Parallel corpora word alignment and applications

Alberto Manuel Brandão Simões
(ambs@di.uminho.pt)

Dissertação submetida à Universidade do Minho para obtenção do grau de Mestre em Informática, elaborada sob a orientação de
Pedro Rangel Henriques e José João Almeida

Departamento de Informática
Escola de Engenharia
Universidade do Minho

Braga, 2004

Abstract

Parallel corpora are valuable resources on natural language processing and, in special, on the translation area. They can be used not only by translators, but also analyzed and processed by computers to learn and extract information about the languages.

In this document, we talk about some processes related with the parallel corpora life cycle. We will focus on the ***parallel corpora word alignment****.*

The necessity for a robust word-aligner arived with the TerminUM project which goal is to gather parallel corpora from different sources, align, analyze and use them to create bilingual resources like terminology or translation memories for machine translation.

The starting point was Twente-Aligner, an open-source word aligner developed by Djoerd Hiemstra. Its results were interesting but it worked only for small sized corpora.

The work done began with the re-engineering of Twente-Aligner, followed by the analysis of the alignment results and the development of several tools based on the extracted probabilistic dictionaries.

Resumo

Os corpora paralelos são recursos muito valiosos no processamento da linguagem natural e, em especial, na área da tradução. Podem ser usados não só por tradutores, mas também analizados e processados por computadores para aprender e extrair informação sobre as línguas.

Neste documento, falamos sobre alguns dos processos relacionados com o cíclo de vida dos corpora paralelos. Iremo-nos focar no ***alinhamento de corpora paralelo à palavra****.*

A necessidade de um alinhador à palavra robusto apareceu com o projecto TerminUM, que tem como principal objectivo recolher corpora paralelos de diferentes fontes, alinhar e usá-los para criar recursos bilingues como terminologia ou memórias de tradução para tradução automática.

O ponto de arranque foi o Twente-Aligner, um alinhador à palavra open-source*, desenvolvido por Djoerd Hiemstra. Os seus resultados eram interessantes mas só funcionava para corpora de tamanhos pequenos.*

O trabalho realizado iniciou com a re-engenharia do Twente-Aligner, seguida pela análise dos resultados do alinhamento e o desenvolvimento de várias ferramentas baseadas nos dicionários probabilísticos extraídos.

The re-engineering process was based on formal methods: the algorithms and data structures were formalized, optimized and re-implemented. The timings and alignment results were analyzed.

O processo de re-engenharia foi baseado em métodos formais: os algoritmos e estruturas de dados foram formalizados, optmizadados e re-implementados. Os tempo e resultados de alinhamento foram analizados.

The speed improvement derived from the re-engineering process and the scale-up derived of the alignment by chunks, permitted the alignment of bigger corpora. Bigger corpora makes dictionaries quality raise, and this makes new problems and new ideas possible.

Os melhoramentos em velocidade derivados do processo de re-engenharia e a escalabilidade derivada do alinhamento por fatias, permitiu o alinhamento de corpora maiores. Corpora maiores fazem aumentar a qualidade dos dicionários, o que torna novos problemas e ideias possíveis.

The probabilistic dictionaries created by the alignment process were used in different tasks. A first pair of tools was developed to search the dictionaries and their relation to the corpora. The probabilistic dictionaries were used to calculate a measure of how two sentences are translations of each other. This naive measure was used to prototype tools for aligning word sequences, to extract multiword terminology from corpora, and a "by example" machine translation software.

Os dicionários probabilisticos criados pelo processo de alinhamento foram usados em tarefas diferentes. Um primeiro par de ferramentas foi desenvolvido para procurar nos dicionários e a sua relação com os corpora. Os dicionários probabilisticos foram usados para calcular uma medida de quão duas frases são tradução uma da outra. Este medida foi usada para prototipar ferramentas para o alinhamento de sequências de palavras, extrair terminologia multipalavra dos corpora, e uma aplicação automática de tradução "por exemplo."

Acknowledgments

Without help, I would never done this work:

- Thanks to my teachers José João Almeida and Pedro Rangel Henriques encouragement. It was crucial to maintain my motivation on high levels. For both of them, thanks a lot for the proofreading and ideas;
- Again, thanks for my teacher José João for the ideas, and work done in cooperation
- Thanks to Djoerd Hiemstra for his work on the Twente-Aligner and for licensing it as GPL and for his help. Thanks for the motivation on e-mails sent. Thanks, too, to Pernilla Danielson and Daniel Ridings for the GPL version of the vanilla aligner.
- Thanks to Diana Santos, for her patience with me and José João, and for presenting me to Daniel Ridings;
- Thanks to my TerminUM project colleagues: José Alves de Castro for his work in the Internet grabbing and validation for Parallel Corpora; and Bruno Martins for his work on the sub-title alignment and K-vec tools;
- Thanks to Gustavo Carneiro for the bug-hunting help, and some proofreading.
- Thanks to Rui Mendes for the proofreading and insults about my English;
- À minha família um grande obrigado pelo apoio, sustentação económica e afectiva. Especialmente aos meus pais pela educação que me deram e que me continuam a dar.
- Aos meus colegas do laboratório 1.05 do Departamento de Informática da Universidade do Minho pelos tempos bem passados, jantares, concursos de dardos, jogos de TetriNet e outros divertimentos não mencionáveis.

Preface

This document is a master thesis in Computer Science (area of Natural Language Processing) submitted to University of Minho, Braga, Portugal.

Through the document I will use the plural instead of the first person. This choice had two reasons: first, some of the work here presented was done in cooperation; second, the plural can help the reader to feel inside the work done.

Finally, I must advice the reader that most examples of parallel corpora and respective alignment is done with English and Portuguese languages. This means that a little knowledge of Portuguese can be helpful to understand completely the examples shown.

Document structure

Chapter 1 introduces the subject, defining the basic concepts used in the remaining document. It presents an overview of the concepts involved on parallel corpora creation, alignment and use (corpora life cycle), and application of the produced resources;

Chapter 2 describes the TerminUM project. It introduces how we are harvesting parallel texts from the Internet and how we align them at sentence level;

Chapter 3 describes the re-engineering process, explaining the basic algorithm, together with a discussion about the profiling of Twente-aligner, the tool our aligner is based on. Times comparisons and corpus size limitations are discussed;

Chapter 4 analyzes the NATools probabilistic dictionaries, and how they can be enhanced using simple techniques;

Chapter 5 shows the use of NATools probabilistic dictionaires: how the translation dictionaries obtained from the word alignment process can

be used in other tools, to obtain useful resources;

Chapter 6 Concludes this dissertation discussion and analysis of the work done. Explores some new tracks we can explore on future works.

Some complementary information is presented on the appendixes:

Appendix A Introduces a data structures calculus and respective notation.

Appendix B Includes the documentation for some tools developed through this thesis (also available as man pages).

Contents

1 Introduction **1**
- 1.1 Parallel Corpora Introduction . . . 3
  - 1.1.1 Corpora and Parallel Corpora . . . 3
  - 1.1.2 Parallel Corpora Alignment . . . 5
  - 1.1.3 Parallel Corpora Applications . . . 7
  - 1.1.4 Parallel Corpora Types . . . 8
- 1.2 NATools overview . . . 9
  - 1.2.1 Design principles . . . 10
  - 1.2.2 The NATools package . . . 11
- 1.3 Document Summary . . . 12

2 TerminUM Project **15**
- 2.1 The big picture . . . 15
- 2.2 Parallel corpora construction . . . 20
  - 2.2.1 Finding candidate pairs . . . 21
  - 2.2.2 Validation of candidate pairs . . . 26
- 2.3 Sentence alignment . . . 30
  - 2.3.1 Segmentation Tool . . . 32
  - 2.3.2 Sentence Aligners . . . 34

3 From Twente-Aligner to NATools **39**
- 3.1 Twente-Aligner . . . 39
- 3.2 NATools Architecture . . . 41
  - 3.2.1 Pre-Processing . . . 43
  - 3.2.2 Corpora encoding . . . 44
  - 3.2.3 Matrix initialization . . . 47
  - 3.2.4 EM-Algorithm . . . 51
  - 3.2.5 Dictionary creation . . . 52
- 3.3 Addition of Dictionaries . . . 53
  - 3.3.1 Motivation . . . 53

3.3.2 Addition formalization 54
3.4 Timings and measures 56
3.4.1 From Twente-Aligner to NATools 57
3.4.2 Scalability 58
3.4.3 Disk usage measures 61
3.5 Re-engineering Conclusions 63

4 Word Alignment Results 65
4.1 Simple alignment 66
4.2 Simple Multi-word term alignment 69
4.3 Simple Domain Specific Terminology Extraction 71
4.4 Iterative Alignment 73
4.5 Using Language specific pre-processing 74
4.5.1 English Genitives 74
4.5.2 Infinitive verbs 75
4.6 Results Analysis 78

5 NATools based Tools 79
5.1 NATools Programming Interface 80
5.1.1 Lexicon files' interface 81
5.1.2 Dictionary mapping files interface 82
5.1.3 Programming with NATools dictionaries 83
5.2 Word search on Parallel Corpora 86
5.3 Dictionary browsing 88
5.4 Estimation os Translation Probability 89
5.5 Word sequence alignment 91
5.6 Multi-word terminology extraction 93
5.6.1 Monolingual multi-word term extraction 93
5.6.2 Multilingue multi-word term extraction algorithm . . 93
5.6.3 Results and evaluation 94
5.7 Statistical machine translation 96
5.8 Tools development conclusions 99

6 Conclusions 101

A Mathematical Notation 111

B Software Documentation 115
B.1 Software Installation 115
B.1.1 Requirements 115
B.1.2 Compilation 116

B.2 nat-this 116
B.3 nat-pre 117
B.4 nat-initmat 119
B.5 nat-ipfp 120
B.6 nat-samplea 120
B.7 nat-sampleb 121
B.8 nat-mat2dic 122
B.9 nat-postbin 122
B.10 nat-css 123
B.11 NAT::Lexicon 124
B.12 NAT::Dict 126

List of Figures

2.1 Parallel corpora life-cycle 17
2.2 Overview of the Internet candidate pair retrieval process . . . 21
2.3 Sample page to enter a bilingual site 24
2.4 Schematic view of a bilingual site based on Resnik query structure 24
2.5 Example of two pages with bidirectional links 26
2.6 Schematic view of a bilingual site based on bi-directional links 26
2.7 Validation stages 27
2.8 Modules for a sentence aligner 31
2.9 Model and two examples of segments for a text 33

3.1 Word aligner structure 42
3.2 Initial co-occurrence matrix diagram 47
3.3 First 1000×1000 elements of the sparse matrix 49
3.4 Sparse matrix structure for the Bible alignment 49
3.5 Co-occurrence matrix first iteration diagram 51
3.6 Co-occurrence matrix second iteration diagram 51
3.7 Co-occurrence matrix final iteration diagram 52
3.8 Timing evolution regarding corpus size 58
3.9 Timing evolution with chunk alignment 61

4.1 Alignment before Portuguese verb stemming 76
4.2 Alignment after Portuguese verb stemming 77
4.3 Alignment after Portuguese and English verb stemming . . . 77
4.4 Alignment after Portuguese and English special verb stemming 78

5.1 Searching for a word on a parallel corpus 86
5.2 Searching for a words and probable translation 87
5.3 Dictionary navigation CGI 88
5.4 Word sequence alignment example 92

5.5 Relation between the number of units found and their occurrence number on the corpus 95
5.6 Example-Based Machine Translation example 98

List of Tables

3.1 Initial co-occurrence matrix 48
3.2 Twente-aligner vs NATools times 57
3.3 Times comparison of alignment in a single chunk 58
3.4 Measures for two halves of UEP corpus 59
3.5 Measures for the ten chunks of UEP corpus 59
3.6 Comparison between the times of alignment in two or ten chunks for the UEP corpus 60
3.7 Measures for the fourteen chunks of EP corpus 60
3.8 Disk usage in corpus analysis step 62
3.9 Disk usage for sparse matrices on disk 62
3.10 Disk usage for the dictionaries files 63
3.11 Disk usage for full alignment 63

4.1 Extract of the alignment for the words “God” and “Deus.” . . 66
4.2 Examples of alignment for “Deus” and “God” 67
4.3 Extract from the alignment for words “gosta” and “loves” . . . 68
4.4 Examples of alignment for “gosta” and “loves” 69
4.5 Extract from the word-pairs alignment result for “Christ Jesus” and “Jesus Cristo.” 70
4.6 Examples of alignment for “um pouco” and “a little” 71
4.7 Extract from the word-pairs alignment for “a little” and “um pouco” word pairs. 71
4.8 Extract from the result of the domain specific terminology extraction method. 72
4.9 Dictionaries sizes and qualities for the Iterative Alignment . . 73
4.10 Results of alignment using correctly tokenized genitives . . . 75

5.1 Translation probability measure samples 90
5.2 Multi-word terms translations 96

Chapter 1

Introduction

> [6] *and Yahweh said, "They are one people and they have one language. If they carry this through, nothing they decide to do from now on will be impossible.* [7] *Come! Let us go down and confuse their language so that they will no longer understand each other."*
>
> *Genesis 11, 6-7*

Corpora in general and, particularly, parallel corpora are very important resources for tasks in the translation field like linguistic studies, information retrieval systems development or natural language processing. In order to be useful, these resources must be available in reasonable quantities, because most application methods are based on statistics. The quality of the results depends a lot on the size of the corpora, which means robust tools are needed to build and process them.

The alignment at sentence and word levels makes parallel corpora both more interesting and more useful. As long as parallel corpora exist, sentence aligned parallel corpora is an issue which is solved by sentence aligners. Some of these tools are available as open-source software, while others have free licenses for non-commercial use, and produce reasonable results.

Regarding word level alignment, there are many interesting articles about the subject, referring many tools(Melamed, 2000; Hiemstra, 1998; Ahrenberg, Andersson, and Merkel, 2000). Unfortunately, most of them are not open-source nor freely available. Those that are available do not scale up to the size of corpora most researchers wish to align.

With this in mind, word alignment is one area where there is still a dire

need of research. Thus, this dissertation focuses upon the creation of better tools concerning word alignment.

For us, it is very important that the software used and developed follows the open-source philosophy. Without an open license, we cannot adapt the software to bigger applications, study the algorithms and implementations used or correct bugs.

Thus, we chose the only open-source word aligner found, Twente-aligner (Hiemstra, 1998), to help the bootstrap process for a parallel corpora package.

Starting with a working software tool saves a lot of time, which can be applied to more interesting work, as there is no need to develop the application from scratch.

> ***Note that...***
>
> *The dictionaries generated by Twente-Aligner were nice to look at, but hard to parse by another application. The solution was to rewrite the code that outputs the dictionary, to write it in a format that is easier to parse. Thus, if Twente-Aligner was not open-source its results would be very difficult to use by third-part software.*

This dissertation will present work developed in a package named NATools(Simões, 2003; Simões and Almeida, 2003)[1], which is being developed under the Linguateca (Santos et al., 2004) and the TerminUM project (Almeida et al., 2003). This package includes:

- a sentence-alignment tool, with built-in tokenization and segmentation of texts;
- a word-aligner based on Twente-aligner;
- web navigation scripts over aligned corpora;
- segment-alignment methods, based on word-alignment;
- and other tools[2]

The remaining of this chapter includes a section with a brief introduction to Parallel Corpora, corpora types, and applications. Section 1.2 presents a quick view over the resources and tools developed, while section 1.3 presents the structure of the document, summarizing each chapter.

[1] Natura Alignment Tools

[2] Some of these tools are still in prototype stage: they work but need interface improvements for common usage.

