being *wffs*, and X a variable that does not occur free in α. Then we are going to show that $\Gamma \vdash \forall_X\beta$, by presenting a deduction using the formulae in Γ, some tautologies, and modus ponens (MP). In each deductive step (each line) the justification is given in the right hand side. So,

we have:

(1)	$(\neg\alpha \Rightarrow \rho) \Rightarrow (\neg\rho \Rightarrow \alpha)$	is a tautology
(2)	$\neg\alpha \Rightarrow \rho$	formula in Γ
(3)	$\neg\rho \Rightarrow \alpha$	MP (1) and (2)
(4)	$\neg\rho$	formula in Γ
(5)	α	MP (3) and (4)
(6)	$\forall_X(\alpha \Rightarrow \beta) \Rightarrow (\forall_X\alpha \Rightarrow \forall_X\beta)$	instance of axiom 3
(7)	$\forall_X(\alpha \Rightarrow \beta)$	formula in Γ
(8)	$\forall_X\alpha \Rightarrow \forall_X\beta$	MP (6) and (7)
(9)	$\alpha \Rightarrow \forall_X\alpha$	instance of axiom 4
(10)	$\forall_X\alpha$	MP (5) and (9)
(11)	$\forall_X\beta$	MP (8) and (10)

therefore we can say that $\Gamma \vdash \forall_X\beta$, or that $\forall_X\beta$ is deducible from Γ, or even that it is a theorem obtained from Γ.

The deduction mechanism is clearly a useful practical method conveying a systematic reasoning search, though not deterministically, in theorem proving. At this point two pertinent theoretical question rises:

1. *Does a theorem from a given set still be logically implied by it, according to definition* 11?
2. *Can a logically implied formula be deduced*?

Note that the deduction mechanism was presented without any reference to a particular interpretation nor any s function. So, one may wonder if a deduced formula still be logically valid, and reversely if any valid or logically implied formula has a deduction.

These two questions have indeed positive answers through two of the most important theoretical results in *Logic*: the *soundness theorem* and well known Gödel's completeness theorem, with which we conclude our theoretical notes on FOL.

Theorem 5 (Soundness). *For a set of wffs Γ and a wff φ, if we have $\Gamma \vdash \varphi$ then $\Gamma \models \varphi$ holds.*

Theorem 6 (Completeness - Gödel, 1930). *For a set of wffs Γ and a wff φ we have:*

(a) *If $\Gamma \models \varphi$ then $\Gamma \vdash \varphi$*

(b) *Any consistent set of formulae is satisfiable.*

From the previous two theorems we know now that $\Gamma \models \varphi$ if and only if $\Gamma \vdash \varphi$, which is a remarkable result. On one hand it says that deductions will lead only to correct conclusions (valid formulae), and on the other hand that a valid formula must have a deduction. This equivalence gives deduction a higher theoretical value, worth to be employed in theorem proving. Now we do not need to worry about interpretations and variable-universe mapping functions to show that $\Gamma \models_{\mathcal{I}} \varphi\ [s]$. It is sufficient to find a deduction of φ from Γ. The fields of *logic programming* and *automatic theorem proving* are mainly based on these two theoretical results.

Appendix B

Relevant Code Fragments

This appendix contains a selection of several code fragments relative to some implementation key aspects. The presentation and description about the *Java* class hierarchy we have implemented was made in subsection 6.2 and the material provided here will complement a bit further that general overview.

```
/**
 * The Sumo-Metric for calculating similarity
 * among sentences.
 *
 * 2006-03-25
 *
 * @param s The other sentence to compare to.
 * @return double The similarity value.
 */
public double dsumo(Sentence s)
{
    if ( s == null )  return 0.0;

    int[] u= this.getCodes();
    int[] v= s.getCodes();
    int n, m;
    if (u.length > v.length) {
        n = v.length; m = u.length;
    }
    else {
        n= u.length; m= v.length;
    }

    //count the number of matches
    int NL= 0;
    for (int i=0; i<u.length; i++)
        for (int j=0; j<v.length; j++)
            if ( v[j] >= 0  &&  u[i] >= 0  &&  Math.abs(u[i]-v[j]) == 0 )  {
                NL++;
                v[j]= -1*v[j];
                u[i]= -1*u[i];
                break;
            }

    //reset link marks
    for (int i=0; i<u.length; i++) if ( u[i] < 0 )  u[i]= -1*u[i];
    for (int i=0; i<v.length; i++) if ( v[i] < 0 )  v[i]= -1*v[i];

    //proceed to final calculations
    double pm= (double)NL/m;
    double pn= (double) NL/n;
    double alfa= 0.5;
    double beta= 1.0 - alfa;

    double similarity= - alfa*log2(pm) - beta*log2(pn);
    if ( similarity > 1.0 )  similarity= 1.0/Math.exp(3*similarity);

    return similarity;
}
```

Figure B.1: The *Sumo-Metric* method for computing sentence similarity.

The first code snippet is about the calculation of the *Sumo* function for two sentences, represented

by the class "Sentence" (see the class diagram in Figure 4.14). Like other implemented sentence similarity measures, it calculates the proximity of two sentences and returns a value in the $[0,1]$ interval. For efficiency reasons the getCodes() method returns the array of word indexes representing the sentence words. Hence, all the calculations are performed by using integer comparisons between two arrays: u and v. Whenever a link is discovered among two positions, lets say i and j their content are switched to negative values in order to avoid any further connections of this already assigned positions. That is exactly what the v[j]=-1*v[j] and u[i]=-1*u[i] instructions make.

An example of a paraphrasic pair alignment is made inside the printAlignments(*,*,*) method, from the GenAlignParaphCorpus class, listed in Figure B.2. We can see the issues of *global*, *local*,

```
/**
 * Compute and print alignments for a paraphrase sentence pair. The output format
 * comply with an XML syntax.
 * @param st One of the paraphrase sentences.
 * @param su The other paraphrase sentence.
 * @param dx The proximity value under which parphrases were selected. To be printed.
 */
private void printAlignments(Sentence st, Sentence su, double dx) {
    double px = -1.0;
    if (ALIGNTYPE == DYNAMIC) {
        px = Sentence.countNormIntersectLinks(su, st);
    }

    if (ALIGNTYPE == SWATERMAN || (ALIGNTYPE == DYNAMIC && px > 0.4)) {
        SWaterman sw = new SWaterman(st.getCodes(), su.getCodes(), dictionary);
        Vector<String[]> vs = sw.getParaAlignsHoriz();
        if (vs == null || vs.size() < 1) {
            return;
        }
        outfile.printf(Locale.US, "   <paraph id=\"%d\" alg=\"SW\" prc=\"%f\">\n", ++IDPAIR, dx);
        outfile.printf(Locale.US, "      <nx>(%d, %d, %f)</nx>\n", st.cod, su.cod, dx);
        outfile.printf("      <s1>%s</s1>\n", st);
        outfile.printf("      <s2>%s</s2>\n", su);
        for (int k = 0; k < vs.size(); k++) {
            String[] as = vs.get(k);
            outfile.printf("      <subalgn id=\"%d\">\n", k + 1);
            outfile.printf("         <sa1>%s</sa1>\n", as[0]);
            outfile.printf("         <sa2>%s</sa2>\n", as[1]);
            outfile.print( "      </subalgn>\n");
        }
        outfile.printf("   </paraph>\n\n");
    } else { //---> NWunsh - Global Alignment.
        NWunsch nw = new NWunsch(st.getCodes(), su.getCodes());
        nw.setDic(dictionary);
        nw.buildMatrix();
        String[] as = nw.getAlignmentH();
        outfile.printf("   <paraph id=\"%d\" alg=\"NW\">\n", ++IDPAIR);
        outfile.printf(Locale.US, "      <nx>(%d, %d, %f)</nx>\n", st.cod, su.cod, dx);
        outfile.printf("      <s1>%s</s1>\n", st);
        outfile.printf("      <s2>%s</s2>\n", su);
        outfile.printf("      <sa1>%s</sa1>\n", as[0]);
        outfile.printf("      <sa2>%s</sa2>\n", as[1]);
        outfile.printf("   </paraph>\n");
    }
}
```

Figure B.2: The "printAlignments" method for computing and printing paraphrase sentence alignments to an XML file.

and *dynamic* alignment being involved here to choose the best algorithm for aligning a given sentence pair, indicated by the first two parameters ("st" and "su").

If the ALYGNTYPE is DYNAMIC, the number of *crossings* between two sentences, as discussed in Subsection 4.2, and in particular illustrated in Figure 4.7, is calculated and based on this result the decision for one algorithm is made, as presented in Algorithm 2. The "SWaterman" and "NWunsch" classes hold

the data items and the processing methods to compute the alignment algorithms, respectively the *Smith-Waterman* and the *Needleman-Wunsch*, as suggested by the corresponding class names.