1.1 Parallel Corpora Introduction

This section presents an introduction to the parallel corpora concept. First, the concept of corpora and parallel corpora is explained, and the different levels of parallelism we can find on parallel texts are discussed. It explains how we can enrich parallel corpora by aligning them. Section 1.1.3 presents a quick survey of parallel corpora usages. Section 1.1.4 presents a classification of parallel corpora based on the dependency between original text and its translation, and on the objective of the translation.

1.1.1 Corpora and Parallel Corpora

Since the invention of writing, men wrote and continue to write large amounts of documents. When these texts are organized and annotated for a certain specific use (for example, for natural language processing), we call them *corpora*.

Definition 1 ***Corpora*** *is the term used on Linguistics, which corresponds to a (finite) collection of texts (in a specific language).* ◇

Note that...

There are many available corpora in different languages, and with different text categories: journalistic, literary, poetic and so on.
For the Portuguese language, CETEMPúblico (Rocha and Santos, 2000) is one of the most used monolingual corpus. It consists of over 191 million words of journalistic text segments from the Público Portuguese newspaper.
Some years after the release of CETEMPúblico, a similar approach was used to create a Brazilian corpus, based on "Folha de São Paulo" newspaper. CETEN-Folha consists of 33 million of words.
One example for the English language is the British National Corpus (`http://www.natcorp.ox.ac.uk/`), which contains both written and spoken modern English.
Normally, corpora include sentence mark-up, Part-Of-Speech, morphological analysis, and so on (like the first million of words from CETEMPúblico, part of the AC/DC project(Santos and Bick, 2000)).

A collection of documents in more than one language is called a multilingual corpora.

Multilingual corpora may be classified according to their properties, as was described in (Abaitua, 2000). The first level of multilingual corpora is

named **Texts in different idioms**. Normally they are used for quantitative and/or statistical studies.

> ***Note that...***
>
> *Quantitative and statistical studies can provide interesting hints about the number of verbs, adjectives and other morphological categories that appear on a specific portion of text in two different languages.*

Definition 2 ***Comparable corpora*** *are texts in different languages with the same main topic*[3]*.* ◇

> ***Note that...***
>
> *A set of news articles, from journals or news broadcast systems, as they refer the same event in different languages can be considered Comparable Corpora. Consider a news item about the September 11 tragedy. Different newspapers, in different languages will refer the World Trade Center, airplanes, Bin Laden and a lot of other specific words. This is the case of Comparable Corpora.*
> *This kind of corpora can be used by translators who know day-to-day language but need to learn how to translate a specific term. In this case, the translator can find a specific comparable corpora where the term is used, and read the translation to search for the translation of that specific term.*
> *They can also be used for terminology studies, comparing words used in different languages for similar concepts.*

Definition 3 ***Parallel corpora*** *is a collection of texts in different languages where one of them is the original text and the other are their translations.* ◇

> ***Note that...***
>
> *The COMPARA (`http://www.linguateca.pt/COMPARA/`) project is a parallel corpus of Portuguese/English fiction texts with about 40 thousand translation units (at the end of 2003).*
> *Another example is the well known Aligned Hansards of the 36th Parliament of Canada (`http://www.isi.edu/natural-language/download/hansard/`). This is a English/French corpus with more than one million translation units.*

[3] Although comparable corpora, we consider translations from a common third language as almost-parallel corpora, as we use them for direct alignment.

To formalize this concept, let us introduce some notation:

- languages are denoted by a calligraphic "$\mathcal{L}$" and a Greek letter index like $\mathcal{L}_\alpha$ or $\mathcal{L}_\gamma$;
- a text is written with an uppercase "T" and the respective Greek letter index: T_α is written in language $\mathcal{L}_\alpha$.
- the translation function $\mathcal{T}_{(\alpha,\beta)}$ applied to a text T_α returns its translated version: T_β: $\mathcal{T}_{(\alpha,\beta)}(T_\alpha) = T_\beta$. When languages can be inferred from the context, we will write T_β: $\mathcal{T}(T_\alpha) = T_\beta$ for simplicity.

Definition 4 *Two texts T_α, T_β in two different languages are **parallel texts**, or **bitexts**[4] (Harris, 1988) if one is the translation of the other[5]: $\mathcal{T}_{(\alpha,\beta)}(T_\alpha) = T_\beta$.* ◇

Given this, we can say that parallel corpora is a set of bitexts. Throughout this document, "parallel corpora" and "parallel texts" will be used interchangeably.

1.1.2 Parallel Corpora Alignment

> *The first step in extracting useful information from bitexts is to find corresponding words and/or text segment boundaries in their two halves (bitext Maps).*
> *Bitexts are of little use, however, without an automatic method for matching corresponding text units in their two halves.*
>
> *(Melamed, 1999)*

Although we can add morphological analysis, word lemmas, syntactic analysis and so on to parallel corpora, these properties are not specific to parallel corpora.

The first step to enrich parallel corpora is to enhance the parallelism between units on both texts. This process is called "alignment". Alignment can be done at different levels, from paragraphs, sentences, segments, words and characters.

Usually, alignment tools perform the alignment at sentence and word levels:

[4] for more than two texts we say they are *multitexts*

[5] or $\mathcal{T}_{(\beta,\alpha)}(T_\beta) = T_\alpha$. For the sake of simplicity we will use always T_α as the original text (and $\mathcal{L}_\alpha$ as the source language)

- Texts are sequences of sentences. To **sentence align** two texts is to create relationships between related sentences.

> ***Note that...***
>
> *Parallel corpora projects, such as COMPARA, or the Hansard of the 36th Parliament of Canada, are aligned at sentence level. This makes it easier to find the corresponding sentence or word in the target language.*

- The same idea can be used for the **word alignment** process: sentences are sequences of words. So, the word alignment process will add links between words from the original and the translated text.

  Word alignment can be viewed in two different ways:

  - for each word, in a sentence, find the corresponding word in the translated sentence. This means that, for each occurrence of a word, it has a specific word linked to it.
  - for each word from the source corpus, find a set of possible translations (and its probability) into the target corpus.

    This leads to a Probabilistic Translation Dictionary[6] (PTD), where for each different word of the corpus we have a set of possible translations and their respective probability of correctness. The following example illustrates this concept[7].

```
1 | novo
2 |   again      - 0.608494997024536
3 |   new        - 0.181253835558891
4 |   young      - 0.096220552921295
5 |   younger    - 0.032383352518081
6 |   back       - 0.017278868705034
7 |   trodden    - 0.016316164284944
```

  In this document, we will work with the second kind of alignment, as its result is more useful, and the first one can be obtained from it with small effort.

[6] In this document, when we use "translation dictionary" we are referring to these probabilistic translation dictionaries.

[7] This extract is from the word alignment of European Parliament corpus, which will be presented in further detail later.

Note that...

For multilingual information retrieval, to have a Probabilistic Translation Dictionary is more important than a word-to-word alignment scheme. For example, the probablistic information can be used to weight retrieved documents relevance.

These two definitions will be formalized in next chapters, but they are important to help the reader understand the following chapters.

1.1.3 Parallel Corpora Applications

Parallel corpora can be used for many tasks, e.g. teaching, terminological studies, automatic translation or cross-language information retrieval engines:

- *teaching second languages/translation didactics*
  parallel corpora can be searched by translation students to find translation samples, gather common errors done, and learn translation techniques. It can also be used in the process of learning a second language. By reading parallel texts, the student can try to understand the translated sentence and mentally align concepts and structures with the original one;

- *terminology studies*
  parallel corpora can be mined to bootstrap or enrich multilingual terminology dictionaries or thesaurus. In fact, when new knowledge areas appear, new terms will not be present on dictionaries. The word alignment process of parallel corpora is very important to aid the extraction of specific multilingual terminology;

- *automatic translation*
  by studying human translations, developers can learn and infer new automatic translation algorithms. As translation resources, the sentence aligned corpora can be used to create translation memories to be used on MBMT (memory-based machine translation), and the full word aligned corpora can be used for EBMT (example-based machine translation);

- *multilingual edition*
  as an alternative to the automatic translation, the multilingual edition intends to generate different languages from a meta-language: it

is defined an artificial language $\mathcal{L}$ where all information possible is inserted, such that it is possible to generate diverse natural languages from it. This method can be effective when generating texts in a closed environment;

- *product internationalization*
  similar to automatic translation, but with a narrower focus;

- *multilingual information retrieval*
  systems that gather documents in different languages, where the query is written in any language (the original objective of Twente-aligner). This means that the query must be translated to all languages used on the database documents. As the translated query is not shown to the user, word-by-word translation based on translation probability can be used, with effective results;

1.1.4 Parallel Corpora Types

To discuss parallel text alignment and understand alignment problems, we will begin by pointing out some translation characteristics. As presented in (Abaitua, 2000), we can classify translations according to the dependency between the original text and its translation:

- *Type A*
  when the translated text will completely substitute the original text in the target language. This is the case of literary translations (where readers will choose to read only one version of them);

- *Type B*
  when translations will coexist in time and space. This is the case of bilingual literary editions (where the reader will probably compare the texts on both languages);

- *Type C*
  when the translations will be used for the same purpose as the original, and work in a symmetrical way. This is the case for institutional documents of the European Union and other multilingual institutions;

or classify them with respect to the translation objective:

- *Pragmatic*
  the translated text will be used for the same communication purpose as the original;

- *Stylistic*
  the translated text tries to maintain the original text structure and form of language;

- *Semantic*
  the translated text tries to transmit essentially the same message.

Parallel text alignment problems are highly dependent on these classifications:

- *type A* translations cannot be viewed as parallel corpora. The translator often changes the order of sentences and some content[8] as soon as they maintain the basic idea behind the text;

- *type B* translations give reasonable results on word alignment, as most specific terms from the corpora will be coherently translated between sentences;

- *type C* translations are the best type of parallel corpora for alignment. As this type of parallel corpora is normally composed of institutional documents with laws and other important information, translation is done accurately, so that no ambiguities are inserted in the text, and they maintain symmetrical coherence;

Considering the automatic translation objective, *stylistic* and *semantic* translation types can have problems. Stylistic approach makes the translator look for some similar sound, sentence construction, rhythm, or rhyme. This means that the translator will change some of the text semantic in favor of the text style. The semantic approach has the advantage that the text message and semantic is maintained, but the type of language can change (as the translation will be addressed to an audience that differs significantly from the one of the original text).

1.2 NATools overview

This section is a quick overview of NATools. First, the state of Twente-aligner and our design principles to make it a better tool are presented. Then, a quick presentation of NATools package contents follows.

[8] like localization issues (changing inches to millimeters, for example).

1.2.1 Design principles

After choosing Twente-aligner as our base application, we defined our goals for the re-engineered tool:

- maintain it open-source (GPL):
  if the aligner can be useful to someone, it will be a lot more useful if it is open-source.
- use standard GNU configuration and compilation mechanics:
  in Unix, one of the most common problems when installing software is the anarchic profusion of methods available. One of the most common and recommended method is the `autoconf/automake`, which makes it simple to configure and compile software.
- improve the existing algorithm efficiency:
  results presented by Twente-aligner were good, but it used too much time to align small pieces of parallel corpora. Improving its speed allows us to produce much more material in smaller amounts of time.
- make it scalable to bigger corpora:
  as has been previously mentioned, parallel corpora (and corpora in general) are needed in big quantities to achieve good results. Then, if we want to align and build good PTDs, it is crucial to have an aligner that scales up to big corpora.
- use scripting languages to add levels for pre and post-processing:
  Probability translation dictionaries can be used for several interesting tasks:
  - treat corpora before alignment (tokenizing and segmenting it), and in some cases, apply stemmers and POS taggers;
  - create different formats for PTDs;
  - build tools to analyze generated PTDs;
  - try to use word-aligned corpora for translation tools.

Given these goals, we used the following approach to the aligner re-engineering:

1. writing formal models of each step of the alignment process;
2. detection of critic data structures and algorithms;

3. code profiling of the problematic areas;

4. calculus of isomorphic data structures for the slowest portions of the code;

5. re-codification of the application;

After the application of these design principles, the alignment method achieved a huge efficiency improvement and, consequently, large corpora began to be alignable.

After some tests with bigger corpora, we noticed that the process needed more memory than the physical memory available. The solution was to divide the problem in smaller pieces, align them independently and join the results. Given the complete independence between the alignment of these pieces, the use of parallel processing is simplified.

Various corpora were aligned for debugging and testing purposes, including the full European Parliament parallel corpora(Koehn, 2002). Some specific corpora were aligned for their authors: Andrius Utka parallel corpus (a 6 million English/French parallel corpus constituted by European Parliament documents) and for TMX alignment of the "Corpus Lingüístico da Universidade de Vigo[9]."

1.2.2 The NATools package

After the re-engineering of Twente-Aligner, we created a NATools package adding auxiliary programs and scripts.

First, Perl scripts were added to simplify the use of the aligner:

- TMX files can be aligned directly, without requiring format conversion(Simões and Almeida, 2004);

- huge parallel corpora can be aligned directly: the script checks its size, and splits it into chunks, if needed.

As Twente-Aligner is based on sentence aligned corpora, we added an implementation(Danielsson and Ridings, 1997) of vanilla aligner (based on (Gale and Church, 1991)). While not a good aligner[10], it is free. Although we

[9] With Portuguese/English, Portuguese/Galician, Galician/English alignments, available at `http://sli.uvigo.es/CLUVI/`

[10] Vanilla aligner is considered to be the low-level base code for sentence aligners. It just uses the similarity between sentence sizes to perform the alignment.

included it on the NATools package, the word aligner can be used in corpora aligned with any sentence aligner. Some other Perl scripts were added to the vanilla aligner, which makes it an independent and useful tool.

Given the efficiency boost, new research areas appeared. Thus, Perl applications were developed for[11]:

- translation ranking:
  based on a PTD, qualify or rank translations. This ranking is the probability of the proposed translation being a correct translation (see section 5.4):
  $$rank : T_\alpha \times T_\beta \longrightarrow \mathcal{P}\left(\mathcal{T}\left(T_\alpha\right) = T_\beta\right)$$

- sequence alignment:
  given a measure of translation quality between two sentences, it is possible to align sequence of words, which are part of a sentence (see section 5.5):
  $$sequenceAlign : S_\alpha \longrightarrow S_\beta$$

- bilingual multi-word term extraction:
  based on a monolingual multi-word term extractor(SENTA, 2003), extract from the source corpus a set of multi-word terms and find in the target corpus the corresponding translation (see section 5.6).
  $$extract : T_\alpha \times T_\beta \longrightarrow (w_\alpha^\star \times w_\beta^\star)^\star$$

- translation "by example:"
  based on a PTD and corresponding aligned corpus, try to translate sentences by aligning small sequences of text. (see section 5.7).
  $$translate : T_\alpha \longrightarrow T_\beta$$

1.3 Document Summary

TerminUM Project
The next chapter introduces the TerminUM Project. Our parallel corpora life-cycle is explained, showing where the word alignment is done, and which steps we need to perform before using it: parallel corpora harvesting from the Internet and sentence alignment.

[11] In the following function prototypes we did not include the dictionaries and parallel corpora as input for simplicity.

From Twente-Aligner to NATools
On chapter 3, the structure of the NATools word aligner is presented, along with an explanation of the involved algorithms and reverse engineering of the original Twente-aligner. The chapter ends with a benchmark of these two aligners, and an analysis of the resources needed by NATools for corpora of different sizes.

Word Alignment Results
This chapter is the linguistic analysis of the word alignment results, where we present extracts of probabilistic translation dictionaries built using NATools. It includes some experiences done with different kinds of corpora input, like corpora with stemmed verbs, or corpora with pairs of words.

Tools based on Dictionaries
Chapter 5 describes a variety of tools developed, which use the translation dictionaries extracted by NATools. These tools range from web-browsing of parallel corpora and dictionaries to a translation system "*by example*."

Conclusions and Future Work
Finally, chapter 6 concludes with an analysis of the work performed and the usefulness of the tools developed. It includes some ideas for future research on this area.

Chapter 2

TerminUM Project

The TerminUM Project is part of Natura Project, devoted to Natural Language Processing. The TerminUM team is composed by José João Almeida, José Alves de Castro Bruno Martins and me. We have been working on different stages of corpora processing.

As told in the introduction, our focus will be the word-aligner derived from Twente-Aligner, to which we called NATools. Before explaining its evolution, it is important to explain the context where this development is taking place. Next section explains the TerminUM goals. Afterwards, the discussion of the corpora harvesting, segmentation and sentence alignment is presented. These are the tasks that need to be performed before the word alignment process begins.

2.1 The big picture

The main goal of the TerminUM Project is to produce multilingual resources and develop tools to manage and enrich them. The base idea of making parallel corpora freely available is very similar to the work done in Opus(Tiedemann and Nygaard, 2003; Tiedemann and Nygaard, 2004). Although Opus is working on parallel corpora for any pair of words, TerminUM is being devoted to parallel corpora where one of the languages is Portuguese.

TerminUM is being developed incrementally and, each time the project elements meet, new ideas of multilingual resources and corpora treatment processes emerge. This leads to a step-by-step definition of a corpora life-cycle corpora, as presented in figure 2.1.

In this diagram, each boxed item is a tool, and italic words are data types.

Shadowed boxes are the main focus of this document. The TerminUM project wants to distribute each of the developed tools independently, as open-source applications. Each data type should be a resource, available to download by any user.

Although we call "life cycle" to this diagram, the parallel corpora processing can begin at any point, provided we have the appropriate resources to initiate the process.

The data types involved in the life-cycle are:

- web $\vee$ directory
  there are vast amounts of parallel corpora available on the web. The definition of an efficient process to grab parallel corpora from available resources is very important given that the biggest parallel corpora available is the web itself. Our life cycle starts with the process of building parallel corpora, which can be performed directly from Internet websites or from a set of files in our filesystem (see section 2.2);

- candidatePairs $\equiv \left(file^2\right)^\star$
  texts grabbed from the Internet must pass a set of steps before being considered bitexts. These files, already in our filesystem and ready to be validated will be called "candidate (file) pairs";

- bitexts $\equiv \left(file^2\right)^\star$
  as defined before (definition 4), bitexts are the texts which passed the validation process. These texts will be considered as real parallel texts

  $$\left(file_\alpha \times file_\beta : file_\alpha \in \mathcal{L}_\alpha \wedge file_\beta \in \mathcal{L}_\beta \wedge \mathcal{T}\left(file_\alpha\right) = file_\beta\right)^\star$$

  in subsequent processes.

- parallel books $\equiv (text_\alpha \times text_\beta)$
  parallel books, some of which are freely available on the web, can enter the process directly at this point, as we know they are real translations;

- segmented bitexts $\equiv \left((s^\star \times id)^2\right)^\star$
  are bitexts after the cleaning and sentence segmentation processes. The cleaning process may remove mark-up (tags, labels and commands) specific of file formats, as well as sentences that are too small (or segments without real natural language).

  The sentence segmentation process divides the text paragraphs into sentences (see section 2.3.1). The paragraph structure is inferred from the mark-up of the original file and may use language specific knowledge;

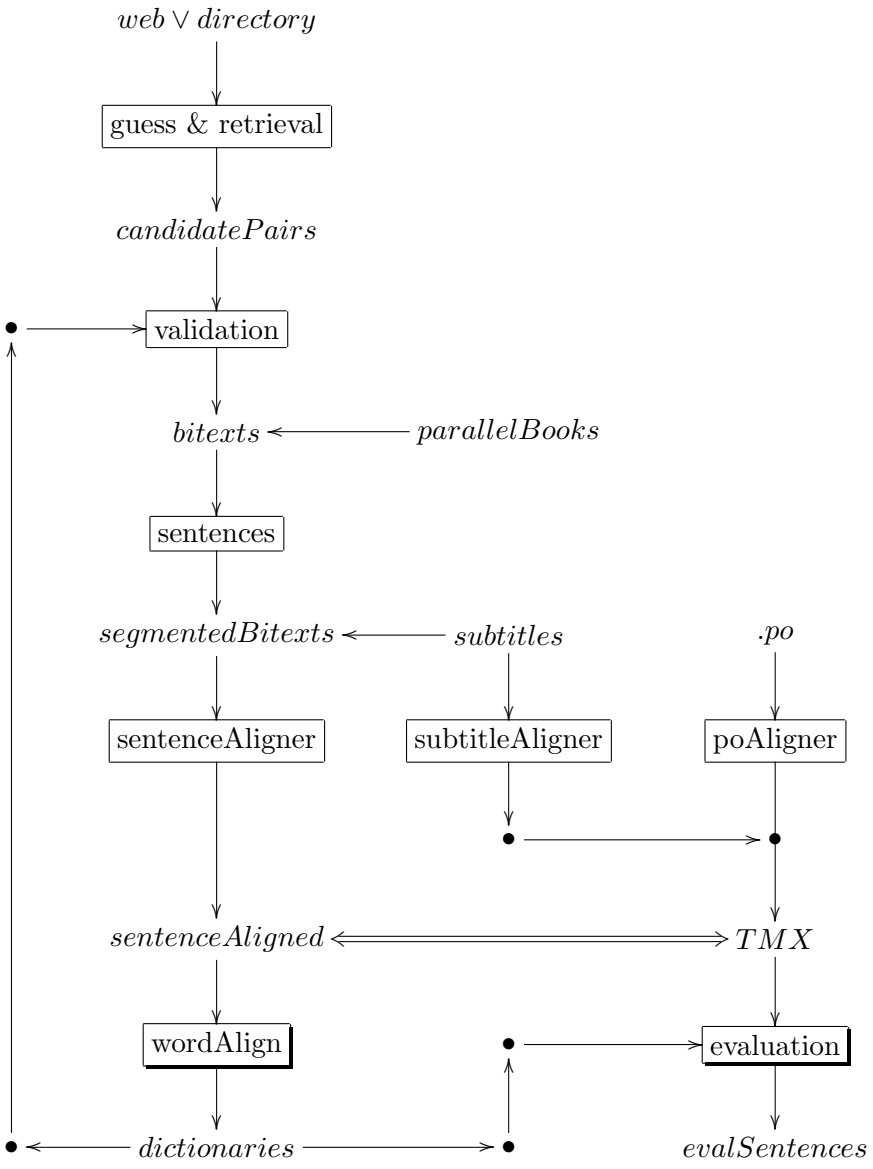

Figure 2.1: Parallel corpora life-cycle

- sentenceAligned $\equiv \left((s \times s)^\star \times id^2\right)^\star$
  texts aligned at sentence level consisting of pairs of sentences (and, optionally, an identifier pair which can be used to know the corpus where the sentence came from).

- probabilistic translat. dictionaries $\equiv (w_\alpha \mapsto (\#occ \times (w_\beta + null) \mapsto \%))^2$
  the word alignment process creates a probabilistic translation dictionary pair (one for each language, as the alignment process is not symmetrical).

  Each of these dictionaries maps a word (w_α) from the source language[1] to a tuple with the occurrence of the source word in the source corpus ($\#occ$), and another map, from words of the target language (w_β) to the probability (%) that w_β is a good translation of the original word (w_α).

- evalSentences $\equiv (s_\alpha \times s_\beta \times \%)^\star$
  as we will propose (section 5.4) a method to quantify (using statistical information) the translation probability between two segments, we can develop a tool to produce pairs of segments together with a translation probability estimation.

- subtitles $\equiv (subtitle_\alpha)^\star \times (subtitle_\beta)^\star$
  With the advent of DVD and "`DivX ;-)`" many sites with movie subtitles in different languages appeared on the Internet. These subtitles can be aligned (using timestamps or frame rates) and produce parallel texts;

  For completeness sake, here is an extract of a subtitle file for the Portuguese movie "Canção de Lisboa":

```
{82069}{82159}- So, I'm not a doctor?|- Thank God you're
              still here, doctor.
{82199}{82252}See? I really am a doctor!
{82290}{82360}Doctor,|one of your patients is very sick.
{82407}{82473}I need you to come with me|right away,
{82498}{82553}otherwise, he may pass away.
{82576}{82628}Let's see him then.|Come on.
{82628}{82713}Stay here and play with the lions.|They're
              harmless pets.
{82719}{82767}You know, duty calls.
{82810}{82873}So, am I a doctor or not?|Let's go.
```

[1] Note that the source language for one of the dictionaries is the target of the other and vice-versa.

```
{82875}{82914}- Call me doctor.|- Yes, doctor.
{82914}{82953}- Louder.|- Yes, doctor!
{82953}{83005}So, am I a doctor or not?
{83057}{83121}- Here's your patient.|- My patient?
{83135}{83198}There's some|misunderstanding here.
```

Numbers between curly brackets are the first frame of the movie where the subtitle will be shown, and the frame where it should disappear. Next to them is the subtitle itself, where the vertical bar is used to change lines.

- `.po` $\equiv (id \mapsto message_\alpha) \times (id \mapsto message_\beta)$
  Open source projects like Gnome or KDE use a GNU tool for localization which produce files known as `.po` files. Each `.po` file has relationships between an identifier and a message on the file language.

  An example of a `.po` file for Portuguese is:

```
#: src/dlg-add.c:124 src/window.c:1915 src/window.c:1948
msgid "Could not add the files to the archive"
msgstr "Não foi possível adicionar os ficheiros ao arquivo"

#: src/dlg-add.c:274 src/toolbar.h:56
msgid "Add"
msgstr "Adicionar"

#: src/dlg-add.c:285
msgid "_Add only if newer"
msgstr "_Adicionar se mais recente"

#: src/dlg-add.c:286
msgid "_Include subfolders"
msgstr "_Incluir sub-pastas"
```

  Each group consists of a comment (where the message was found), an identifier for the message (normally the message in the original language) and the message in the target language.

  Two of these files can be easily aligned using the message `msgid`. It is important to use two files and not `msgid` together with `msgstr` given that the identifier of the message can be out-of-date.

Although it is possible to follow the complete corpora life-cycle, this is not required. In fact, some of these tools are important because they can be used independently. This is true because it is common to need a specific resource in the middle of the life-cycle, or to have some corpora in a given

state and needing to perform only a few steps to reach another state of the life-cycle.

Each one of the life-cycle processes are, by themselves, an area of study and research.

Next follows a synthesis of the process that will be discussed in the rest of the document:

The first step is the parallel text acquisition. This is performed looking by a directory of files and trying to find files and their respective translations, or using some heuristics to retrieve parallel texts from the Internet in an automatic way. From this process we get pairs of texts which will need to pass a *validation* step to be considered real parallel texts. The retrieval and validation steps are presented in section 2.2 (Almeida, Simões, and Castro, 2002).

To align parallel texts at sentence level we need to divide each text in sentences (or paragraphs, or segments) and, using heuristics and dictionaries, relate them. This process is known as sentence alignment and will be discussed in section 2.3.

The word alignment process is based on *Twente-aligner* (Hiemstra, 1998; Hiemstra, August 1996) and is the main focus of the document. Chapter 3 presents the discussion on the method and implementation of the original *Twente-aligner* and how it was improved. Chapter 4 shows word-alignment results and discusses their quality according to the used corpora. Chapter 5 presents tools which use the alignment dictionaries for corpora browsing, sentence segment aligning, translation evaluation, statistical translation and multi-word term extraction.

For more information about the alignment of subtitles and internationalization messages check (Almeida et al., 2003), as this topic will not be covered in this document.

2.2 Parallel corpora construction

To use the Internet as a parallel corpora source there are two main tasks we must perform: the retrieval of the candidate parallel corpora and its validation(Almeida, Simões, and Castro, 2002; Almeida, Castro, and Simões, 2002).

The first step is not just a *download* issue. We should only download pages we suppose to be parallel. This means we need heuristics to detect parallel

pages (a process that is described on section 2.2.1), so we can retrieve them from the web. These pages will be called candidate file pairs.

Definition 5 *A **Candidate file pair**[2] is a pair of files that we expect to contain parallel texts.* ◇

The retrieved candidate pairs must pass a validation stage to be considered *parallel texts*. In fact, candidate file pairs (F_α, F_β) can be such that $\mathcal{T}(F_\alpha) = F_\beta \wedge F_\alpha \in \mathcal{L}_\alpha \wedge F_\beta \in \mathcal{L}_\beta$ is not guaranteed. The validation process is discussed on section 2.2.2.

2.2.1 Finding candidate pairs

The process of grabbing parallel corpora aims to find candidate file pairs in the Internet. This is specially useful because there are few other parallel corpora sources as translated documents (articles, books) freely available.

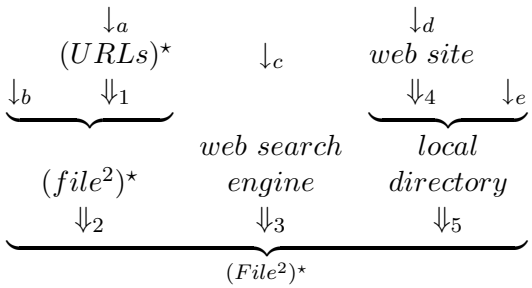

Figure 2.2: Overview of the Internet candidate pair retrieval process

There are different ways to detect parallel corpora from the Internet. Figure 2.2 illustrates them. On this figure, simple arrows are entry points for the retrieval process:

a) many tools create lists of URLs, like web spiders. These lists can be used to extract candidate parallel texts;

[2]Through this document "candidate pairs" will be used instead of "candidate file pairs" for simplicity.

Note that...

It is possible to write programs to retrieve a web page, extract links, and follow them. The resulting list of URLs can be something like:

```
1  http://escrital.pt/images/main8.jpg
2  http://gasa3.dcea.fct.unl.pt/gasa/gasa98/eco/dentes/img008.jpg
3  http://ftpdem.ubi.pt/dsp/ftp/d10free/mssql.zip
4  http://www.digiserve.pt/dsrvlogo_novo.gif
5  http://www.deec.isel.ipl.pt/eventos/je.../jetc99/pdf/art_60.pdf
6  http://www.cm-arcos-valdevez.pt/videopor.html
7  http://astrologia.sapo.pt/css/main.css
8  http://www.madinfo.pt/organismos/ceha/livros/Ed/Tania/back
9  http://www.bioportugal.pt/30.gif
10 http://www.joaquim-luis-cav.pt/imagens/botao1.jpg
11 http://www.adral.pt/_derived/index.htm_cmp_expeditn110_bnr.gif
12 http://www.metacortex.pt/img_prototipo/prototipo_r2_c1.gif
13 http://planeta.clix.pt/concurso/Concurso_ficheiros/filelist.xml
```

From these URL's we can exclude images and file types without textual content. Other files can be translated to textual formats for corpora creation.

b) you may have pairs of URLs which you know (or hope) to be parallel texts;

c) there are some techniques to find parallel texts directly on the Internet using web search engines;

d) you may know a web-site where several parallel texts exist;

e) after downloading a set of files, it is necessary to look for parallel texts in the file system;

The processed represented by the double arrows in figure 2.2 will be described in the next subsections and correspond vaguely to each entry point.

Using a list of URLs

From a list of URLs we can use heuristics to detect blocks that seem to refer to documents and their translations.

When web-masters build web-sites in more than one language they tend to use a systematic way of structuring web-pages names. For example, it is common to use a directory for each language (giving the name of the

language to the directory), to use a prefix or suffix to web-page files, to use CGI options, and so on.

Our strategy is to rely on this organization of web-sites when searching for language tags[3] on their URLs. This means we look for URLs that are very similar.

For example, the following URLs have a language tag, and should be normalized to a common string.