The implementation of the *crossings* calculation, as described in Subsection 4.2, is shown in the code listing from Figure B.3.

```
/**
 * Counts te number of link intersections existing between the two sentences.
 * <p>Purpose: help in the choosing of the alignment algorithm:
 * SWaterman or NWunsch</p>
 * <p>Date: 2007/03/23</p>
 * @param s1 Sentence
 * @param s2 Sentence
 * @return int
 */
public static int countIntersectLinks(Sentence sa, Sentence sb) {
    if ( sa == null || sb == null || sa.length() < 1 || sb.length() < 1 ) return -1;

    int[] va= sa.getCodes();
    int[] vb= sb.getCodes();

    int[][] A= readLinks(va, vb);
    if ( A == null) {
        Text t= new Text();
        t.add(sa);
        t.add(sb);
        t.codify();
        va= t.get(0).getCodes();
        vb= t.get(1).getCodes();
        A= readLinks(va, vb);
    }
    if ( A == null )  return -101;

    int counter= 0;
    for (int i=0; i<A[0].length-1; i++) {
        for (int j = i+1; j < A[0].length; j++) {
            //compare ai =<x,a> with aj=<y,b>
            int x= A[0][i], y= A[0][j];
            int a= A[1][i], b= A[1][j];
            if ( (x-y)*(a-b) < 0 ) {
                counter++;
            }
        }
    }
    return counter;
}
```

Figure B.3: Count the relative number of *crossings* between two sentences. Method from the *Sentence* class and related to the previous listing.

Appendix C

Relevant *Aleph* Elements

This appendix contains several *Aleph* data elements, some illustrative and some employed in our research, both referred in the thesis. The first listing contains the background knowledge file for the *good graph* example, presented in Section 5.4, and directly related with Figure 5.5.

```
:- set(depth, 100).
:- set(clauselength,10).


:- modeh(1, goodgraph(+gname)).

:- modeb(1, numvertex(+gname, -int)).
:- modeb(1, numedges(+gname, -int)).
:- modeb(1, maxdegree(+gname, -int)).
:- modeb(*, diff(+int, +int, #int)).
:- modeb(*, odd(+int)).
:- modeb(*, even(+int)).
:- modeb(*, prime(+int)).


:- determination(goodgraph/1, numvertex/2).
:- determination(goodgraph/1, numedges/2).
:- determination(goodgraph/1, maxdegree/2).
:- determination(goodgraph/1, diff/3).
:- determination(goodgraph/1, odd/1).
:- determination(goodgraph/1, even/1).
:- determination(goodgraph/1, prime/1).


%-------------------------------------------
% TYPES
%-------------------------------------------
gname(a).   gname(b).   gname(c).  gname(d).
gname(e).   gname(f).   gname(g).  gname(h).
gname(i).   gname(j).   gname(k).  gname(l).
%-----------------------------------------------------
% ALL 12 GRAPHS
%-----------------------------------------------------
% POSITIVES %
graph(a, [1-2, 2-3, 3-1]).
graph(c, [1-2, 1-3, 1-4, 1-5, 4-5, 5-3, 3-2]).
graph(e, [1-2, 1-7, 1-6, 1-5, 7-6, 4-5, 4-2, 2-3]).
graph(g, [1-2, 2-3, 1-5, 5-4]).
graph(i, [1-4, 1-5, 5-4, 4-2, 4-3, 2-3]).
graph(k, [1-2, 3-2, 1-9, 1-8, 1-6, 6-7, 1-5, 1-4]).

% NEGATIVES %
graph(b, [1-3, 1-4, 4-3, 3-2, 3-5, 5-6, 2-6]).
graph(d, [1-3, 3-2, 2-4, 4-1]).
graph(f, [1-5, 1-6, 1-2, 1-3, 4-5]).
graph(h, [1-2, 2-4, 4-3, 4-7, 4-5, 4-6, 5-8, 5-9]).
graph(j, [1-2, 1-3, 1-4, 3-5, 4-5]).
graph(l, [1-2, 2-3, 3-5, 4-1, 4-5, 4-6, 6-5, 6-7, 7-1]).
%-----------------------------------------------------


%-----------------------------------------------------
% ARITHEMETIC DEFINITIONS.
%-----------------------------------------------------
odd(N) :- N mod 2 =:= 1.

even(N) :- N mod 2 =:= 0.
```

```

diff(X,Y,YX) :- YX is Y-X.


%-----------------------------------------------------
% LIST OF ALL DISTINCT VERTICES, FROM A GRAPH
%-----------------------------------------------------
vertices(ID, LVert) :-
    graph(ID, G),
    vertices(G, [], LVert)
    .

vertices([], LVert, LVert).
vertices([X-Y|R], V, LVert) :-
    union([X,Y], V, VXY),
    vertices(R, VXY, LVert)
    .


%-----------------------------------------------------
% NUMBER OF VERTEXES IN A GRAPH
%-----------------------------------------------------
numvertex(ID, N) :-
    vertices(ID, V),
    length(V, N)
    .


%-----------------------------------------------------
% NUMBER OF EDGES IN A GRAPH
%-----------------------------------------------------
numedges(ID, N) :-
    graph(ID, G),
    length(G, N)
    .
%-----------------------------------------------------
% MAXIMUM VERTEX DEGREE FOR A GIVEN GRAPH
%-----------------------------------------------------
maxdegree(ID, Max) :-
    graph(ID, G),
    vertices(ID, V),
    maxdegree(V, G, 0, Max)
    .

maxdegree([], _, M, M).
maxdegree([A|R], G, M, Max) :-
    degree(A, G, N),
    (N > M ->
        maxdegree(R, G, N, Max)
     ;
        maxdegree(R, G, M, Max)
    )
    .


%-----------------------------------------------------
% DEGREE OF A VERTEX.
%-----------------------------------------------------
degree(A, G, N) :- degree(A, G, 0, N), !.

degree(_, [], N, N).
degree(A, [X-Y|R], K, N) :-
    (X == A ; Y == A),
    K1 is K+1,
    degree(A, R, K1, N)
    .
degree(A, [_|R], K, N) :-
    degree(A, R, K, N)
    .


%-----------------------------------------------------
% REUNION OF TWO LISTS.
%-----------------------------------------------------
union([], Set, Set) :- !.
union([X|R], Set, USet) :-
    orput(X, Set, SetX),
    union(R, SetX, USet).


orput(X, [X|R], [X|R]) :- !.
orput(X, [Y|R], [Y|RX]) :-
    X \== Y,
    !,
```

```
142         orput(X, R, RX)
143         .
144
145     orput(X, [], [X]).
146
147
148
149     %-----------------------------------------------------
150     % TEST IF A GIVEN INTEGER IS PRIME.
151     %-----------------------------------------------------
152     prime(1).
153     prime(2).
154     prime(N) :-
155         N > 2,
156         divisor(K,N,_),
157         K == 1
158         .
159
160
161     %-----------------------------------------------------
162     % K DIVIDES N GIVING Q AS THE QUOTIENT: N = K*Q.
163     %-----------------------------------------------------
164     divisor(K,N,Q) :-
165         N>0,
166         SN is sqrt(N),
167         divisor__(K, 2, N:SN, Q)
168         .
169
170     divisor__(1, Kx, N:SN, N) :-
171         Kx > SN,
172         !
173         .
174     divisor__(K, K, N:_, Q) :-
175         N mod K =:= 0,
176         Q is N//K,
177         !
178         .
179     divisor__(K, Kx, N:SN, Q ) :-
180         Kx < SN+1,
181         Kx1 is Kx+1,
182         divisor__(K, Kx1, N:SN, Q)
183         .
```

Listing C.1: The *good graph* background knowledge file.