`http://www.ex.pt/index_pt.html` `http://www.ex.pt/index_en.html`

By removing the language tags and adjacent non-letters, these URLs would be normalized to `www.ex.pt/indexhtml`[4]. This process must also handle cases where the file in the main language does not contain the language tag.

Note that...

Given a list of more than eighteen million URLs from the Portuguese web, the block detection algorithm was applied to extract parallel corpora from the web. These are some examples of detected URLs blocks:

```
1  http://www.portugal-linha.pt/endereco/pt0001.html (pt)
2  http://www.portugal-linha.pt/endereco/de0001.html (de)
3  http://www.portugal-linha.pt/endereco/it0001.html (it)
4  http://www.portugal-linha.pt/endereco/no0001.html (no)

5  http://emcdda.kpnqwest.pt/pt/chap2/political.html (pt)
6  http://emcdda.kpnqwest.pt/en/chap2/political.html (en)
7  http://emcdda.kpnqwest.pt/de/chap2/political.html (de)

8  http://emcdda.kpnqwest.pt/pt/chap4/situation.html (pt)
9  http://emcdda.kpnqwest.pt/de/chap4/situation.html (de)
10 http://emcdda.kpnqwest.pt/en/chap4/situation.html (en)
```

Blocks with more than one URL consist of possible translations, where we can find candidate file pairs. We use the language tag to identify the URL language (this result is later confirmed with a language identifier, as shown on section 2.2.2).

Given two languages and a list of URLs, the process creates blocks and downloads respective files creating a list of candidate pair filenames.

[3](ISO 639, 1992) defines tags to represent languages. For example, for English it defines the following abbreviations: 'english', 'en' and 'eng'

[4]Domain portion of the URL is not processed

Using a list of filename pairs

The user can supply a list of URL pairs. These files will be downloaded and enter directly in the validation step. This step does not need to guess from the web. In fact, it is more a way to concatenate methods than a real method to infer parallelism.

Using a web search engine

(Resnik, 1998) presents an algorithm to detect candidate pairs using a web search engine like Google[5] or Altavista[6].

The idea is to find candidate pairs based on a *query* in a search engine. The query will ask for pages with at least two hyperlinks, each one containing one of the languages being searched.

These pages are normally used for bilingual web-sites front pages, where the user is presented with the choice for a language. Figure 2.3 shows a page with entry points for a Portuguese and an English version (figure 2.4 presents the schematic view of this kind of page).

Figure 2.3: Sample page to enter a bilingual site

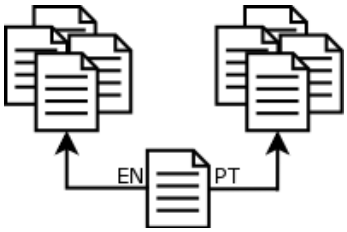

Figure 2.4: Schematic view of a bilingual site based on Resnik query structure

For Altavista(Overture Systems, 2004) we could write the following query:

```
1     (anchor:"portuguese" OR anchor:"portugues")
2 AND (anchor:"english" OR anchor:"ingles")
```

[5]http://www.google.com
[6]http://www.altavista.com

This method returns a big quantity of candidate pairs. We need to select a subset of the results obtained using some heuristics (use only some countries, some domains, remove suspicious addresses and so on). This subset of candidate pairs is then retrieved and a filename list is created. This list is then used in the validation step.

Using a known web-site

There are some organisms we know to have documentation in different languages — like European institutions (normally with documents for each country of the Union), Canadian government pages (in English and French), and so on.

We can supply a web-site address and all the web-site pages will be downloaded. The downloaded pages will be then processed as explained below for files in a local directory.

Using a local directory

To detect candidate pairs from a local directory we first try to use the same process used for a list of URLs (see subsection 2.2.1).

If the method fails or gives too few candidate pairs we can use two different approaches: a combinatory strategy or a mutual link strategy.

Combinatory strategy: create a list of combinations for the files in the directory. As this solution can give us an enormous list, files are processed with a language identifier tool (see subsection 2.2.2) and then, we only add to the list files on the desired languages and whose parent directories follow (for example) one of these conditions:

- both directories belong to the same dictionary (`/x/y/z` and `/x/y/w`);
- both directories have the same depth, and share a common parent (`/z/x/y` and `/z/a/b`);
- one of the directories is a parent of the other (`/x` and `/x/y/z`);

The pairs obtained by combining files are then validated for parallelism as is explained in the next section. This method should be used only when no other method wields acceptable results, given the high computational requirements.

Mutual link strategy: another issue to be addressed is the existence of bidirectional links. The idea is to find pages (say A) containing a link with a language name to another file (say B) where the file B has a link to the file A with another language name. This is very common on sites where the webmaster wants users to go directly from one page to its translation in another language. Figure 2.6 shows a schematic view of this web-site structure, while figure 2.5 shows a sample web-page.

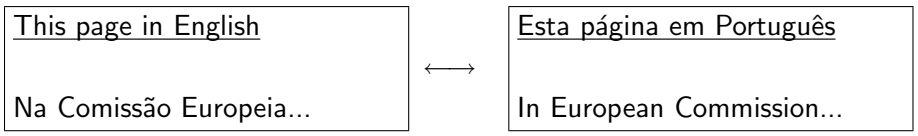

Figure 2.5: Example of two pages with bidirectional links

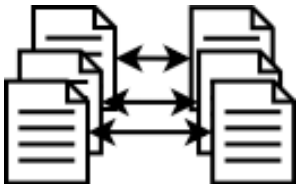

Figure 2.6: Schematic view of a bilingual site based on bi-directional links

All these methods should return a list of filename pairs referring to local files. These files will be tested for translation probability, using heuristics based on file statistics.

2.2.2 Validation of candidate pairs

None of the previous methods give us certainty to have parallel texts. In fact, some of the methods produce pairs where none of the files are in the chosen languages.

This process validates a set of aspects from the candidate pairs. Results are measured and weighted. Candidate pairs which do not have a good result have to be discarded (unfortunately this process may discard real translations).

Figure 2.7 shows some validation stages. The validation is always growing and new stages appearing. We can divide these stages in three major categories: language identification, file type check and file content check.

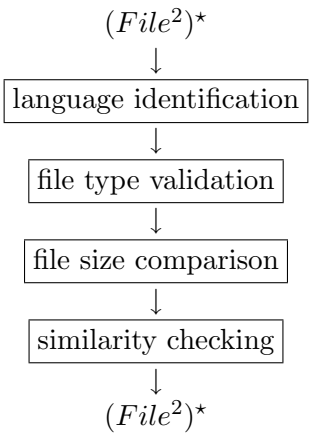

Figure 2.7: Validation stages

Language identification

Language identification is crucial as we don't have any assurance that the candidate pairs we retrieved from the Internet are in the languages we are trying to use. If the files are not in the desired languages, they must be discarded.

The language identification module (`Lingua::Identify`(de Castro, 2004)) uses implementations of various known methods for language identification like the "small word technique", the "trigram method"(Grefenstette, 1995) and prefix and suffix techniques. Although with different names, all these techniques rely one the same idea: there are some words, trigrams, suffixes and prefixes which are very common in a specific language. The module has a database relating language names to the most used words, trigrams, suffixes (and word endings) and prefixes (and word beginnings) on that language. The module then calculates statistically what is the most probable language.

Note that...

*The most common Portuguese words are "*de*", "*que*", "*a*", "*e*" and some more. Common word beginnings are "*par*", "*con*", "*est*" while common word endings include "*ção*", "*nte*", "*ara*".*

File type validation

Candidate pairs should have the same file type. In fact, they should be in a known file type, as we must know how to extract the text from the files.

The file type validation is performed using a Perl module available on CPAN and named `File::Munger`. It uses heuristics and magic[7] (`File::MMagic` (Darwin, 1997)) to detect file types.

After detecting the file type, `File::Munger` is used to access file specific information, like:

- file size;
- text content;
- meta-information, like title, author for LaTeX files, document types for XML, image sizes, and so on.

This step will discard pairs with no matching file types or with unknown file types.

File contents check

This step includes not only content comparison but also comparison of some meta-data like file size and file names.

Each of these methods requires a minimum similarity value and, if the candidate pair yields a lower result, it is removed from the candidate list. This leads to the removal of non-translation pairs and the cancellation of tests for these files, reducing the processing time. All these minimum values may be configured by setting program parameters.

Resulting values from the methods are used to compute a weighted average of all the methods to check if the final value is above a given threshold.

File size comparison Files which are translations of each other have, with high probability, a similar size. This is the idea behind this test. We compute the size of each file ignoring file mark-up(tags and/or commands). The resulting file size will be used not only to compute file similarity but also to discard files with small textual content.

[7] This is the way the author defines it. In fact, it is a set of specific bit headers which are compared – xor'ed – with file headers to detect their file-type. This is used, for example, by the "`file`" unix command

When computing sizes for two files we use the following formula to calculate a size comparison measure:

$$sizeComp(f_1, f_2) = \frac{min\left(size\left(f_1\right), size\left(f_2\right)\right)}{max\left(size\left(f_1\right), size\left(f_2\right)\right)}$$

Then, using a certain threshold, we can discard file pairs with big disparities regarding file size. The threshold values are dependent of the languages being compared, but for European languages typical values lie between 10% and 15%.

Similarity checking Before the explanation of the similarity checks performed, we must introduce some concepts:

Definition 6 *The **edit distance** between two strings is the number of simple operations (insertion or remotion of characters) needed to transform the first on the second string.*

Edit distance was first introduced by Vladimir Levenshtein in 1965 and is referred many times as the Levenshtein distance. ◇

> ***Note that...***
>
> *The edit distance between the words "*they*" and "*them*" is two: remove "y" and add "m".*

Definition 7 *The **maximum edit distance** possible between two strings is the sum of their lengths. This value is the number of operations necessary to remove all characters from the first string, and insert all the characters from the second one.* ◇

The similarity between two strings is their edit distance divided by their maximum edit distance:

$$similarity(\alpha, \beta) = \frac{edit\ distance(\alpha, \beta)}{max\ possible\ dist(\alpha, \beta)}$$

The concept of edit distance can be used to compare non-textual contents of files. For that purpose, we extract punctuation, dates, HTML tags, PDF or RTF commands and build a string for each one of them. Then, they will be compared using the edit distance.

Note that...

Punctuation lists extracted from two files (something like `!!!.??...` *and* `!!..??..!`*) can be compared for similarity, and yield a good measure of translation probability.*

While simple, this method can easily discard true translation pairs, and accept wrong pairs. Meanwhile, its use together with all the other methods tends to be reasonable to get true translation pairs.

2.3 Sentence alignment

In the introduction of this document, a brief introduction about what is a sentence aligner was presented: a tool to create correspondences between sentences on two parallel texts.

There are different tools for sentence alignment and they tend to have different functionalities. Some authors include in the alignment task the process of segmentation of the text into sentences. Others claim that the aligners must not divide the text, but should simply receive it already segmented.

To clarify these differences, and find relevant modules, we looked at three different tools, and how they work:

Trados WinAlign is an interactive aligner from Trados(TRADOS, Ireland Ltd, 2001) for Microsoft Windows. Given two texts in the same format, the program is configured to break sentences in specific places. Then, the tool will perform the alignment, and visually present the result. The user can interactively edit the alignment;

easy-align is part of the IMS Workbench(IMS Corpus Workbench, 1994-2002). This tool works over CQP(König, March 8, 1999 (CQP V2.2)) managed corpora. During the process of CQP corpora creation, the text is tokenized (divided in tokens) and then, easy-align uses this format to segment and align.

vanilla aligner is an implementation of (Gale and Church, 1991) algorithm implemented by (Danielsson and Ridings, 1997). It uses a pair of tokenized and segmented files, returning a pair of aligned text files.

These three tools are slightly different:

- only WinAlign includes a segmentation tool;
- the sentence alignment task is performed by all of the tools;
- all sentence aligners need a segmented text;
- only WinAlign has an interactive editor.

So, it is possible to define three independent modules, as can be seen on figure 2.8: segmentation, alignment and edition.

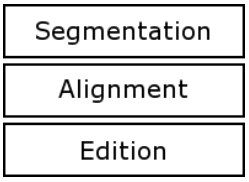

Figure 2.8: Modules for a sentence aligner

In this section, we will focus on the implementation of segmentation and alignment tools. The edition module will not be addressed because it is an interactive task. This makes it impossible to treat large amounts of text or to glue applications together.

We will start with the formalization of these two modules. As was previously explained, sentences are obviously necessary in order to align corpora at the sentence level. This task is performed by segmentation tools: starting from a text, these tools must be able to extract a list of sentences. We can define a segmentation function (denoted the "sentences" function) as:

$$sentences : text \longrightarrow sentence^{\star}$$

Applying this function to two parallel texts, we get a pair of sentence lists. Notice that the number of sentences on each list is not necessarily the same:

$$sentences(text_{\alpha}) \times sentences(text_{\beta}) = sentence_{\alpha}^{n} \times sentence_{\beta}^{m}$$

The alignment process should take this output and return aligned sentences, one from each language. This would be the ideal definition but not the real one:

$$sentalign : sentence_{\alpha}^{n} \times sentence_{\beta}^{m} \longrightarrow (sentence_{\alpha} \times sentence_{\beta})^{\star}$$

This is not the real definition because it is common to have sentences that we translate to two or more sentences in the target language. In some other cases, the translator removes (or inserts new) sentences. So, the real definition for the sentence alignment process should be:

$$sentalign : sentence_{\alpha}^{n} \times sentence_{\beta}^{m} \longrightarrow (sentence_{\alpha}^{p} \times sentence_{\beta}^{q})^{\star}$$

Next subsections will present:

- a module for Portuguese text segmentation, and a proposal of a method for the evaluation of segmentation tools;
- two sentence aligners tested in the TerminUM project.

2.3.1 Segmentation Tool

Our tests with sentence and word alignments were performed using languages based on Latin characters: Portuguese, English, French, Spanish and Galician.

Although different, we used the same segmentation tool for all these languages: `Lingua::PT::Segmentador`(Almeida et al., 2004). This is a Perl module for Portuguese text segmentation. Its goal is to simplify code reusability between Portuguese natural language research centers.

The segmentation tool needs to know where a sentence starts and where it ends. Ends of sentences are normally detected by the use of a punctuation mark, but this heuristic raises a lot of problems because we use punctuation marks for many other things: abbreviations, decimal digits, e-mails, URLs and others.

Some abbreviations are easy to find using regular expressions like e-mails, Internet addresses or decimal digits. For these, we use a list of regular expressions to match them in the text. For each successful match, we replace that text with a specific token (saving the relationship between the token and the text in an auxiliary matrix). These tokens are replaced back after full segmentation. To handle abbreviations we can use a list. This is the simpler way to track them, but fails whenever a new abbreviation is found.

An example of a difficult text to segment follows:

```
1 O Dr. Silva é arq. e está a trabalhar num proj. enorme.
2 A área do edifício a ser construído irá exceder os 200 m.
3 quadrados. Junto, e até à av. Continental, irá ser
4 construído um estádio para o F.C.B.
```

```
5 Esta obra monumental está orçada em 9.4 milhões de euros.
6 A data de conclusão está prevista para daqui a 5 meses.
```

Segmentation Tools evaluation

Although segmentation is not a focus of this document, some ideas about their evaluation were proposed by the Linguateca Project to the Portuguese community for Natural Language Processing during Avalon'03. This section tries to explain briefly how segmentation tools can be evaluated and how the proposed evaluation method can contribute to better segmentation tools.

The best way to examine a program without looking at its algorithm and/or implementation, is seeding it with some input, and look to its output: like a black box. To use this process in the segmentation tools evaluation we must feed them with texts, and look at the sequence of returned sentences. If we want to be able to evaluate them automatically (using a program for that purpose) we need a model that will be used to evaluate the answers. So, we need a set of texts and the corresponding sequence of sentences returned by the segmentation tool.

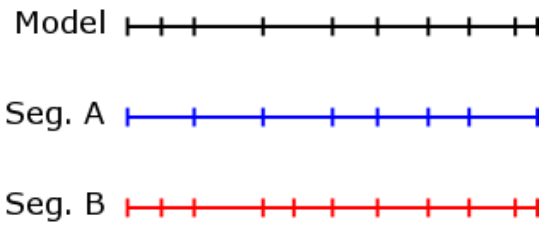

Figure 2.9: Model and two examples of segments for a text

Figure 2.9 shows a model and the output of two segmentation tools: A and B. Looking to this example, we will define three different methods of evaluation:

- the rigid comparison mode adds a point for each cut in the right place, and subtracts a point for each cut failed and for each cut in the wrong place. Using this method we have

$$\begin{aligned} A &= 6-2 = 4 \\ B &= 8-1 = 7 \end{aligned}$$

  and B would be better than A.

- the lazy comparison considers that the model is too difficult: some of the cuts are very difficult to find. So, if some of the segments to be evaluated contain two or more segments from the model, ignore the error. Meanwhile, cuts in the wrong place continue to be bad. Using this method, we have:

$$\begin{aligned} A &= 6-0 = 6 \\ B &= 8-1 = 7 \end{aligned}$$

  With this type of comparison, one could choose B because it finds more correct cuts. Meanwhile, A can be a good option given that it does not guess wrong cuts.

- finally, we can assign a different weight to each cut. If it is found, add its weight, if not found, subtract its weight, and subtract a given amount for each wrong cut. Although this method can be useful to give importance to some cuts, wrong cuts will have too low cost.

2.3.2 Sentence Aligners

The methods used to align parallel texts at the sentence level can be classified(Véronis, 2000) as statistical or linguistic methods. Other authors(Abaitua, 2000) suggest an hybrid method:

statistical method uses quantitative measures and their similarities between corpora (like sentence size, sentence character number, word occurrences and co-occurrences) to create an alignment relationship;

linguistic method uses linguistic knowledge like morphological analyzers, bilingual dictionaries, word list pairs, and so on, to relate sentences;

hybrid method combines the statistical and linguistic methods, using grammatical categories identification to achieve accurate statistical information.

For more detail on alignment methods and their implementation check (Caseli, 2003), where seven alignment methods are described and compared.

As we needed sentence aligned texts to serve as input for we used *easy-align* to prepare them. Later, the vanilla aligner was studied because *easy-align* is not open-source nor freely available[8]. The following two subsections present some details about these tools.

[8]We wanted to have a sentence aligner application in the NATools bundle.

Easy-align

Easy-align is an hybrid sentence aligner which is part of (IMS Corpus Workbench, 1994-2002). This set of applications was first developed for monolingual corpus management, namely with Corpus Query Workbench (König, March 8, 1999 (CQP V2.2)).

The alignment process appeared later using the same corpus engine. The idea behind CWB for parallel corpora is based on the inclusion of each language independently in the corpora database: then, *easy-align* uses these corpora to perform the alignment. Optionally, *easy-align* can use a bilingual lexicon (pairs of words) to achieve better results in the alignment process.

Easy-align is a robust aligner and produces very good alignments. Although not freely available nor open-source, it is available for the research community. We must also thank Stephen Evert (main developer of *easy-align*) for all the help using and configuring *easy-align*.

Vanilla Aligner

Vanilla aligner is a public domain sentence aligner implementing (Gale and Church, 1991) algorithm. Our implementation is based on (Danielsson and Ridings, 1997).

The aligner looks to each language text as a sequence of paragraphs, where each one is a sequence of sentences. The alignment will be done synchronizing paragraphs and aligning sentences looking to their relative size.

On each text file, the aligner expects a word or mark per line. Marks represent sentence and paragraph ends (for example `.EOP`: End Of Paragraph — `.EOS`: End Of Sentence). These marks are customizable on the command line.

```
1 | align -D '.EOP' -d '.EOS' file.en file.de
```

The "end of paragraph" mark is a rigid delimiter and is used for synchronization. The "end of sentence" mark is a soft delimiter and can be ignored by the aligner.

The idea behind this aligner algorithm is that a sentence and its translation will have approximately the same size (character count). This proves to be true in a high number of cases. When this is not true, and a big difference is found ($s_\alpha \gg s_\beta$), it is common to have a small sentence after s_β (say s'_β) whose size is the difference between s_α and s_β: $s_\alpha = s_\beta + s'_\beta$. This kind of

algorithm is applied to the whole text, calculating holes: sentences that do not align correctly with any other sentence.

The current algorithm aligns 1 to 1 sentence (the optimal case), 0 to 1 (when the translator adds a sentence), 1 to 0 (when the translator removes a sentence), 2 to 1 (when the translator creates a sentence joining the semantics of two original sentences), 1 to 2 (when the translator splits a sentence into two) and 2 to 2 (when the source language has a small sentence followed by a bigger one, and in the target language, there is a big sentence followed by a smaller one).

In (Danielsson and Ridings, 1997) this alignment process if fully presented, explaining for example that 3 to 1 or 1 to 3 alignment could only be possible if we delete the 0 to 1 and 1 to 0 possibilities. This happens as the algorithm would be unable to distinguish between a $(1-2, 0-1)$ and a $(1-3)$ sequence.

As the aligner method relies on the supposition that lengths of the original and translation sentences are very similar, it cannot be used in disparate languages like Hebrew and English where this does not occur. In these cases some other methods should be used as discussed on (Yaacov Choueka and Dagan, 2000).

Although with worse results than easy-align, this tool is very important given that is is open source, which means it can be used in a didactic context (to teach natural language processing and in particular sentence alignment) and serve as base for research for enhanced aligners.

A lot more can be said about segmentation, sentence alignment, and their evaluation. Evaluation methods for these two categories of tools will be proposed very soon by Linguateca.

For segmentation and alignment tools of the TerminUM project we think that it is important to:

- handle big quantities of texts: corpora are typically quite big, growing easily to thousands of Megabytes;
- glue with other tools: it is impossible to handle big quantities of corpora using interactive processes. Output of these tools must be computer readable (parsable);
- open-source: TerminUM is related to the teaching of Natural Language Processing and Terminology and Translation Tools and, as such, it is

important to be able to learn from these tools as to enhance them and test new techniques.

Chapter 3

From Twente-Aligner to NATools

Learning French is trivial: the word for horse is cheval, and everything else follows in the same way.

Alan J. Perlis

This chapter introduces the Twente-Aligner algorithm and enumerates its problems (why we changed it) and how we improved it (software re-engineering) in order to obtain the initial version of the NATools word aligner. A discussion on the importance of a probabilistic dictionary data type is formalized, and defined how two objects of this type can be summed up together. It concludes with a comparative study of both tools (Twente-Aligner and NATools) in terms of time and an analysis of the disk space needed for the alignment.

3.1 Twente-Aligner

Twente-Aligner is a word aligner developed by Djoerd Hiemstra (Hiemstra, 1998; Hiemstra, August 1996), for the project Twenty-One. This project's main goal is to put environmental organizations, research organizations and companies working together in the creation, distribution and use of common interest documents about ecology and sustainable development. These documents are written in different languages which makes information retrieval difficult. Twente-Aligner was developed to extract terminology from

the "Agenda 21"[1] parallel corpus to help in Cross Language Information Retrieval.

Twente-Aligner is open-source and licensed as GPL, making it a good starting point because it is possible to study the code, experiment and re-engineer it. Re-engineering was very important given a set of problems found on the Twente-Aligner original version:

- **Interface Problems:**
  the output from the alignment was given in a table whose format is extremely readable by humans but which is very difficult to use (e.g. to parse) by subsequent programs. Following is an example of the output generated by Twente-Aligner.

```
a                  ability              about
------------------ ------------------ ------------------
un           0.43  capacité     0.53  environ      0.43
une          0.24  dépend       0.35  sur          0.15
à            0.04  Or           0.03  mers         0.08
Le           0.04  disponibles  0.03  Elaboration  0.07
de           0.04  incertitude  0.03  concernant   0.06
la           0.03  empêche      0.02  offertes     0.02
les          0.03                     polluants    0.02
en           0.02                     lieu         0.02

ACCELERATE         Accelerating       acceptable
------------------ ------------------ ------------------
accélérer    0.60  accélération 0.80  acceptables  0.47
ACCELERER    0.20  étendre      0.20  acceptable   0.25
redoubler    0.12                     Elaborer     0.16
atténuer     0.08                     mondiale     0.04
                                      Observer     0.03
                                      conceptuel   0.02
                                      définitions  0.02
```

  Another problem of interface was the use of Twente-Aligner. It consists of a set of five different tools developed to be used in pipeline. Each of them takes as input the output of the other, until the last one, which produces the alignment result. Although it is nice to have different applications, dividing the big problem in small problems, to have to remember how to run each one independently is not practical.

[1] The United Nations conference on ecology and sustainable development in Rio de Janeiro in 1992

- **Robustness Problems:**
  the original Twente-Aligner worked for small corpora. When it was used on large corpora it broke down with illegal memory accesses. When the code was analyzed, we discovered not only many places where the code accesses out-of-bounds positions of arrays, but also many memory leaks.

- **Scalability Problems:**
  after the memory leak problems were solved, Twente-Aligner was robust enough to align medium size corpora. The alignment took a lot of time, but worked. Meanwhile, when experimented with large corpora, it could not align anything at all, given the physical limits of available memory.

The next section explains the NATools architecture with a special emphasis on the places where re-engineering was done to solve these problems.

3.2 NATools Architecture

Twente-Aligner was re-engineered in order to develop the NATools word-aligner, making it more robust, scalable and fast. This section explains the different steps involved in the NATools word aligner algorithm, including notes about how that particular portion of the code was re-engineered.

The aligner is based on statistical methods, counting co-occurrences of words from each language. To do the alignment we need the steps shown on figure 3.1, and quickly introduced here:

pre-process this first step cleans and prepares each corpus file. The process is language dependent. It includes the tokenization process and other forms of pre-processing whose goal is to enhance the alignment results as we will see on section 3.2.1;

encode encodes the corpora files into a binary format, where each word of the corpus is represented by an integer. It also creates a lexicon table with a bijection between identifiers and words.

mkMatrix prepares a sparse matrix with words' co-occurrences, used in the statistical alignment process;

EM-algorithm this algorithm's goal is to transform the sparse matrix: it enhances word translation cells and removes noise.

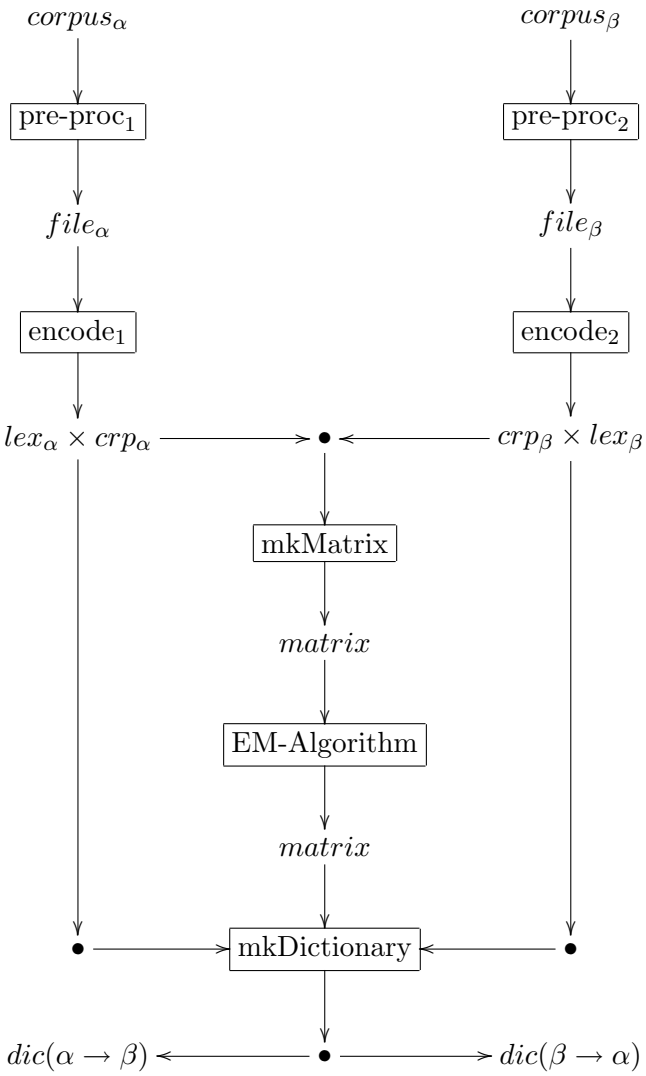

Figure 3.1: Word aligner structure

mkDictionary the final step on the alignment process is to interpret the resulting matrix, using the original lexicon tables and the matrix enhanced by the EM-algorithm, and to create a pair of dictionaries.

These tools work in pipeline, each one creating files the others will use. While some of the created files are only temporary, and used to communicate from one tool to the next, other files are crucial for subsequent use of the aligned lexicon. A top level script is provided so that the user does not need to know how to use each component of the aligner.

3.2.1 Pre-Processing

This is the only step in the alignment process that did not exist in the original Twente-Aligner. It is, in most cases, a simple tokenization process. In some cases, we can add a specific method to transform the corpora, and obtain more interesting alignment results.

Tokenization

The corpus' tokenization takes as input a text file, and returns a sequence of tokens (words surrounded by spaces, which will be considered by the word alignment process):

$$tokenize : text \longrightarrow token^{\star}$$

The input text for the word aligner must be sentence aligned. Thus, instead of a text, the tokenizer will receive a sequence of sentences or segments. It should maintain this division, and return a sequence of sequences of tokens:

$$tokenize : sentence^{n} \longrightarrow (token^{\star})^{n}$$

Tokenization should take care of different situations:

- *abbreviations*
  each language has specific abbreviations, which must be detected, or we can mistake abbreviation dots for sentence boundaries. If possible, these abbreviations should be expanded so that abbreviations and expanded versions can be considered the same word;

- *numbers*
  numbers are likely to contain dots, or special symbols that can be considered sentence or word boundaries. They must be detected and protected;

- *Internet tokens*
  URIs and e-mails should not be considered as standard text, or they will be tokenized too. As these strings are not likely to be translated, we can remove them from the corpus for alignment purposes, or protect them;

- *language specific issues*
  some languages have some specific issues which should be treated at this point, like the genitive in English, where it should be considered independently of the name it is associated with.

Corpora specific treatment

The normal word alignment process looks at words[2] as a simple sequence of characters. With the use of a pre-processing tool with knowledge about the language being processed, we can obtain more interesting results in the word alignment:

- One common problem when aligning English with Portuguese, is the big number of forms Portuguese verbs can take. This makes the co-occurrence of each form with the respective English form very low. A solution to this problem passes by the use of a pre-processing tool to stem Portuguese verbs. Section 4.5 explains with bigger detail how this was performed, and the results obtained;

- A different pre-processor can glue words in pairs, joining them by an underscore, for example. As the aligner looks to the tokens as sequences of characters, it will align pairs with pairs. This is a simple method to find multi-word unit translations. Check section 4.2 for a detailed explanation of this process.

- Another example, not discussed in this document, is to glue words with their respective Part-Of-Speech. This will allow homograph words to be distinguished.

3.2.2 Corpora encoding

After applying the correct language pre-processor to each text, we obtain the following data type:

$$(token_{\alpha}^{\star})^{n} \times (token_{\beta}^{\star})^{n}$$

[2]We will use "words" instead of "tokens" because it simplifies the text, and our main interest is in the alignment of "word tokens" and not the other tokens found on corpora.

which is a pair of sequences, of sequences of tokens.