The next listing contains the template file used to generate *Aleph*'s *background knowledge* file for a given set of examples (*bubbles*).

```
1     %
2     %-------------------------------------------------------------------------------
3     % AUTOMATICALLY GENERATED BY:... "$PROG_NAME"
4     % INPUT FILE:................... "$INPUT_FILE"
5     % MOMENT:....................... $MOMENT
6     %-------------------------------------------------------------------------------
7     %
8     %
9     % ALEPH PARAMETERS
10    :- set(evalfn,user).
11    :- set(minpos, 5).
12    :- set(verbosity, 0).
13
14
15    %-------------------------------------------------------------------------------
16    % MODE DECLARATIONS
17    %-------------------------------------------------------------------------------
18    :- modeh(1, rule(+bub)).
19
20    :- modeb(1, transfdim(+bub, n(#nat,#nat))).
21    :- modeb(3, chunk(+bub, #side, #chk)).
22    :- modeb(*, inx(+bub, #side, #k, #tword)).
23
24    :- determination(rule/1, transfdim/2).
25    :- determination(rule/1, chunk/3).
26    :- determination(rule/1, inx/4).
27
28
29    %-------------------------------------------------------------------------------
30    % YAP FILE
31    %-------------------------------------------------------------------------------
```

```
32  $YAP_FILE
33
34
35  %-----------------------------------------------------------------------------
36  % DATA FILE
37  %-----------------------------------------------------------------------------
38  numero_instancias($DATA_SIZE).
39
40  $DATA_FILE
41
42
43  %-----------------------------------------------------------------------------
44  % COSTS | CONSTRAINTS | REFINEMENT
45  %-----------------------------------------------------------------------------
46
47  cost(_, [P,_,L], Cost) :-
48      value_num_literals(L, ValueL),
49      Cost is P/$DATA_SIZE * ValueL
50      .
51
52  value_num_literals(1, 0.10). %     |
53  value_num_literals(2, 0.25). % 1.0 -                   _
54  value_num_literals(3, 0.50). %     |                   _     _
55  value_num_literals(4, 1.00). %     |               _   _     _     .
56  value_num_literals(5, 0.60). %     |         _     _   _     _     _     .
57  value_num_literals(6, 0.40). %     |   _     _     _   _     _     _     _
58  value_num_literals(7, 0.20). %   ------------------------------------------->
59  value_num_literals(_, 0.00). %         1     2     3     4     5     6     7
60
61
62  false :-
63      hypothesis(rule(_), true, _)
64      .
65
66  false :-
67      hypothesis(rule(_), Body, _),
68      num_literais(Body, N),
69      N < 2
70      .
71
72
73  num_literais(Body, N) :- num_literais(Body, 0, N).
74
75  num_literais((_,Resto), K, N) :-
76      K1 is K+1,
77      num_literais(Resto, K1, N),
78      !
79      .
80
81  num_literais(_, K, N) :-
82      N is K+1
83      .
84
85
86  /**/
87  false :-
88      hypothesis(rule(_), Body, _),
89      contar_restr_zonas(Body, NL, NX, NY, NR),
90      nao_valido(NL, NX, NY, NR)
91      .
92  /**/
93  /**/
94  nao_valido( 0,  0,  0,  0).  %----> nenhuma zona restrita.
95  nao_valido( _,  0,  _,  _).  %----> só contextos laterais definidos.
96  nao_valido( 0,  _,  _,  _).  %----> left sem restrições.
97  nao_valido( _,  _,  _,  0).  %----> right sem restrições.
98  /**/
99
100
101 contar_restr_zonas(Body, NL, NX, NY, NR) :-
102     contar_restr_zonas__(Body, 0:NL, 0:NX, 0:NY, 0:NR)
103     .
104
105 contar_restr_zonas__((G,GR), KL:NL, KX:NX, KY:NY, KR:NR) :-
106     arg(2, G, Zona),
107     incrementa_zona(Zona, KL:KL1, KX:KX1, KY:KY1, KR:KR1),
108     contar_restr_zonas__(GR, KL1:NL, KX1:NX, KY1:NY, KR1:NR),
109     !
110     .
111 contar_restr_zonas__(G, KL:NL, KX:NX, KY:NY, KR:NR) :-
112     arg(2, G, Zona),
113     incrementa_zona(Zona, KL:NL, KX:NX, KY:NY, KR:NR)
114     .
115
```

```
116
117   incrementa_zona(left,      KL:KL1, KX:KX,  KY:KY,  KR:KR ) :- KL1 is KL+1, !.
118   incrementa_zona(center:x,  KL:KL,  KX:KX1, KY:KY,  KR:KR ) :- KX1 is KX+1, !.
119   incrementa_zona(center:y,  KL:KL,  KX:KX,  KY:KY1, KR:KR ) :- KY1 is KY+1, !.
120   incrementa_zona(right,     KL:KL,  KX:KX,  KY:KY,  KR:KR1) :- KR1 is KR+1, !.
121   incrementa_zona(    _,     KL:KL,  KX:KX,  KY:KY,  KR:KR).
122
123
124   false :-
125       hypothesis(rule(_), Body, _),
126       contem(chunk(_,center:x,vp), Body)
127       .
```

Listing C.2: Our template for background knowledge generation - file "bub-b.lgt"

This template file from Listing C.2 is used by a *Java* program to generate the final *background knowledge* file for *Aleph*. Therefore some *Prolog* extraneous elements are contained in it, specially meta-variables which start with the "$" character. These are meant to be replaced by concrete elements, during a given *background knowledge* file generation. For example: `$PROG_NAME`, `$INPUT_FILE`, and `$MOMENT`, in lines 3, 4, and 5. The `$DATA_FILE` meta-variable will be replaced by a *Prolog* directive to load the file containing the set of training examples - the codified bubbles. Similarly the `$YAP_FILE` will include a file containing common *Prolog* utilities, employed in the "`*.b`" file being generated, and presented in Listing C.3.

```
1   %-----------------------------------------------------------------------------
2   % IMPORTS & AUXILIAR DEFINITIONS.
3   %-----------------------------------------------------------------------------
4   import(File) :-
5       atom_concat('/lib/prolog/', File, PathFile),
6       reconsult(PathFile)
7       .
8
9
10  :- import('utils.lgt').
11
12  bub(X) :- entre(X, 0, $DATA_SIZE).
13
14
15  :- import('penpostags.lgt').
16
17  tag(X) :- penpost(_, X, _).
18
19
20  %-----------------------------------------------------------------------------
21  % TYPE DEFINITIONS
22  %-----------------------------------------------------------------------------
23  side(left).
24  side(right).
25  side(center:x).
26  side(center:y).
27
28
29  k(1).  k(2).   k(3).
30
31  chk(np).    chk(undefined).     chk(np).    chk(vp).
32  chk(pp).    chk(prt).           chk(advp).  chk(multi).
33
34
35  nat(X) :- entre(X,0,20).
36
37
38  %-----------------------------------------------------------------------------
39  % DOMAIN KNOWLEDGE
40  %-----------------------------------------------------------------------------
41  :- op(250, xfx, --->).
42
43  % DIMENSIONS
44  dimension(ID, left, Ln) :-  bub(ID, t(_,0), L, _--->_, _), length(L, Ln).
45  dimension(ID, right, Rn) :- bub(ID, t(_,0), _, _--->_, R), length(R, Rn).
46  dimension(ID, center:x, Xn) :- bub(ID, t(_,0), _, X--->_, _), length(X, Xn).
```

```
47  dimension(ID, center:y, Yn) :- bub(ID, t(_,0), _, _--->Y, _), length(Y, Yn).
48
49  dimLeft(ID, Ln) :- dimension(ID, left, Ln).
50  dimRight(ID, Rn) :- dimension(ID, right, Rn).
51
52
53  %-----------------------------------------------------------------------------
54  % Definitions of a dimensional transformation
55  %-----------------------------------------------------------------------------
56  transfdim(ID, n(Xn,Yn)) :-
57      dimension(ID, center:x, Xn),
58      dimension(ID, center:y, Yn)
59      .
60
61
62  %-----------------------------------------------------------------------------
63  % In lexicon-syntactic context
64  %-----------------------------------------------------------------------------
65  inx(ID, Side, K, TWord) :-
66      side(Side),
67      context(ID, Side, Context),
68      k(K),
69      inKpos(Context, K, TWord)
70      .
71
72  inKpos(Context, K, Word) :- inKLEX(Word, K, Context).
73  inKpos(Context, K, pos(Tag)) :- inKPOS(Tag, K, Context).
74
75  context(ID, left,  L) :- bub(ID, t(_,0), L, _--->_, _).
76  context(ID, right, R) :- bub(ID, t(_,0), _, _--->_, R).
77  context(ID, center:x, X) :- bub(ID, t(_,0), _, X--->_, _).
78  %context(center:y, Y) :- bub(ID, _, _--->Y, _).
79
80
81  %-----------------------------------------------------------------------------
82  % Lexical scan
83  %-----------------------------------------------------------------------------
84  inLEX(ID, center:x, Word) :- bub(ID, t(_,0), _, X--->_, _),  inLEX(Word, X).
85  inLEX(ID, left,   Word) :-   bub(ID, t(_,0), Context, _, _), inLEX(Word, Context).
86  inLEX(ID, right,  Word) :-   bub(ID, t(_,0), _, _, Context), inLEX(Word, Context).
87  inLEX(ID, center:y, Word) :- bub(ID, t(_,0), _, _--->Y, _),  inLEX(Word, Y).
88
89  inLEX(Word, [Word/_/_|_]).
90  inLEX(Word, [_|Tail]) :- inLEX(Word, Tail).
91
92
93  %-----------------------------------------------------------------------------
94  % Positional lexical scan
95  %-----------------------------------------------------------------------------
96  inKLEX(ID, left,  K,  Word) :-
97      bub(ID, t(_,0), Context, _, _),
98      k(K),
99      inKLEX(Word, K,  Context)
100     .
101 inKLEX(ID, right, K,  Word) :-
102     bub(ID, t(_,0), _, _, Context),
103     k(K),
104     inKLEX(Word, K,  Context)
105     .
106
107 inKLEX(ID, center:x, K,  Word) :-
108     bub(ID, t(_,0), _, CX--->_, _),
109     k(K),
110     inKLEX(Word, K,  CX)
111     .
112
113 inKLEX(ID, center:y, K,  Word) :-
114     bub(ID, t(_,0), _, _--->CY, CY),
115     k(K),
116     inKLEX(Word, K,  CY)
117     .
118
119 inKLEX(LEX, 1, [LEX/_/_|_]).
120 inKLEX(LEX, K, [_|Tail]) :-
121     K > 1,
122     K1 is K-1,
123     inKLEX(LEX, K1, Tail)
124     .
125
126
127 %-----------------------------------------------------------------------------
128 % Syntactical scan
129 %-----------------------------------------------------------------------------
130 inPOS(ID, center:x, Tag) :- bub(ID, t(_,0), _, X--->_, _),  inPOS(Tag, X).
```

```
131   inPOS(ID, left,   Tag) :-   bub(ID, t(_,0), Context, _, _), inPOS(Tag, Context).
132   inPOS(ID, right,  Tag) :-   bub(ID, t(_,0), _, _, Context), inPOS(Tag, Context).
133   inPOS(ID, center:y, Tag) :- bub(ID, t(_,0), _, _--->Y, _),  inPOS(Tag, Y).
134
135   inPOS(POS, [_/POS/_|_]).
136   inPOS(POS, [_|Tail]) :- inPOS(POS, Tail).
137
138
139   %-----------------------------------------------------------------------------
140   % Positional syntactical scan
141   %-----------------------------------------------------------------------------
142   inKPOS(ID, left,  K,  Tag) :-
143       bub(ID, t(_,0), Context, _, _),
144       k(K),
145       inKPOS(Tag, K,  Context)
146       .
147   inKPOS(ID, right, K,  Tag) :-
148       bub(ID, t(_,0), _, _, Context),
149       k(K),
150       inKPOS(Tag, K,  Context)
151       .
152
153   inKPOS(ID, center:x, K,  Tag) :-
154       bub(ID, t(_,0), _, CX--->_, _),
155       k(K),
156       inKPOS(Tag, K,  CX)
157       .
158
159   inKPOS(ID, center:y, K,  Tag) :-
160       bub(ID, t(_,0), _, _--->CY, CY),
161       k(K),
162       inKPOS(Tag, K,  CY)
163       .
164
165   inKPOS(POS, 1, [_/POS/_|_]).
166   inKPOS(POS, K, [_|Tail]) :-
167       K > 1,
168       K1 is K-1,
169       inKPOS(POS, K1, Tail)
170       .
171
172
173   %-----------------------------------------------------------------------------
174   % Chunking scan
175   %-----------------------------------------------------------------------------
176   chunk(ID, left, TAG) :-
177       bub(ID, t(_,0), Left, _, _),
178       inKCHK(TAG, 1, Left)
179       .
180
181   chunk(ID, right, TAG) :-
182       bub(ID, t(_,0), _, _, Right),
183       inKCHK(TAG, 1, Right)
184       .
185
186   chunk(ID, center:x, TAG) :-
187       bub(ID, t(_,0), _, CX--->[], _),
188       struct_chunk(CX, TAG)
189       .
190
191
192   %-----------------------------------------------------------------------------
193   % Positional chunking scan
194   %-----------------------------------------------------------------------------
195   inKCHK(TAG, 1, [_/_/TAG|_]).
196   inKCHK(TAG, K, [_|Tail]) :-
197       K>1,
198       K1 is K-1,
199       inKCHK(TAG, K1, Tail)
200       .
201
202
203   %-----------------------------------------------------------------------------
204   % Structural chunks in a sequence.
205   %-----------------------------------------------------------------------------
206   struct_chunk([_/_/CHK|Tail], Tag) :- struct_chunk(Tail, CHK, Tag).
207
208   struct_chunk([], CHK, CHK) :- !.
209   struct_chunk([_/_/CHK|Tail], CHK, Tag) :- struct_chunk(Tail, CHK, Tag), !.
210   struct_chunk([_/_/_|_], _, multi).
```