Afterwards, the encoding process will first verify that the number of sentences (the size of each sequence of sequences of tokens) is the same (because the texts must be sentence aligned).

If the sizes match, each element of the pair will be treated independently. For each corpus, we will encode each different word with an identifier (a positive integer) and create two auxiliary files: a lexicon and a corpus file:

$$encode : (token_{\alpha}^{\star})^n \times (token_{\beta}^{\star})^n \longrightarrow (Lex_{\alpha} \times Crp_{\alpha}) \times (Lex_{\beta} \times Crp_{\beta})$$

In following steps each word will be coded as a number. However, the same identifier might refer to different words in different corpora. This operation is performed to achieve an efficiency gain, given that to compare words is more time-consuming than to compare integers. Besides the gain in efficiency, this will also provide a memory gain as integers are usually smaller than words.

Lexicon files

The lexicon file maps each different word to an unique identifier, and includes information about its number of occurrences:

$$Lex_{\alpha} = w_{\alpha} \rightharpoonup (id \times occurrenceCount)$$

Corpora files

The corpus file is a sequence of identifiers (integers) delimited by the zero integer:

$$Crp_{\alpha} = (id^{\star} \times 1)^n \equiv (id^{\star})^n \quad id \in \pi_1^{\star}(ran(Lex_{\alpha}))$$

This structure was enriched with a set of flags for each identifier. At the moment, these flags are being used to know how the original word was capitalized. Future versions could include more information, like the morphological class of each word. This would restrict the alignment to be performed only between words in the same morphological class.

Reverse Engineering

The file encoding with the original Twente-Aligner takes about 180 seconds to process a parallel Portuguese – English Bible (about 805K words).

As the first step for our analysis, we started by formalizing the data structures used in the original code. The full text is loaded to a text-buffer

and the sentences processed one at a time. For each word found, we search it on a list of words to check if it is a known word. For new words, a new identifier is created, while for existing words, their respective identifier is used.

The data structure for this original list can be formalized as

$$List = (word \times id \times occurrenceCount)^{\star}$$

without any type of condition (which means, for example, it is not sorted). Every time a new word appears in the corpus, the full list should be searched. If not found, the word is added to the end of the list. For each word in the corpus we do a medium of $\frac{n}{2}$ comparisons, where n is the size of the list. For the full corpus with k words, we got about $\sum_{n=1}^{k} \frac{n}{2}$ comparisons.

The proposed data structure is a binary search tree wich can be defined as

$$BT(X) = (X \times BT(X)^2) + 1$$

where

$$X = word \times id \times occurrenceCount$$

As we know from the computer science literature for each word we get $log_2(n)$ searches, where n is the number of words in the list. This means that for the full corpus we got $\sum_{n=1}^{k} log_2(n)$ comparisons.

Here it should be noticed that the binary tree is not being balanced because a normal corpus should have so different and random words that the tree will be very stable (unless you are aligning a sorted list of words).

Some other technical improvements were performed in this step:

- instead of duplicating each word when it is inserted into the binary tree, reuse the memory space from the original text;
- an auxiliary array with size equal to the number of different words was created to make it possible to directly access the tree node given to the word identifier;
- the corpus file is written in zipped format (using `zlib` library) reducing the disk space used;

The re-engineered version of this step takes about 4 seconds to process the same 805K word parallel corpora. More measures may be found in section 3.4.

3.2.3 Matrix initialization

This step creates a sparse matrix for the word alignment process, using the two enconded corpora files created in the previous step:

$$mkMatrix : Crp_\alpha \times Crp_\beta \longrightarrow Mat_{\alpha,\beta}$$

In this matrix, the row indexes represent each word identifier on the source corpus, and column indexes represent each word identifier on the target corpus. Each cell includes the number of times that two words (the one identified by the row index and the one identified by the column index) occur in the same aligned sentence:

$$Mat_{\alpha,\beta} = (i, j) \rightharpoonup count \equiv (w_\alpha, w_\beta) \rightharpoonup count$$

where:

- $1 \leq i \leq m$ with m the number of words on Crp_α ($w_\alpha \in Crp_\alpha$);
- $1 \leq j \leq n$ with n the number of words on Crp_β ($w_\beta \in Crp_\beta$);
- *count* the number of times words represented by i and j (w_α, w_β) appear on sentences $s_{\alpha,k}$ and $s_{\beta,k}$ which are aligned.

To understand the basic idea of the alignment process let us consider three sentences, presented in figure 3.2. When initializing the matrix, the algorithm does not know the translation of each word. So, it creates all possible relationships between them. This alignment is represented by a co-occurrence matrix, shown on table 3.1.

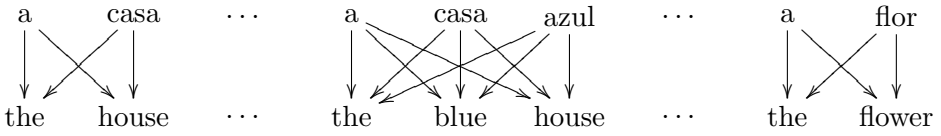

Figure 3.2: Initial co-occurrence matrix diagram

This relation is then enhanced using the EM-Algorithm that will be discussed in the next section.

Reverse Engineering

	the	house	blue	flower
a	3	2	1	1
casa	2	2	1	0
azul	1	1	1	0
flor	1	0	0	1

Table 3.1: Initial co-occurrence matrix

This step was taking too much time with the original Twente-aligner data structures. For the same Bible we mentioned in the previous section (about 805K words) this step takes 390 seconds.

Again, the data structure was analyzed and formalized. The matrix was implemented as a big chunk of memory divided in the number of rows of the matrix. Each cell contained the column number and the co-occurrence value.

$$Mat = ((col \times occ)^n)^m$$

where m is the number of rows in the matrix and n is the number of cells for each row.

Although each row was accessed directly by a simple calculation (size of each row times the index of the row) it was very slow. One reason was that each time a row had more elements than the number of cells available, the full matrix was reallocated and elements copied for each row (this was not a problem for small sized corpora).

To get a clear picture of how sparse the matrix is, we performed a small modification to the code for the matrix allocation function to print to a text file each cell it uses. From this map of Cartesian points we drawn figures 3.3 and 3.4. These figures are the first 1000×1000 and the full matrix for the bible alignment.

The different levels of sparse space for each row/column depends on how frequent words are. Most frequent words will have co-relation with most of the other words, leading to filled rows and/or columns.

The analysis of these matrices shows that it is not correct to give the same size to each row, as they have very different number of elements.

The new data structure is based on the original but with separate chunks for each row, which are allocated with a small amount of memory and grow according to the needs:

$$Mat = row \rightharpoonup (col \times occ)^\star$$

This simple operation made the amount of memory needed diminish, and

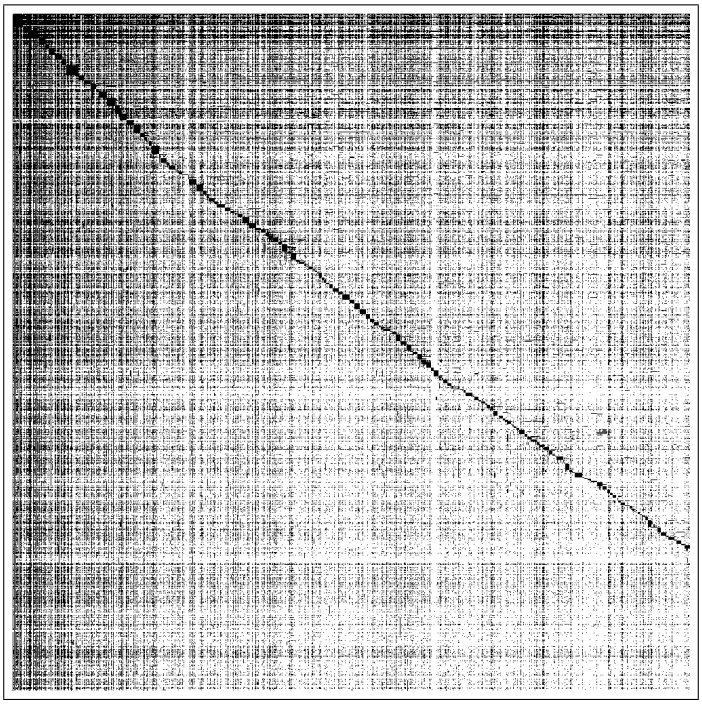

Figure 3.3: First 1000 × 1000 elements of the sparse matrix for the Bible alignment

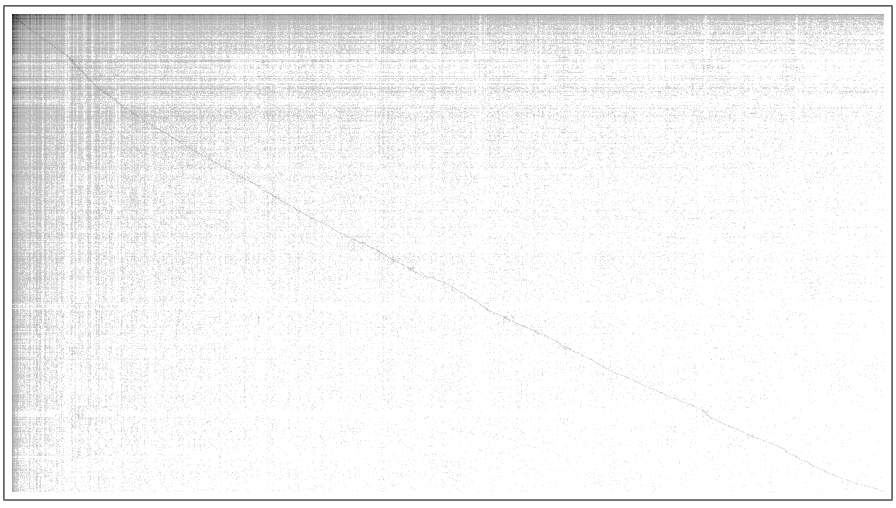

Figure 3.4: Sparse matrix structure for the Bible alignment

the process a little faster. At this point we used a profiler (`gprof`) to analyze what were the most frequently used functions[3]

```
 Each sample counts as 0.01 seconds.
  %   cumulative     self              self     total
time    seconds  seconds    calls  ms/call  ms/call  name
66.78      7.82     7.82 19936334     0.00     0.00  SearchItem
13.83      9.44     1.62 17628303     0.00     0.00  IncValue
11.96     10.84     1.40        1  1400.00 11664.00  InitEstimate
 2.13     11.09     0.25  2308931     0.00     0.00  Put
 1.79     11.30     0.21  2308031     0.00     0.00  PutValue
 1.28     11.45     0.15 15319372     0.00     0.00  Inc
 1.02     11.57     0.12 15321714     0.00     0.00  Get
 (...)
```

As the next step (the EM-Algorithm) uses the same data-structure, we applied the same profiling tool to check if there were any relations:

```
 Each sample counts as 0.01 seconds.
  %   cumulative     self              self    total
time    seconds  seconds    calls  ms/call  ms/call  name
81.33     62.39    62.39    27174     2.30     2.30  GetPartMatx
 8.15     68.64     6.25    27174     0.23     0.23  IPFP
 5.29     72.70     4.06 11437914     0.00     0.00  SearchItem
 1.33     73.72     1.02 11244130     0.00     0.00  IncValue
 0.93     74.43     0.71 11244130     0.00     0.00  OddsRatio
 0.59     74.88     0.45        2   225.00   256.09  MatrixTotal
 0.53     75.29     0.41 33890507     0.00     0.00  Get
(...)
```

These extracts show that functions `Get`, `Inc`, `IncValue`, `SearchItem` and `OddsRatio` are called very frequently. An analysis of the code and the number of calls for each of these functions, we noticed that `SearchItem` is called from the other functions and concluded it was the critical one.

The purpose of this function was to search for a given column on each row. Although the values are sorted, the search was linear. This time, we implemented a binary search over the buffer.

These two changes made this step take 5% of the original time (21 seconds instead of the original 390 seconds) and EM-Algorithm takes about 270 seconds instead of the original 2128 seconds (12% of the original time).

[3]In these two extracts times are from the algorithm after re-engineering of the data structure given that we did not save the original measures (oops!) but the most important part of these extracts is the number of calls for each function.

3.2.4 EM-Algorithm

The EM-Algorithm's (Entropy Maximization Algorithm) purpose is to iterate over the matrix created on the previous step, removing noise and enhancing the points of correct translations(Hiemstra, August 1996).

The signature for this step is simply:

$$EMAlgorithm : M_{\alpha,\beta} \longrightarrow M_{\alpha,\beta}$$

Let us look again at our example from figure 3.2. The EM-Algorithm will realize that there are a lot of correspondences between "a" and "the". This conclusion will result in the enhancement of the relationship between both words, as shown on figure 3.5.

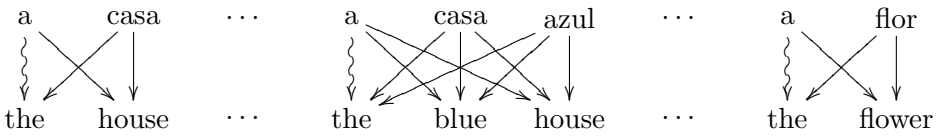

Figure 3.5: Co-occurrence matrix first iteration diagram

The same can be inferred from the relation between "casa" and "house". As this process continues it becomes apparent that other connections like "flor" and "flower" are correct (Pigeon hole principle). This would lead to a new version like the one shown on figure 3.6.

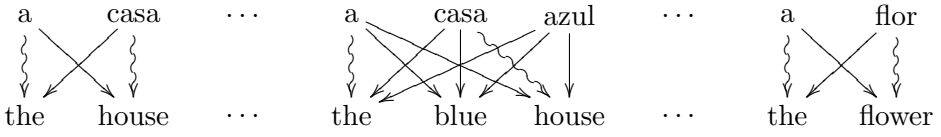

Figure 3.6: Co-occurrence matrix second iteration diagram

Finally, remaining words are connected. The resulting alignment is shown on figure 3.7. This algorithm has the advantage of convergence.

From the alignment we can estimate the translation probabilities like:

$$P\left(\mathcal{T}(a) = the\right) = 0.453$$

$$P\left(\mathcal{T}(casa) = house\right) = 0.876$$

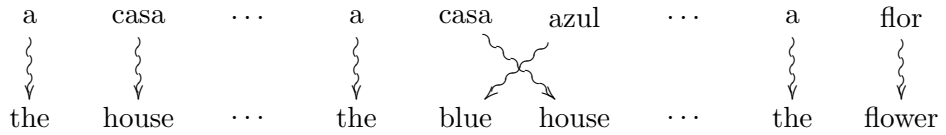

Figure 3.7: Co-occurrence matrix final iteration diagram

$$P\left(\mathcal{T}\left(azul\right) = blue\right) = 0.563$$

Note that...

This probabilistic information included into the dictionary is very useful. As discussed on (Melamed, 2000), in a cross-language information retrieval (CLIR) tool, a word-to-word translation dictionary would give the same importance for each translation. With probabilistic information we can gather the documents which seems more related with the original word.

From the matrices' figures shown in the previous section, we conclude that the main diagonal is normally the point of translation which means that both languages use very similar word sequences. The same would not be true for English and Hebrew, for example, but this method would also work for these languages, as sentences are treated as simple word bags instead of word sequences.

For a formal and detailed description of EM-Algorithm see Djoerd Hiemstra's master thesis (Hiemstra, August 1996).

3.2.5 Dictionary creation

After the iteration of the EM-Algorithm, the matrix should be traversed and interpreted to extract a pair of dictionaries. We need to extract two dictionaries because we do not have a symmetric matrix.

$$mkDictionary : Lex_\alpha \times Lex_\beta \times M_{\alpha,\beta} \longrightarrow \mathcal{D}(\mathcal{L}_\alpha, \mathcal{L}_\beta) \times \mathcal{D}(\mathcal{L}_\beta, \mathcal{L}_\alpha)$$

where

$$\mathcal{D}(\mathcal{L}_\alpha, \mathcal{L}_\beta) = w_\alpha \rightharpoonup (occur \times w_\beta \rightharpoonup P\left(\mathcal{T}\left(w_\alpha\right) = w_\beta\right))$$

The dictionary maps each word from the source language (w_α) to the number of times the word occurs in the source corpus, and a table of possible translations. Each one of these translations has an estimated probability of being the correct translation.

The quality of the probability dictionary obtained depends heavily on the translation quality of the used corpora.

3.3 Addition of Dictionaries

NATools dictionaries will be considered a data type. We will define an operation over dictionaries that we call Addition of Dictionaries.

This operation is an important process when aligning huge corpora. It can also be useful when aligning different corpora with the objective of joining them in a single dictionary.

3.3.1 Motivation

Although the changes presented in the previous section makes NATools able to align big corpora, it is still impossible to align huge corpora given that time and specially memory needs become unacceptable.

The natural approach is to split the corpus into several chunks and align them independently. The problem is that we obtain several different dictionaries instead of only one for the full corpus.

To solve this problem, an algorithm to sum dictionaries was developed (and will be discussed shortly). This way, we can join the created dictionaries and use all the chunks together as a single parallel corpus.

With this method, we changed the alignment process to check corpus size before alignment. If it is too big for single alignment, it is spliced, aligned by chunks, and summed up.

As the lexicon file is the same for all chunks (where words are being added), when aligning the last chunk this file will contain all words of the corpus. This means that the matrix used for this last alignment will have the same dimensions as the matrix one would use to align the entire corpus at the same time. Meanwhile, not all of these words appear in the chunk, and as such, the matrix will be very sparse.

As stated on the law known as “Zipf law”, if you take a big corpus and add some more text, new words will appear (Zipf, 1932). This means that each time a new chunk is processed new words appear.

Addition of dictionaries is also useful to concentrate results on a single dictionary. This means that we can align different corpora and at the end add their dictionaries together, building a bigger and, hopefully, better dictionary.

3.3.2 Addition formalization

Let us consider two dictionaries ($\mathcal{D}_1(\mathcal{L}_\alpha, \mathcal{L}_\beta)$ and $\mathcal{D}_2(\mathcal{L}_\alpha, \mathcal{L}_\beta)$). To simplify the notation let us abbreviate them to $\mathcal{D}_1$ and $\mathcal{D}_2$, and define some auxiliary functions:

- $\%(w_\alpha, w_\beta, \mathcal{D}) = \pi_2(\mathcal{D}(w_\alpha))(w_\beta)$
  the probability of w_β being a correct translation of w_α based on dictionary $\mathcal{D}$;
- $\#(w_\alpha, \mathcal{D}) = \pi_1(\mathcal{D}(w_\alpha))$
  the number of occurrences of word w_α on the source corpus, based on dictionary $\mathcal{D}$;
- $size(\mathcal{D}) = \sum_{w_\alpha \in dom(\mathcal{D})} \pi_1(\mathcal{D}(w_\alpha))$
  the total number of words on the source corpus of dictionary $\mathcal{D}$;

We want to define a formula to calculate $\mathcal{D}_{1+2} = \mathcal{D}_1 + \mathcal{D}_2$. For each word on the source language we need to:

- calculate the number of occurrences on both source corpora:

$$\#(w_\alpha, \mathcal{D}_{1+2}) = \#(w_\alpha, \mathcal{D}_1) + \#(w_\alpha, \mathcal{D}_2)$$

- sum the probabilities of translation by each word of the target corpora. This cannot be a simple addition of values because we must **preserve a probability value**: it must range between 0 and 1 (or 0 and 100). The simple way to solve this problem could be a simple mean:

$$\%(w_\alpha, w_\beta, \mathcal{D}_{1+2}) = \frac{\%(w_\alpha, w_\beta, \mathcal{D}_1) + \%(w_\alpha, w_\beta, \mathcal{D}_2)}{2}$$

  This formula has a big problem: both probabilities of the original dictionaries will weight the same on the final result. This is not fair because one of the dictionaries can contain a result based on many more occurrences than the other. To solve this problem, we can define the following formula which will **weight probabilities** according to their respective word occurrence count:

$$\frac{\%(w_\alpha, w_\beta, \mathcal{D}_1) \times \#(w_\alpha, \mathcal{D}_1) + \%(w_\alpha, w_\beta, \mathcal{D}_2) \times \#(w_\alpha, \mathcal{D}_2)}{\#(w_\alpha, \mathcal{D}_1) + \#(w_\alpha, \mathcal{D}_2)}$$

Again, in case we have a little corpus with specific content and a big generic one, a word can occur much more on the big corpus but be a word more relevant on the small one. **The relevance of a word in a corpus increases with the probability of getting that word when choosing a random word from the corpus**. We can define yet another formula to take this in consideration:

$$\frac{\%(w_\alpha, w_\beta, \mathcal{D}_1) \times \frac{\#(w_\alpha, \mathcal{D}_1)}{size(\mathcal{D}_1)} + \%(w_\alpha, w_\beta, \mathcal{D}_2) \times \frac{\#(w_\alpha, \mathcal{D}_2)}{size(\mathcal{D}_2)}}{\frac{\#(w_\alpha, \mathcal{D}_1)}{size(\mathcal{D}_1)} + \frac{\#(w_\alpha, \mathcal{D}_2)}{size(\mathcal{D}_2)}}$$

which can be simplified to

$$\frac{\%(w_\alpha, w_\beta, \mathcal{D}_1)\#(w_\alpha, \mathcal{D}_1)size(\mathcal{D}_2) + \%(w_\alpha, w_\beta, \mathcal{D}_2)\#(w_\alpha, \mathcal{D}_2)size(\mathcal{D}_1)}{\#(w_\alpha, \mathcal{D}_1)size(\mathcal{D}_2) + \#(w_\alpha, \mathcal{D}_2)size(\mathcal{D}_1)}$$

This is the formula used in case a word is proposed as translation on both dictionaries.

If a word w_α is translated as w_β only in one dictionary (say $\mathcal{D}_1$), then the formula shown above will be simplified to

$$\frac{\%(w_\alpha, w_\beta, \mathcal{D}_1) \times \#(w_\alpha, \mathcal{D}_1) \times size(\mathcal{D}_2) + 0 \times \#(w_\alpha, \mathcal{D}_2) \times size(\mathcal{D}_1)}{\#(w_\alpha, \mathcal{D}_1) \times size(\mathcal{D}_2) + \#(w_\alpha, \mathcal{D}_2) \times size(\mathcal{D}_1)}$$

which is equivalent to

$$\frac{\%(w_\alpha, w_\beta, \mathcal{D}_1) \times \#(w_\alpha, \mathcal{D}_1) \times size(\mathcal{D}_2)}{\#(w_\alpha, \mathcal{D}_1) \times size(\mathcal{D}_2) + \#(w_\alpha, \mathcal{D}_2) \times size(\mathcal{D}_1)}$$

Another case is when a word w_α exists on the source corpus of one dictionary (for example $\mathcal{D}_1$) but not on the other ($\mathcal{D}_2$). This means that the null probability is the **zero element**.

In this case, we get:

$$\frac{\%(w_\alpha, w_\beta, \mathcal{D}_1) \times \#(w_\alpha, \mathcal{D}_1) \times size(\mathcal{D}_2) + \%(w_\alpha, w_\beta, \mathcal{D}_2) \times 0 \times size(\mathcal{D}_1)}{\#(w_\alpha, \mathcal{D}_1) \times size(\mathcal{D}_2) + 0 \times size(\mathcal{D}_1)}$$

which is simplified to

$$\frac{\%(w_\alpha, w_\beta, \mathcal{D}_1) \times \#(w_\alpha, \mathcal{D}_1) \times size(\mathcal{D}_2)}{\#(w_\alpha, \mathcal{D}_1) \times size(\mathcal{D}_2)} = \%(w_\alpha, w_\beta, \mathcal{D}_1)$$

This formula should also be such that **the dictionary added with himself is itself**: $\%(w_\alpha, w_\beta, \mathcal{D}_{1+2}) = \%(w_\alpha, w_\beta, \mathcal{D}_1)$ when $\mathcal{D}_1 = \mathcal{D}_2$. So, in this case we get

$$\%(w_\alpha, w_\beta, \mathcal{D}_{1+1}) = \frac{\%(w_\alpha, w_\beta, \mathcal{D}_1) \times \frac{\#(w_\alpha, \mathcal{D}_1)}{size(\mathcal{D}_1)} + \%(w_\alpha, w_\beta, \mathcal{D}_1) \times \frac{\#(w_\alpha, \mathcal{D}_1)}{size(\mathcal{D}_1)}}{\frac{\#(w_\alpha, \mathcal{D}_1)}{size(\mathcal{D}_1)} + \frac{\#(w_\alpha, \mathcal{D}_1)}{size(\mathcal{D}_1)}}$$

which can be simplified to

$$\frac{\%(w_\alpha, w_\beta, \mathcal{D}_1) \times \left(\frac{\#(w_\alpha, \mathcal{D}_1)}{size(\mathcal{D}_1)} + \frac{\#(w_\alpha, \mathcal{D}_1)}{size(\mathcal{D}_1)}\right)}{\frac{\#(w_\alpha, \mathcal{D}_1)}{size(\mathcal{D}_1)} + \frac{\#(w_\alpha, \mathcal{D}_1)}{size(\mathcal{D}_1)}} = \%(w_\alpha, w_\beta, \mathcal{D}_1)$$

This database addition was implemented on the `NAT::Dict` Perl module using this last formula proposed, and latter ported to C for higher execution velocity.

3.4 Timings and measures

In this section, we will present measures about times and sizes of files in the word alignment process. In order to analyze the tools in different situations, we will use corpora with different characteristics: different sizes, different noise levels and in different languages.

- **Tom Sawyer** (TS)
  A parallel Portuguese/English classic from Mark Twain.
- **Harry Potter, and the Sorcerer's Stone** (HP)
  The first book from the Harry Potter series, parallel Portuguese/English;
- **Anacom** (IPC)
  An automatically generated parallel Portuguese/English corpus, taken from the web-site of the Portuguese communications authority: `http://www.ipc.pt`. The corpus was not manually reviewed and as such, contains a lot of noise;

- **Bible** (Bib)
  The already presented Portuguese/English Bible, where one of them is not the translation of the other, but they are both translations from a third language[4]

- 1/2**UtkaEuroParl** (HUEP)
  This is half of a 6 million word English/French parallel corpus created by Andrius Utka from the Center of Linguistic Corpus of Birmingham University constituted by European Parliament documents.

- **UtkaEuroParl** (UEP)
  The full version of the 6 million word English/French parallel corpus created by Andrius Utka.

- **EuroParl** (EP)
  A multilingual corpus for evaluation of machine translation(Koehn, 2002) available in pairs from English to other ten languages. The used pair was Portuguese/English which contains roughly 20 million words in 740,000 sentences per language

3.4.1 From Twente-Aligner to NATools

Although time comparisons between the original Twente-aligner and NATools were presented through section 3.2, table 3.2 presents them again for easier comparison. These statistics, from the already presented parallel Bible of 805 thousand words, were measured on an Intel Pentium IV, running at 1.7 GHz, 256 MB of RAM memory and running Linux.

	Twente	**NATools**
Corpus analysis	180 sec	4 sec
Matrix initial.	390 sec	21 sec
EM-Algorithm	2128 sec	270 sec

Table 3.2: Efficiency comparison between Twente-aligner and NATools implementations using a parallel Bible

[4]Some authors consider this kind of corpora comparable and not parallel. Although not a favorable example, this corpus produced very interesting results.

3.4.2 Scalability

Only the first five of the presented corpora list can be aligned in a single chunk using our work machine. The last on the list, **HUEP** was our test study for tests on the scalability limit. These five corpus sizes and times of the different alignment steps are shown on table 3.3.

	TS	**HP**	**IPC**	**Bib**	**HUEP**
K words	77	94	118	805	3 500
Corpus analysis	0.5 sec	1 sec	1 sec	5 sec	67 sec
Matrix initial.	6 sec	8 sec	4 sec	57 sec	893 sec
EM-Algorithm	42 sec	73 sec	44 sec	468 sec	5 523 sec

Table 3.3: Times comparison for five different corpora alignment on a single chunk

The evolution of the time required to analyze the corpus, create the sparse matrix and iterate over the matrix is shown on figure 3.8. We notice that each of these operations presents a different evolution pattern.

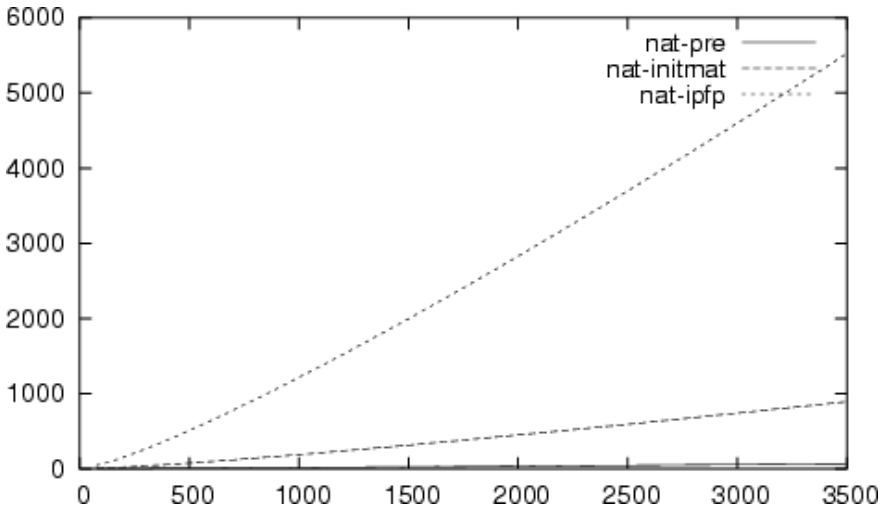

Figure 3.8: Timing evolution regarding corpus size. From top to bottom, EM-Algorithm, matrix initialization and corpus analysis

As explained before, the **UEP** corpus was the first corpus we could not align in only one chunk. For that reason, we aligned the two halves and computed some measures, which we present on table 3.4.

This table shows also the size of the matrices we need to allocate. Al-

though they are sparse, the size of each matrix was about 500MB.

	First half	Second half
K sentences	190	261
K EN words	3 295	3 798
K FR words	3 705	4 165
Matrix size ($rows \times columns$)	61 324 × 71 143	59 888 × 70 270
Corpus analysis	68 sec	71 sec
Matrix initialization	894 sec	731 sec
EM-Algorithm	5 524 sec	4 792 sec

Table 3.4: Measures for two halves of UEP corpus

Although this alignment was performed with about 190 thousand sentences, the machine used had 512 MB of RAM memory. With 256 MB of RAM we could not align more than 60 thousand sentences. That is the reason to use by default 50 thousand sentences as a limit for single chunk alignment. To rely on the sentence number instead of the number of words makes this process easier and faster.

Using the auto-splitter script 10 chunks of about 50 thousand sentences each were created.