Listing C.3: File with common *Prolog* utilities to be included in any generated *background knowledge* file.

The "utils.lgt" file, imported in the previous listing's line 10, is another *Prolog* file with more general utilities and it can be seen next, in our last listing.

```
% ------------------------------------------------------------------------------
%  General Utilities
%
%  JPC, since May 2005
% ------------------------------------------------------------------------------


% ------------------------------------------------------------------------------
%  Write a line with N characters indicated in CH.
% ------------------------------------------------------------------------------
linha(N, CH) :-
    linha(user_output, N, CH)
    .


linha(Stream, N, CH) :-
    N > 0,
    linha_(N, CH, Lista:[]),
    atom_concat(Lista,Linha),
    write(Stream, Linha)
    .


linha_(0, _, L:L).
linha_(N, C, L:LAcc) :-
    N > 0,
    N1 is N-1,
    linha_(N1, C, L:[C|LAcc])
    .


% ------------------------------------------------------------------------------
%  Defines the membership of a literal, in a list of literals:
%  example:
%          ?- contem(p(X), (q(t), p(a), s(u))).
%          ?- X = a
% ------------------------------------------------------------------------------
contem(Literal, (Literal,_)) :- !.
contem(Literal, ( _, Resto)) :- contem(Literal, Resto), !.
contem(Literal, Literal).


% ------------------------------------------------------------------------------
%  Operating system utilities.
% ------------------------------------------------------------------------------
:- op(100, fx, $).
$(Command) :- system(Command).

:- op(95, fx, say).
say(Message) :-
    atom_concat('say ', Message, Command),
    $(Command)
    .


% ------------------------------------------------------------------------------
%  Left padding - example:
%  ?- lpad(75, 0, 5, A).
%  A = '00075'
%
%  ?- lpada(ubi, '|', 5, A).
%  A = '||ubi'
% ------------------------------------------------------------------------------
lpad(Atom, CH, M, CHMAtom) :-
    ( number(Atom) ->
        number_chars(Atom, AC)
     ;
        atom_chars(Atom, AC)
    ),
    length(AC, N),
    ( N >= M ->
        CHMAtom = Atom
     ;
        DMN is M-N,
        list_expand(CH, DMN, List),
        atomic_concat(List, CHM),
        atomic_concat([CHM, Atom], CHMAtom)
    )
    .
```

```


list_expand(_, 0, []).
list_expand(X, N, [X|Tail]) :-
    N > 0,
    N1 is N-1,
    list_expand(X, N1, Tail)
    .


% ------------------------------------------------------------------------------
%  Reverse a list of elements.
% ------------------------------------------------------------------------------
reverse(List, RList) :- reverse(List, [], RList).

reverse([], LACC, LACC).
reverse([X|R], LACC, RList) :- reverse(R, [X|LACC], RList).


% ------------------------------------------------------------------------------
%  Outputs all list elements. The second argument is the separator, which  is
%  meant to be written between any two list elements.
% ------------------------------------------------------------------------------
output([], nl) :- nl, !.
output([], _) :- !.
output([X|R], nl) :-
    write(X), nl,
    output(R, nl),
    !
    .
output([X|R], Separator) :-
    write(X), (Separator \== void -> write(Separator) ; true),
    output(R, Separator)
    .

output(Lista) :- output(Lista, void).


% ------------------------------------------------------------------------------
%  Test if X is a number satisfying A =< X =<B. If X is a variable and A, and B
%  are numbers in entre/2, then X will be instantiated with each element between
%  A, and B, in H steps, i.e: X = A, A+H, A+2H, ..., until A+nH > B
% ------------------------------------------------------------------------------
entre(X,A,B) :- entre(X,[A,B]:1).

entre(X,[A,B]) :- entre(X,[A,B]:1).

entre(X,[A,B]:_) :-        number(X), !, A=<X, X=<B.
entre(A,[A,B]:_) :- A=<B.
entre(X,[A,B]:H) :-
        AH is A+H,
        AH =< B,
        entre(X,[AH,B]:H).


% ------------------------------------------------------------------------------
%  Instantiates a list of variables with numbers ranging in the [A,B] interval,
%  with H increments.
% ------------------------------------------------------------------------------
instanciar([],_).
instanciar([X|R], [A,B]:H) :- entre(X,[A,B]:H), instanciar(R, [A,B]:H).


% ------------------------------------------------------------------------------
%  Membership test
% ------------------------------------------------------------------------------
member(X, [X|_]).
member(X, [_|R]) :- member(X, R).
```

Listing C.4: The "utils.lgt" file with general *Prolog* utilities, used and inported in the file from listing C.3.

Appendix D

System Execution Example

Previously, on Section 6.2, we have already presented the key issues related to the execution of each one of our system modules. This appendix is a kind of complement for that section, where not only the complete set of execution procedures are shown but also their necessary pre-configurations to make it possible, like for example the *operating system* settings and third part necessary modules. Therefore we will focus on a concrete example, comprehending a set of *web news stories* collected during 15 days. All the necessary files, for settling this execution example in a machine, may be downloaded from the following address:

http://www.di.ubi.pt/∼jpaulo/competence/ [November 2010]

These 15 days of *news stories* corresponds to 15 files, one per day, obtained by running our *news* extractor program "`GoogleNewsSpider.java`" in a *POSIX* machine. A small *Bash* script launched daily by the *Unix* `cron` tool, start the *news* extractor, as shown in Listing D.1.

```
1  #!/bin/bash
2  #
3  # JPC, SEP 2008
4  #
5  FILE=$(date "+n%Y%m%d-%OHh.xml")
6  CP='/home/jpaulo/bin/HultigJPCLib.jar:/home/jpaulo/bin/GNewsSpider.jar'
7  java -cp $CP GoogleNewsSpider > /dev/null
```

Listing D.1: Google News Spider, launched by a shell script.

Note that the generated file name will be relative to the moment of creation, including year, month, day and hour, as specified in line five of listing D.1.

The subsequent programs for *paraphrase extraction*, *alignment* and *bubble selection* is also performed by using a shell script, as shown in listing D.2.

```
1  #!/bin/bash
2
3  # GENERATE A CORPUS OF ALIGNED PARAPHRASES, AUTOMATICALY EXTRACTED FROM
4  # A FOLDER WITH WEB NEWS STORIES FILES.
5  java -Xms32m -Xmx1024m GenAlignParaphCorpus -dir "./fnews" -out pac.xml
6
```

```
7   # FROM A CORPUS OF ALIGNED PARAPHRASES, EXTRACTS A LIST OF BUBBLES, WHICH
8   # IS SAVED IN A BYNARY FILE.
9   java -Xms32m -Xmx1024m ExtractBubbles -inp pac.xml -fdata lbub.dat mute
10
11  # GENERATE ALEPH DATASET, FROM BINARY "ListXBubble" FILES, IN THE
12  # CURRENT DIRECTORY, TAKING INTO ACCOUNT THE DEFINED ALEPH TEMPLATE
13  # FILES bub-b.lgt and bub-yap.lgt, STORED IN THE /lib/prolog FOLDER.
14  java hultig.sumo.ListXBubble -aleph dir
```

Listing D.2: A script launching various system modules.

The commented lines[1] in this script describe briefly each corresponding command. The input folder for the first program (line 5) is naturally the "`fnews`" folder, contained in the current execution folder. Here the output file is "`pac.xml`", which is used as the input for the second program execution concerning *bubble selection* (line 9). This execution produces the `lbub.dat` binary file which in turn is used as input in the third execution command (line 14). In this "`ListXBubble`" execution the "`dir`" parameter specifies that the current folder will be searched for input files, since more than one *bubble* binary file, coming from different origins, can be loaded. In such cases repeated learning instances are removed from the final list.

As a result of executing the script from Listing D.2, in a folder containing only the "`fnews`" data folder, six files are generated, where besides "`pac.xml`" and "`lbub.dat`" already mentioned, we also obtain "`lxbub.b`", "`lxbub.f`", "`lxbub.lgt`", and "`lxbub.yap`", which are the *Prolog* and *Aleph* files that will be used in the induction process. To start the induction process an ISO *Prolog* interpreter must be installed[2] and also the *Aleph* system (Srinivasan, 2000), which in essence is also a program written in *Prolog*. After having these elements installed, one should first start the *Prolog* interpreter and the *Aleph* system. Then it is possible to load the learning data files for *Aleph* and start induction.

```
1   [john@jpcmacbook#0 2010-07-30 23:00:05] /a/news@google/test14days (85.197 Mb)
2   $ ls
3   fnews          lxbub.b         lxbub.lgt        pac.xml
4   lbub.dat       lxbub.f         lxbub.yap        rset20100723233854.txt
5
6   [john@jpcmacbook#0 2010-07-30 23:00:11] /a/news@google/test14days (85.197 Mb)
7   $ yap
8   % Restoring file /usr/local/lib/Yap/startup
9   YAP version Yap-5.1.2
10     ?- ['~/bin/aleph.yap'].
11   % consulting /Users/john/bin/aleph.yap...
12
13
14  A L E P H
15  Version 5
16  Last modified: Sun Jun  4 10:51:31 UTC 2006
17
18  Manual: http://www.comlab.ox.ac.uk/oucl/groups/machlearn/Aleph/index.html
19
20   % consulted /Users/john/bin/aleph.yap in module user, 99 msec 1126832 bytes
21  yes
22     ?- read_all(lxbub).
23   % reconsulting /a/news@google/test14days/lxbub.b...
```

[1] Those starting with the "#" character.

[2] Preferably the "Yap Prolog" (Santos Costa *et al.*, 2002), for efficiency reasons.

```
24     % reconsulting /a/news@google/test14days/lxbub.yap...
25      % reconsulting /lib/prolog/utils.lgt...
26      % reconsulted /lib/prolog/utils.lgt in module user, 2 msec 10176 bytes
27      % reconsulting /lib/prolog/penpostags.lgt...
28      % reconsulted /lib/prolog/penpostags.lgt in module user, 0 msec 4400 bytes
29     % reconsulted /a/news@google/test14days/lxbub.yap in module user, 5 msec 35136 bytes
30     % reconsulting /a/news@google/test14days/lxbub.lgt...
31     % reconsulted /a/news@google/test14days/lxbub.lgt in module user, 2862 msec 27844696 bytes
32     % reconsulting /usr/local/share/Yap/system.yap...
33      % reconsulting /usr/local/share/Yap/lists.yap...
34      % reconsulted /usr/local/share/Yap/lists.yap in module lists, 3 msec 22768 bytes
35     % reconsulted /usr/local/share/Yap/system.yap in module system, 9 msec 82792 bytes
36
37
38   DATA SET SIZE: 39323
39    % reconsulted /a/news@google/test14days/lxbub.b in module user, 2877 msec 27981472 bytes
40   [consulting pos examples] [lxbub.f]
41   [cannot open] [lxbub.n]
42   yes
43      ?- induce.
44   ...
45   ...
```

Listing D.3: Starting *Aleph* and loading the learning datasets.

In Listing D.3 we can see the command line interaction taken to start Aleph, loading its learning data files and starting induction. Line one shows the bash prompt, indicating, among other things, the current directory location. First we list the directory's content, through "`ls`" command and then start the YAP interpreter (line 7). Afterwards *Aleph* is started (line 10) and all learning data files are loaded through the *Aleph*'s "`read_all/1`" predicate (line 22). Among the loaded data files, we can also find several specific *Prolog* files, some from YAP, and some from our *Prolog* library "`/lib/prolog`". Namely we are loading "`utils.lgt`" and "`penpostags.lgt`" files, which contains a set of auxiliary predicates used in the *Aleph* learning data files. After loading the data we can finally initiate the induction process by evoking the *Aleph*'s "`induce/0`" predicate (line 43).

In this particular case with 15 days of *news* data, from where 39323 learning instances were extracted, the induction process took approximately 97 minutes and generated 6673 reduction rules. The machine used was a *MacBook* with a 2 GHz *Intel Core 2 Duo* CPU, having 4 GB of RAM. The final lines of this induction execution are shown in Listing D.4.

```
1   ...
2   ...
3   [Rule 6978] [Pos cover = 165 Neg cover = 0]
4   rule(A) :-
5      chunk(A,left,np), chunk(A,right,vp), inx(A,center:x,1,pos(cc)).
6
7   [Rule 6979] [Pos cover = 35 Neg cover = 0]
8   rule(A) :-
9      chunk(A,right,np), inx(A,left,3,pos(nn)), inx(A,center:x,1,pos(cc)).
10
11  [Rule 6982] [Pos cover = 24 Neg cover = 0]
12  rule(A) :-
13  [Rule 6984] [Pos cover = 25 Neg cover = 0]
14  rule(A) :-
15     chunk(A,right,np), inx(A,left,1,x), inx(A,center:x,1,and).
16
17  [Training set performance]
```

```
                Actual
            +            -
      + 39001           0          39001
Pred
      -  322            0           322

        39323           0          39323

Accuracy = 0.991811408081784
[Training set summary] [[39001,0,322,0]]
[time taken] [5192.898]
[total clauses constructed] [2949018]
```

Listing D.4: Final lines from the 15 days data induction process.

Appendix E

The Penn Treebank Tag Set

We include here the *Penn Treebank Tag* set (Marcus *et al.*, 1993) that we have used in our work for *part-of-speech* tagging. All the 48 tags are briefly described in the following tables:

Table E.1: The *Penn Treebank Tag* Set (1-36).

1.	CC	Coordinating conjunction
2.	CD	Cardinal number
3.	DT	Determiner
4.	EX	Existential there
5.	FW	Foreign word
6.	IN	Preposition or subordinating conjunction
7.	JJ	Adjective
8.	JJR	Adjective, comparative
9.	JJS	Adjective, superlative
10.	LS	List item marker
11.	MD	Modal
12.	NN	Noun, singular or mass
13.	NNS	Noun, plural
14.	NP	Proper noun, singular
15.	NPS	Proper noun, plural
16.	PDT	Predeterminer
17.	POS	Possessive ending
18.	PP	Personal pronoun
19.	PP$	Possessive pronoun
20.	RB	Adverb
21.	RBR	Adverb, comparative
22.	RBS	Adverb, superlative
23.	RP	Particle
24.	SYM	Symbol
25.	TO	to
26.	UH	Interjection
27.	VB	Verb, base form
28.	VBD	Verb, past tense
29.	VBG	Verb, gerund or present participle
30.	VBN	Verb, past participle
31.	VBP	Verb, non-3rd person singular present
32.	VBZ	Verb, 3rd person singular present
33.	WDT	Wh-determiner
34.	WP	Wh-pronoun
35.	WP$	Possessive wh-pronoun
36.	WRB	Wh-adverb

Table E.2: The *Penn Treebank Tag* Set (37-48: punctuation marks).

37.	#	Pound sign
38.	$	Dollar sign
39.	.	Sentence-final punctuation
40.	,	Comma
41.	:	Colon, semi-colon
42.	(	Left bracket character
43.	)	Right bracket character
44.	"	Straight double quote
45.	`	Left open single quote
46.	``	Left open double quote
47.	'	Right close single quote
48.	"	Right close double quote

References

Altschul, S.F., Madden, T.L., Schffer, A.A., Schäffer, R.A., Zhang, J., Zhang, Z., Miller, W. & Lipman, D.J. (1997). Gapped blast and psi-blast: a new generation of protein database search programs. *Nucleic Acids Res*, **25**, 3389--3402. 62

Anita, H.M. (2002). *Numerical Methods For Scientists and Engineers*. Birkhäuser Verlag, Basel, Germany, 2nd edn. 132

Barzilay, R. & Elhadad, N. (2003). Sentence alignment for monolingual comparable corpora. *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP).*, 25--33. 40

Barzilay, R. & Lee, L. (2002). Bootstrapping lexical choice via multiple-sequence alignment. *In Proceedings of the Conference on Empirical Methods in Natural Language Processing, (EMNLP)*, 164--171. 59

Barzilay, R. & Lee, L. (2003). Learning to paraphrase: An unsupervised approach using multiple-sequence alignment. In *HLT-NAACL 2003: Main Proceedings*, 16--23, Edmonton, Canada. 38, 39, 40, 41, 42, 45, 52, 53, 136

Baum, L. & Eagon, J. (1967). An inequality with applications to statistical estimation for probabilistic functions of Markov processes and to a model for ecology. *Bulletin of the American Mathematical Society*, **73**, 360--363. 22

Brants, T. (2000). Tnt -- a statistical part-of-speech tagger. In *ANLP*, 224--231. 34

Camacho, R. (1994). Learning stage transition rules with indlog. In GMD-Studien, ed., *Gesellschaft für Mathematik und Datenverarbeitung MBH*, vol. 237, 273--290. 99

Carroll, J., Minnen, G., Pearce, D., Canning, Y., Devlin, S. & Tait, J. (1999). Simplifying text for language-impaired readers. *In Proceedings of the 9th Conference of the European Chapter of the Association for Computational Linguistics (EACL 1999)*. 39

Chandrasekar, R. & Srinivas, B. (1997). Automatic induction of rules for text simplification. *Knowledge-Based Systems*, 10:183--190. 16, 39

Clarke, J. & Lapata, M. (2006). Constraint-based sentence compression an integer programming approach. In *Proceedings of the COLING/ACL on Main conference poster sessions*, 144--151, Association for Computational Linguistics, Morristown, NJ, USA. 25, 148

Clarkson, P. & Rosenfeld, R. (1997). Statistical language modeling using the cmu-cambridge toolkit. In *PROCEEDINGS EUROSPEECH*, 2707--2710. 122

Cohen, J. (1968). Weighted kappa: Nominal scale agreement provision for scaled disagreement or partial credit. *Psychological bulletin*, **70**, 213--220. 142

Collins, M. (1997). Three generative, lexicalised models for statistical parsing. In *In Proceedings of the 35th Annual Meeting of the Association for Computational Linguistics*, 16--23. 29

Cordeiro, J.P., Dias, G. & Brazdil, P. (2007a). Learning paraphrases from wns corpora. In *FLAIRS Conference*, 193--198. 12, 55, 149

Cordeiro, J.P., Dias, G. & Brazdil, P. (2007b). A metric for paraphrase detection. In *Proceedings of the 2nd International Multi-Conference on Computing in the Global Information Technology (ICCGI 2007)*, IEEE Computer Society, Guadeloupe, French Caribbean. 12, 55

Cordeiro, J.P., Dias, G. & Brazdil, P. (2007c). New functions for unsupervised asymmetrical paraphrase detection. *Journal of Software*, **2**, 12--23, dBLP. 12, 55, 149

Cordeiro, J.P., Dias, G. & Cleuziou, G. (2007d). Biology based alignments of paraphrases for sentence compression. In *RTE '07: Proceedings of the ACL-PASCAL Workshop on Textual Entailment and Paraphrasing*, 177--184, Association for Computational Linguistics, Morristown, NJ, USA. 12, 45, 46, 48, 81, 149

Cordeiro, J.P., Dias, G. & Brazdil, P. (2009). Unsupervised induction of sentence compression rules. In *UCNLG+Sum '09: Proceedings of the 2009 Workshop on Language Generation and Summarisation*, 15--22, Association for Computational Linguistics, Morristown, NJ, USA. 13, 126, 149

Dayhoff, M.O., Schwartz, R.M. & Orcutt, B.C. (1978). A model of evolutionary change in proteins. *Atlas of protein sequence and structure*, **5**, 345--351. 70

Dias, G., Guilloré, S. & Lopes, J.G.P. (2000). Extraction automatique d'associations textuelles à partir de corpora non traités. *In Proceedings of 5th International Conference on the Statistical Analysis of Textual Data*, 213--221. 43, 55, 150

Dias, G., Moraliyski, R., Cordeiro, J.P., Doucet, A. & Ahonen-Myka, H. (2010). Automatic discovery of word semantic relations using paraphrase alignment and distributional lexical semantics analysis. *Journal of Natural Language Engineering. Special Issue on Distributional Lexical Semantics*, **16**, 439--467. 13, 45

Dolan, W.B., Quirk, C. & Brockett, C. (2004). Unsupervised construction of large paraphrase corpora: Exploiting massively parallel news sources. In *Proceedings of 20th International Conference on Computational Linguistics (COLING 2004)*. 40, 41, 45, 127

Edmundson, H.P. (1969). New methods in automatic extracting. *J. ACM*, **16**, 264--285. 3, 7

Enderton, H. (2001). *A Mathematical Introduction to Logic*. Academic Press, 2nd edn. 153, 156, 159, 160

Forney, J. & David, G. (1973). The Viterbi algorithm. *Proceedings of the IEEE*, **61**, 268--278. 19

Grefenstette, G. (1998). Producing intelligent telegraphic text reduction to provide an audio scanning service for the blind. *In Proceedings of AAAI spring Workshop on Intelligent Text Summarization*. 17, 39

Gregory, S.G., Barlow, K.F., McLay, K.E., Kaul, R., Swarbreck, D., Dunham, A., Scott, C.E., Howe, K.L., Woodfine, K., Spencer, C.C., Jones, M.C., Gillson, C., Searle, S., Zhou, Y., Kokocinski, F., McDonald, L., Evans, R., Phillips, K., Atkinson, A., Cooper, R., Jones, C., Hall, R.E., Andrews, T.D., Lloyd, C., Ainscough, R., Almeida, J.P., Ambrose, K.D., Anderson, F., Andrew, R.W., Ashwell, R.I., Aubin, K., Babbage, A.K., Bagguley, C.L., Bailey, J., Beasley, H., Bethel, G., Bird, C.P., Bray-Allen, S., Brown, J.Y., Brown, A.J., Buckley, D., Burton, J., Bye, J., Carder, C., Chapman, J.C., Clark, S.Y., Clarke, G., Clee, C., Cobley, V., Collier, R.E., Corby, N., Coville, G.J., Davies, J., Deadman, R., Dunn, M., Earthrowl, M., Ellington, A.G., Errington, H., Frankish, A., Frankland, J., French, L., Garner, P., Garnett, J., Gay, L., Ghori, M.R., Gibson, R., Gilby, L.M., Gillett, W., Glithero, R.J., Grafham, D.V., Griffiths, C., Griffiths-Jones, S., Grocock, R., Hammond, S., Harrison, E.S., Hart, E., Haugen, E., Heath, P.D., Holmes, S., Holt, K., Howden, P.J., Hunt, A.R., Hunt, S.E., Hunter, G., Isherwood, J., James, R., Johnson, C., Johnson, D., Joy, A., Kay, M., Kershaw, J.K., Kibukawa, M., Kimberley, A.M., King, A., Knights, A.J., Lad, H., Laird, G., Lawlor, S., Leongamornlert, D.A., Lloyd, D.M., Loveland, J., Lovell, J., Lush, M.J., Lyne, R., Martin, S., Mashreghi-Mohammadi, M., Matthews, L., Matthews, N.S., McLaren, S., Milne, S., Mistry, S., Moore, M.J., Nickerson, T., O'Dell, C.N., Oliver, K., Palmeiri, A., Palmer, S.A., Parker, A., Patel, D., Pearce, A.V., Peck, A.I., Pelan, S., Phelps, K., Phillimore, B.J., Plumb, R., Rajan, J., Raymond, C., Rouse, G., Saenphimmachak, C., Sehra, H.K., Sheridan, E., Shownkeen, R., Sims, S., Skuce, C.D., Smith, M., Steward, C., Subramanian, S., Sycamore, N., Tracey, A., Tromans, A., Helmond, Z.V., Wall, M., Wallis, J.M., White, S., Whitehead, S.L., Wilkinson, J.E., Willey, D.L., Williams, H., Wilming, L., Wray, P.W., Wu, Z., Coulson, A., Vaudin, M., Sulston, J.E., Durbin, R., Hubbard, T., Wooster, R., Dunham, I., Carter, N.P., McVean, G., Ross, M.T., Harrow, J., Olson, M.V., Beck, S., Rogers, J., Bentley, D.R., Banerjee, R., Bryant, S.P., Burford, D.C., Burrill, W.D., Clegg, S.M., Dhami, P., Dovey, O., Faulkner, L.M., Gribble, S.M., Langford, C.F., Pandian, R.D., Porter, K.M. & Prigmore, E. (2006). The dna sequence and biological annotation of human chromosome 1. *Nature*, **441**, 315--321. 58

Grigonyté, G., Cordeiro, J.P., Dias, G., Moraliyski, R. & Brazdil, P. (2010). A paraphrase alignment for synonym evidence discovery. In *Proceedings of the 23rd International Conference on Computational Linguistics (COLING 2010)*, 23--27, Beijing, China. 13, 149, 150

Grishman, R., Macleod, C. & Meyers, A. (1994). Comlex syntax: building a computational lexicon. In *Proceedings of the 15th conference on Computational linguistics*, 268--272, Association for Computational Linguistics, Morristown, NJ, USA. 24

Güvenir, H.A. & Cicekli, I. (1998). Learning translation templates from examples. *Inf. Syst.*, **23**, 353--363. 20

Hahn, U. & Mani, I. (2000). The challenges of automatic summarization. *IEEE Computer*, **33**, 29--36. 6

Harris, Z. (1968). *Mathematical Structures of Language*. Wiley, New York, NY, USA. 45

Hatzivassiloglou, V., Klavans, J.L. & Eskin, E. (1999). Detecting text similarity over short passages: Exploring linguistic feature combinations via machine learning. *In Proceedings of Empirical Methods in Natural Language Processing and Very Large Corpora (EMNLP 1999)*. 40

Henikoff, S. & Henikoff, J.G. (1992). Amino acid substitution matrices from protein blocks. *Proceedings of the National Academy of Sciences USA (PNAS)*, **89**, 10915--10919. 70

Heyer, L.J., Kruglyak, S. & Yooseph, S. (1999). Exploring expression data: Identification and analysis of coexpressed genes. *Genome Research*, **9**, 1106--1115. 53

Hogg, R., McKean, J. & Craig, A. (2005). *Introduction to Mathematical Statistics*. Upper Saddle River, NJ: Pearson Prentice Hall. 53

Jain, A.K., Murty, M.N. & Flynn, P.J. (1999). Data clustering: A review. *ACM Computing Surveys*, **264--323**. 53

Jing, H. (2000). Sentence reduction for automatic text summarization. 23, 148

Jing, H. & McKeown, K. (2000). Cut and paste based text summarization. *In Proceedings of 1st Meeting of the North American Chapter of the Association for Computational Linguistics*, 178--185. 23, 38, 39, 148

Jones, K.S. (1993). Discourse modelling for automatic summarising. Tech. Rep. UCAM-CL-TR-290, University of Cambridge, Computer Laboratory, 15 JJ Thomson Avenue, Cambridge CB3 0FD, United Kingdom, phone +44 1223 763500. 1, 6

Jones, K.S. (1999). Automatic summarising: Factors and directions. In *Advances in Automatic Text Summarization*, 1--12, MIT Press. 5, 6

Jones, K.S. (2007). Automatic summarising: The state of the art. *Information Processing and Management*, **43**, 1449--1481. 5, 6

Jurafsky, D. & Martin, J. (2000). *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics and Speech*. Prentice-Hall, NJ, USA. 28

Katz, S.M. (1987). Estimation of probabilities from sparse data for the language model component of a speech recognizer. In *IEEE Transactions on Acoustics, Speech and Signal Processing*, 400--401. 19

Knight, K. & Marcu, D. (2002). Summarization beyond sentence extraction: A probabilistic approach to sentence compression. *Artificial Intelligence*, **139**, 91--107. 22, 27, 31, 38, 39, 48, 127, 129, 131, 141, 148

Kramer, S. (2000). Thesis: Relational learning vs. propositionalization. *AI Commun.*, **13**, 275--276. 86

Landis, J. & Koch, G. (1977). Measurement of observer agreement for categorical data. *Biometrics*, **33**, 159--174. 142

Langkilde, I. (2000). Forest-based statistical sentence generation. In *Proceedings of the 1st North American chapter of the Association for Computational Linguistics conference*, 170--177, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA. 30

Lapata, M. (2003). Probabilistic text structuring: Experiments with sentence ordering. *In Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL)*. 39

Lavrac, N. (2001). *Relational Data Mining*. Springer-Verlag New York, Inc., Secaucus, NJ, USA. 87, 92

Lavrac, N. & Dzeroski, S. (1994). *Inductive Logic Programming: Techniques and Applications*. Ellis Horwood, New York. 95

Levenshtein, V. (1966). Binary codes capable of correcting deletions, insertions, and reversals. *Soviet Physice-Doklady*, 10:707--710. 40, 41, 72

Levin, B. (1993). *English Verb Classes and Alternations*. University of Chicago Press, Chicago. 24

Lita, L.V., Rogati, M. & Lavie, A. (2005). Blanc: Learning evaluation metrics for mt. *In Proceedings of Human Language Technology and Empirical Methods in Natural Language Processing Joint Conference (HLT-EMNLP 2005)*. 40

Liu, H. (2004). Montylingua: An end-to-end natural language processor with common sense. 80

Luhn, H.P. (1958). The automatic creation of literature abstracts. *IBM Journal of Research Development*, **2**, 159--165. xvii, 3, 4

Mani, I. (2001). *Automatic Summarization*. Natural Language Processing, John Benjamins Publishing Company. 1, 7

Marcus, M.P., Marcinkiewicz, M.A. & Santorini, B. (1993). Building a large annotated corpus of english: the penn treebank. *Comput. Linguist.*, **19**, 313--330. 11, 183

Marsi, E. & Krahmer, E. (2005). Explorations in sentence fusion. *In Proceedings of the 10th European Workshop on Natural Language Generation*. 38, 39

McCord, M. (1990). English slot grammar. Tech. rep., IBM. 24

Mendelson, E. (1997). *Introduction to Mathematical Logic*. Chapman and Hall CRC, fourth edition edn. 153, 156

Miller, G.A. (1995). Wordnet: a lexical database for english. *Commun. ACM*, **38**, 39--41. 24

Mitchell, T. (1982). Generalization as search. *Artificial Intelligence*, **18**, 203--226. 93

Mitchell, T. (1997). *Machine Learning*. 85, 97

Muggleton, S. (1991). Inductive logic programming. *New Generation Computing*, **8(4)**, 295--318. 90

Muggleton, S. (1996). Learning from positive data. In S. Muggleton, ed., *ILP96*, LNAI, 358--376, SV. 100

Muggleton, S. (1999). Inductive logic programming: Issues, results and the challenge of learning language in logic. *Artificial Intelligence*, **114**, 283--296. 99

Nagao, M. (1984). A framework of a mechanical translation between japanese and english by analogy principle. In *Proc. of the international NATO symposium on Artificial and human intelligence*, 173--180, Elsevier North-Holland, Inc., New York, NY, USA. 20

Needleman, S. & Wunsch, C. (1970). A general method applicable to the search for similarities in amino acid sequence of two proteins. *Proceedings of the National Academy of Sciences USA (PNAS)*, 443--453. 58, 59

Nguyen, M.L., Horiguchi, S., Shimazu, A. & Ho, T.B. (2004). Example-based sentence reduction using the hidden markov model. *ACM Trans. Asian Lang. Inf. Process.*, **3**, 146--158. 1, 20, 21, 22, 38, 39, 127, 144, 149

Och, F. & Ney, H. (2003). A systematic comparison of various statistical alignment models. *Computational Linguistics*, 29(1):19--51. 40

OMG (2010). Omg unified modeling language (omg uml) infrastructure version 2.3. Tech. Rep. formal/2010-05-03. 78

Orengo, C., Brown, N. & Taylor, W. (1992). Fast structure alignment for protein databank searching. *Proteins*, 139--167. 62

Papineni, K., Roukos, S., Ward, T. & Zhu, W.J. (2001). Bleu: a method for automatic evaluation of machine translation. *IBM Research Report RC22176*. 41, 42

Plotkin, G. (1970). A note on inductive generalization. In *Machine Intelligence*, vol. 5, 153--163, Edinburgh University Press. 95

Plotkin, G.D. (1971). *Automatic Methods of Inductive Inference*. Ph.D. thesis, Edinburgh University. 95

Polak, E. (1971). Computational methods in optimization. *New York Academic Press*. 132

Quinlan, J.R. (1986). Induction of decision trees. *Machine Learning*, **1**, 81--106. 83

Quinlan, J.R. (1990). Learning logical deinitions from relations. In *Machine Learning.*, vol. 5, 239--266. 33, 39, 41. 95, 99

Quinlan, J.R. (1993). *C4.5: programs for machine learning*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA. 32

Robinson, J.A. (1965). A machine-oriented logic based on the resolution principle. *J. ACM*, **12**, 23--41. 89, 90, 97

Salton, G. & Buckley, C. (1988). Term weighting approaches in automatic text retrieval. *Information Processing and Management*, 24(5):513--523. 55

Sammut, C. (1993). The origins of inductive logic programming: A prehistoric tale. In *In Proceedings of the 3rd International Workshop on Inductive Logic Programming*, 127--147. 90

Santos Costa, V., Damas, L., Reis, R. & Azevedo, R. (2002). *YAP User's Manual*. http://www.ncc.up.pt/˜vsc/Yap. 117, 180

Shinyama, Y., Sekine, S., Sudo, K. & Grishman, R. (2002). Automatic paraphrase acquisition from news articles. *In Proceedings of Human Language Technology (HLT 2002)*. 38, 39

Sjöbergh, J. & Araki, K. (2006). Extraction based summarization usinga shortest path algorithm. *In Proceedings of 12th Annual Language Processing Conference (NLP 2006)*. 40

Smith, T. & Waterman, M. (1981). Identification of common molecular subsequences. *Journal of of molecular biology*. 62

Srinivasan (2000). The aleph manual, technical report. 93, 99, 101, 105, 117, 118, 180

Stent, A., Marge, M. & Singhai, M. (2005). Evaluating evaluation methods for generation in the presence of variation. *In Proceedings of the Sixth Conference on Intelligent Text Processing and Computational Linguistics (CICLing 2005)*. 40

Turian, J., Shen, L. & Melamed, I.D. (2003). Evaluation of machine translation and its evaluation. In *In Proceedings of MT Summit IX*, 386--393. 40

Turner, J. & Charniak, E. (2005). Supervised and unsupervised learning for sentence compression. In *ACL '05: Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics*, 290--297, Association for Computational Linguistics, Morristown, NJ, USA. 31, 148

Vandeghinste, V. & Pan, Y. (2004). Sentence compression for automated subtitling: A hybrid approach. In S.S. Marie-Francine Moens, ed., *Text Summarization Branches Out: Proceedings of the ACL-04 Workshop*, 89--95, Association for Computational Linguistics, Barcelona, Spain. 34, 36, 148

Viterbi, A.J. (1967). Error bounds for convolutional codes and an asymtotically optimum decoding algorithm. *IEEE Transactions on Information Theory*, **IT-13**, 260--267. 22

Waterman, M.S. (1984). Efficient sequence alignment algorithms. *Journal of Theoretical Biology*, **108**, 333--337. 59

Witbrock, M.J. & Mittal, V.O. (1999). Ultra-summarization (poster abstract): a statistical approach to generating highly condensed non-extractive summaries. In *SIGIR '99: Proceedings of the 22nd annual international ACM SIGIR conference on Research and development in information retrieval*, 315--316, ACM, New York, NY, USA. 18, 148

Witten, I. & Frank, E. (2005). *Data Mining: Practical Machine Learning Tools and Techniques*. Morgan Kaufmann, 2nd edn. 85

Yamamoto, M. & Church, K. (2001). Using suffix arrays to compute term frequency and document frequency for all substrings in a corpus. *Computational Linguistics*, 27(1):1--30. 41, 42, 44

Zhu, X. (2005). Semi-supervised learning literature survey. Tech. Rep. 1530, Computer Sciences, University of Wisconsin-Madison. 85

Index

f_{hill} functions, 48
$\mathcal{L}_{Universe}$, 158

aleph, 99, 111
algval, 58
alignment, 12
aligval, 74
AMT, 16
AP-functions, 45--47, 54
asymmetrical
    paraphrase, 47, 128
asymmetrical paraphrase, 45, 49
asymmetrical paraphrasing, 48
atomic formula, 157
atoms, 156
ATS, 2, 16
automatic machine translation, 16
automatic sentence
    compression, 7
    reduction, 7
    simplification, 7
automatic text generation, 39
Automatic Text Summarization, 1, 2
automatic text summarization, 15, 16
aval, 66
axioms, 160
    logical, 160

background knowledge, 95
baum-welsh learning algorithm, 22
bioinformatics, 58, 70, 72
BLEU, 40, 42, 43
BLOSUM, 70
bottom clause, 100
bubble, 9, 108, 110, 111, 113
bubbles, 9, 23

C4.5, 32
channel model, 29
chunks, 17
class attributes, 79
class diagram, 78
class name, 79
clausal theory, 88
clause, 87
    body, 87
    definite, 87
    head, 87
    horn, 87
    program, 87
clustering, 53
clustering algorithms, 52
complexity, 54
compression rate, 19
compression-related constraints, 27
confusion matrix, 134
consequent, 154, 161
crossings, 68, 69

dagsthul seminar, 5
decision-based model, 31
decoder, 29
deduction, 89, 161
Dependency Model, 17
derived class, 79
distributional hypothesis, 45

DNA, 58
DNA sequence, 63, 70
DNA sequence alignment, 9
dynamic alignment, 63
dynamic programming, 59

EBMT, 20
edit distance, 40, 41, 49, 72
edmundsonian paradigm, 7
EM clustering algorithm, 53
entailment
    inverse, 96
EQsegment, 108
evaluation
    correctness, 144
    n-gram simplification, 143
example-based machine translation, 20
exclusive lexical links, 46, 48
exclusive link, 68
extraction, 12

fact, 88
factors
    input, 5
    output, 6
    purpose, 6
Finite State Grammar, 17
first order logic, 86, 153, 156
FOL, 86, 91, 97
formula
    well formed, 157
FSG, 17

gap penalty, 59, 64
generalization
    least general, 95
gist, 1
global alignment, 59, 63
golden section search, 132

handcrafted rules, 23
hidden markov model, 20
hierarchical agglomerative clustering, 53
HMM, 20, 22
human genome project, 58
hypothesis space, 93

ID3, 84
ILP, 86, 90, 156
induction, 12
induction process, 10, 111
inductive logic programming, 86, 90
inference rules, 89
information
    age, 1, 2
    extraction, 1
    retrieval, 1
integer programming, 25
interpretation, 156, 158
inverse resolution, 96, 97

kernel, 10, 99, 109, 113

latent semantic analysis, 40
lattice, 94, 95
LCP, 44
least general generalization, 118
levenshtein distance, 40
lexical links, 50
lexicon, 24
lgg, 95
linguistic knowledge, 23
literal, 87
    negative, 87
    positive, 87
local alignment, 63
local alignments, 59
logic, 153
    axioms, 160

binary operators, 154
first order, 156
logic program, 88
logic programming, 87, 89, 156
logical equivalence, 155
logical implication, 155
first order logic, 159
propositional logic, 155
longest common prefix, 41, 42, 44
LP, 87
LSA, 40
LTAG, 17

machine learning, 23, 27, 83
mathematical logic, 156
model, 158
modus ponens, 155, 161
MSRPC
corpus, 128
mutation matrix, 59

n-gram overlap, 42
natural deduction, 155
natural language processing, 16, 39
needleman-wunsch algorithm, 59, 64
NIST, 40
NLP, 16, 39, 43
noisy channel model, 27, 33, 48

ontologies, 48
optimization problem, 25

pair-sub-segment, 108
paraphrase, 37
asymmetrical, 38
symmetrical, 38
paraphrase alignment, 57
paraphrase clustering, 52
paraphrastic sentences, 76
part-of-speech, 11, 17
partial order, 94
penalization branch, 49, 50
penn treebank, 11, 143
phrase, 23
POS, 11, 122, 143
precision, 134
predicate, 88
premises, 161
previous neighbors, 61
prolog, 89
proof theory, 89
proposition
valid, 154
propositional logic, 153
propositional symbol, 154
propositions, 154

QT clustering algorithm, 53

recall, 134
resolution, 90
inverse, 96
resolvent, 96
rlgg, 95
RNA, 58

scoring function, 70
segment
kernel, 10
left, 10
right, 10
semantic entailment, 89
sentence
in logic, 159
sentence asymmetry, 45
sentence full parsing, 24
sentence reduction, 15, 17, 20
sentence reduction rules, 23
sequence homology, 59

similarity matrix, 52
sintactic entailment, 89
smith-waterman algorithm, 62--64
source model, 28
SP-functions, 45, 47, 54
stop-words, 3
substitution, 88
substitution matrix, 59
subsumption, 93, 95
summarization
    abstractive, 6
    extractive, 7
sumo, 48, 49, 51, 53, 165
super class, 79
symmetrical paraphrase, 45
syntactic parser, 24

tautology, 155
template-reduction, 20
term, 157
text summarization, 2
theorem, 161
thesaurus, 48
thresholds, 131
translation-template learning, 20
TRR, 20, 21
true assignment, 154
TTL, 20

unification, 89, 97
unification algorithm, 89

valuation, 154
variables
    free, 159

web news stories, 36, 76, 129
well formed formula, 89
wff, 89, 157
word
    cue, 4
    key, 4
    title, 4
word n-gram overlap, 40
WordNet, 24