Chunk	Analysis	Initialization	EM-Algorithm	Total
1	26 sec	175 sec	1231 sec	1432 sec
2	31 sec	181 sec	1099 sec	1311 sec
3	14 sec	242 sec	1286 sec	1542 sec
4	31 sec	222 sec	1067 sec	1320 sec
5	40 sec	276 sec	1473 sec	1789 sec
6	21 sec	247 sec	2007 sec	2275 sec
7	12 sec	180 sec	1138 sec	1330 sec
8	12 sec	156 sec	665 sec	833 sec
9	3 sec	14 sec	632 sec	649 sec
10	12 sec	2 sec	12 sec	26 sec
Total	202 sec	1695 sec	10610 sec	12507 sec

Table 3.5: Measures for the ten chunks of UEP corpus

Table 3.6 compares the times between a two or ten chunk alignment. As we can see the difference is very low: about 400 seconds in 12 000 seconds (3.3%). Although the time difference is not big, the difference on usability of the machine during the alignment is much better as a far lower amount

of memory is required. The time for the dictionary addition can be ignored given that it is very small.

	2 chunks	10 chunks
Corpus analysis	139 sec	202 sec
Matrix initialization	1 625 sec	1 695 sec
EM-Algorithm	10 316 sec	10 610 sec
Total	12 080 sec	12 507 sec

Table 3.6: Comparison between the times of alignment in two or ten chunks for the UEP corpus

Finally, the **EP** corpus was aligned automatically in 14 chunks. Table 3.7 shows the times for each chunk and marginal totals.

Chunk	Analysis	Initialization	EM-Algorithm	Total
1	45 sec	207 sec	1662 sec	1914 sec
2	17 sec	234 sec	1660 sec	1911 sec
3	22 sec	277 sec	1556 sec	1855 sec
4	18 sec	244 sec	1493 sec	1755 sec
5	18 sec	294 sec	1734 sec	2046 sec
6	18 sec	281 sec	1733 sec	2032 sec
7	65 sec	282 sec	1698 sec	2045 sec
8	70 sec	253 sec	1577 sec	1900 sec
9	66 sec	262 sec	1640 sec	1968 sec
10	17 sec	245 sec	1756 sec	2018 sec
11	20 sec	306 sec	1556 sec	1882 sec
12	18 sec	229 sec	1675 sec	1922 sec
13	17 sec	219 sec	1443 sec	1679 sec
14	6 sec	100 sec	444 sec	550 sec
Total	417 sec	3433 sec	21627 sec	25477 sec

Table 3.7: Measures for the fourteen chunks of EP corpus

When comparing table 3.5 with table 3.7, we notice that the times per chunk are very similar. This can be useful to estimate the required time based on the number of chunks.

Figure 3.9 shows a graphic similar to figure 3.8 but where we added information about these two large corpora. It is interesting to realize that the time requirements evolution is very similar to the alignmnet without chunk creation.

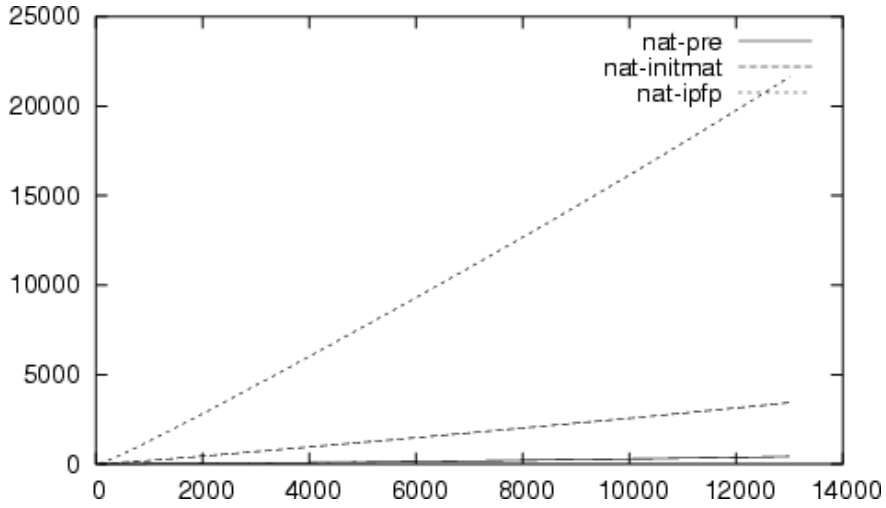

Figure 3.9: Timing evolution regarding corpus size, with chunk alignment. From top to bottom, EM-Algorithm, matrix initialization and corpus analysis

3.4.3 Disk usage measures

The alignment process creates lots of big files. In this section, we will see what is the relation between the corpus size and the disk usage for alignment.

From the created files, some of them are important, which correspond to data we want to use after alignment, while others are only temporary files, created to connect the various tools in a pipeline. There is a NATools option so that the aligner deletes temporary files as soon as they are not needed. This can be very useful in cases we do not have sufficient disk space. To maintain files in the disk makes a posterior alignment faster because only outdated files are recreated.

Corpus Analysis

The corpus encoding step creates three files per corpus: a lexicon file (`.lex`) a corpus file (`.crp`) and an index for the corpus file (`.crp.index`).

Table 3.8 shows disk usage for each one of these files for our study corpora. None of these files are temporary, and as such, cannot be deleted.

From this table we conclude that the lexicon file size evolution is not linear with the size of the corpus, because it is related with the number of different words on the corpus, and their size. The corpus index files are proportional to the number of sentences on each corpus. Because they are aligned, the

	text		lexicon		corpus		index	
	$\mathcal{L}_\alpha$	$\mathcal{L}_\beta$	$\mathcal{L}_\alpha$	$\mathcal{L}_\beta$	$\mathcal{L}_\alpha$	$\mathcal{L}_\beta$	$\mathcal{L}_\alpha$	$\mathcal{L}_\beta$
TS	536	394	123	147	150	132	25	25
HP	567	496	93	163	173	170	29	29
IPC	682	665	82	99	133	135	82	82
Bib	4 172	4 124	270	548	1 232	1 251	127	127
UEP	43 185	49 074	1 621	1 878	11 748	13 104	1 816	1 816
EP	105 414	113 852	1 125	1 698	31 272	34 060	2 664	2 664

Table 3.8: Disk usage (in KB) in corpus analysis step

number of sentences for both languages is the same.

The corpus files are compressed using `gzip`. Uncompressing on of the languages of the **EP** corpus, we get the size of 159 904 KB instead of the original 34 060 KB (about 500% of the original file).

Matrix allocation, iterative method and pre-dictionary

The matrix allocation process creates a sparse matrix and saves it on disk, then the iterative method reads that matrix file, iterates over it and saves it with another name. This means that the matrix file is temporary, but can be deleted only after we have the second matrix on disk: thus we need space to store simultaneously two matrices on disk. Table 3.9 shows disk usage for these files, for some of our corpora.

The table also presents the size of the pre-dictionary files, ready to be transformed in the two translation dictionaries.

	Original	Iterated	Pre-dictionary
TS	11 420	12 688	475
HP	10 963	12 318	416
IPC	8 323	9 088	216
Bib	55 651	61 643	1 251
UEP	$10 \times 90\,000$	$10 \times 98\,447$	$10 \times 1\,892$
EP	$14 \times 90\,000$	$14 \times 98\,447$	$14 \times 2\,089$

Table 3.9: Disk usage (in KB) for sparse matrices on disk

Dictionaries

Finally, the two dictionaries created are zipped. We will present the sizes for them in both formats. The unzipped size is very important because subsequent uses of the dictionary will load it directly to memory. This means that the unzipped size is the memory needed to load the dictionary. Table 3.10 shows these measures.

	Zipped		Unzipped	
	$\mathcal{L}_\alpha$	$\mathcal{L}_\beta$	$\mathcal{L}_\alpha$	$\mathcal{L}_\beta$
TS	150	155	528	599
HP	119	149	408	666
IPC	71	71	354	417
Bib	301	536	1 130	2 149
UEP	2 113	2 128	6 369	7 240
EP	1 792	3 383	4 540	6 561

Table 3.10: Disk usage (in KB) for the dictionaries files

Full disk usage

Table 3.11 shows how much disk space is needed for the full alignment, and the disk space really necessary, after deleting temporary files.

	All files	Needed files
TS	25 916	944
HP	27 972	956
IPC	19 424	820
Bib	128 552	4 436
UEP	1 722 701	36 054
EP		78 486

Table 3.11: Disk usage (in KB) for full alignment

3.5 Re-engineering Conclusions

The work performed to re-engineer Twente-Aligner into NATools was successful, not only because it became faster, but also more robust and able to align real-size parallel corpora.

The dictionary addition helped NATools to be able to scale-up and to provide ways to accumulate dictionaries, creating bigger and better ones. Also, given that the alignment of the chunks can be done independently, it is easy to add parallel processing.

To have the word-alignment dictionaries with probability estimations helps us to start new projects and apply new ideas. Being able to obtain these dictionaries in big quantities makes these projects more interesting. In chapter 5 some examples of applications using probability dictionaries are shown.

Chapter 4

Word Alignment Results

> *Although general purpose machine readable bilingual dictionaries are sometimes available, and although some methods for acquiring translation lexicons automatically from large corpora have been proposed, less attention has been paid to the problem of acquiring bilingual terminology specific to a domain, especially given domain-specific parallel corpora of only limited size.*
>
> *(Resnik and Melamed, 1997)*

This chapter presents NATools in terms of its alignment results(Tiedemann, 1997). Each section presents extracts from NATools dictionaries, where the alignment was done using different techniques.

First section shows results using the bare alignment. Following sections present different alignment processes:

- multi-word term extraction gluing words in tuples: joining two words with a character different from a space, the aligner will consider the full token as a single word, thus aligning word pairs with word pairs;
- domain specific term extraction using an algorithm presented at (Resnik and Melamed, 1997) based on the remotion of common usage lexicon from the generated dictionaries;
- iterate the alignment process, where on each iteration the words with high correlation values are removed from the input on the following iteration;

- perform a language specific pre-processing issue like the tokenization of English possessives or the use of a morphological analyzer to enhance relationship between verbs.

4.1 Simple alignment

The dictionaries created by the word aligner map words from one language to a set of words in the other language, together with translation probabilities.

Let us look to a simple extract from the Bible[1]. Table 4.1 shows the words proposed as translations for the Portuguese word "Deus" (God) and respective translations for the English word "God".

Deus	
God	0.86
(null)	0.04
God's	0.03
He	0.01
Yahweh	0.01
him	0.01

God	
Deus	1.00

Table 4.1: Extract of the alignment for the words "God" and "Deus." Extract from the Bible alignment for the words "God" and "Deus."

From this table, we can conclude:

- the translation in Portuguese for the English word "God" is, undoubtly, the word "Deus";
- the translation in English for the Portuguese word "Deus" is, almost surely, the "God" word;
- on 4% cases of alignment, the Portuguese word "Deus" is aligned with the *null* word. This means that on some sentences where the "Deus" word appears, the translation sentence does not include any word with high probability of translation. See table 4.2 first set of examples for some of these situations.
- the source corpus was not correctly tokenized. The genitive "God's" should be split on two different words: "God" and a symbol to represent

[1]Notice that this is an adverse corpora, because it consists of two parallel translations from a third language.

	Portuguese	**English**
1	Fechado em si mesmo, o homem não aceita o que vem do espírito de Deus.	The one who remains on the psychologic level does not understand the things of the Spirit.
	Eu, como bom arquitecto, lancei os alicerces conforme o dom que Deus me concedeu; outro constrói por cima do alicerce.	I, as a good architect, according to the capacity given to me, I laid the foundation, and another is to build upon it.
2	Depois os Levitas transportaram a Arca de Deus, apoiada em varais sobre os Ombros, conforme moisés lhes havia mandado, segundo a palavra de Deus.	and the Levites carried the ark of God with the poles on their shoulders, as Moses had ordered according to the command of Yahweh.
	"Vai e diz a David: 'Assim diz Javé: proponho-te três coisas. Escolhe uma, e Eu a executarei."	"Go and say to David, 'Yahweh says this: I offer you three things; choose one of them for me to do to you."
3	Mas Deus escolheu o que é loucura no mundo, para confundir os sábios; e Deus escolheu o que é fraqueza no mundo, para confundir o que é forte.	Yet God has chosen what the world considers foolish, to shame the wise; He has chosen what the world considers weak to shame the strong.
	Se dizemos que estamos em comunhão com Deus e no entanto andamos nas trevas, somos mentirosos e não praticamos a Verdade.	If we say we are in fellowship with Him, while we walk in darkness, we lie instead of being in truth.

Table 4.2: Examples of alignment for "Deus" and "God"

the genitive construction. If this was done, the probability for the correct translation would be enhanced.

- "Yahweh," is translated in some of the Bible books as "Javé" and in others as "Deus", while the English Bible is more consistent. The second set of examples from table 4.2 shows some examples.

- finally, personnel pronouns like "He" and "Him" appear given the the use of the pronoun instead of the real noun, as shown on the third set of examples from table 4.2.

Let us see another alignment extract, again from the parallel Bible. Table 4.3 shows the dictionary for the verb "gostar/to love." This is a more difficult situation for the aligner than the previous example.

gosta	
loves	0.50
pleases	0.34
pleasing	0.17

loves	
ama	0.67
gosta	0.08
amas	0.05
estima	0.03
conquista	0.02
acabará	0.02
curta	0.02

Table 4.3: Extract from the Bible alignment for the "gosta" and "loves" words

This example is not as good as the previous one, but is surely more interesting:

- the most used Portuguese translations for "loves" is "ama" from the "amar" verb, or "gosta" (which is translated as "like", too). This can be seen in the other column of the table: "gosta" is being translated as "loves" and "likes" (first set of examples on table 4.4).

- while "estima" can be used in some contexts as a translation of "loves", the other words, "conquista", "acabará" and "curta" does not seem to be translations of "loves" at all. As it can be seen on the second set of examples from table 4.4, these weird alignments are interesting cases for linguistic studies.

	Portuguese	English
1	Se alguém ama o mundo, o amor do Pai não está nele.	If anyone loves the world, the love of the Father is not in him.
	quem gosta de vinho e carne boa jamais ficará rico.	he who loves wine and perfume will never grow rich.
	Javé detesta balanças falsas e gosta de peso justo.	Yahweh detests a false scale but a just weight pleases him.
2	porque ele estima o nosso povo e até nos construiu uma sinagoga.	for he loves our people and even built a synagogue for us.
	Javé detesta a boca mentirosa, mas o homem sincero conquista o seu favor.	Yahweh hates the lips of liars and loves those who speak the truth.

Table 4.4: Examples of alignment for "gosta" and "loves"

4.2 Simple Multi-word term alignment

Given the efficiency boost in the alignment process, some new tests became feasible. In this section we will talk about a simple bilingual multi-word term extraction method. It was not thought as a real and fundamental method for bilingual multi-word term extraction, but as an experiment which gave some interesting results. A more realistic method for bilingual multi-word term extraction will be shown on section 5.6.

This simple method is based on the junction of words from the corpus into n-tuples. For example, for $n = 2$ we will join words into pairs:

```
1 Flitwick told me in secret that I got a
2 hundred and twelve percent on his exam .
```

```
1 BEGIN_Flitwick Flitwick_told told_me me_in
2 in_secret secret_that that_I I_got got_a
3 a_hundred hundred_and and_twelve twelve_percent
4 percent_on on_his his_exam exam_. ._END
```

This new corpus will have many more different words than the original one, which means that the alignment process will create a bigger matrix. This matrix will count co-occurrences for word pairs, instead of co-occurrences for simple words.

Table 4.5 shows a good result example for this alignment process using the Bible corpus, with the "Christ Jesus" term. Follows some considerations about this alignment:

Jesus Cristo	
Christ Jesus	0.67
Jesus Christ	0.26
(null)	0.03
Messiah ,	0.01
Christ who	0.01
the Messiah	0.01

Christ Jesus	
Jesus Cristo	0.94
(null)	0.04
Cristo ,	0.01

Table 4.5: Extract from the word-pairs alignment result for "Christ Jesus" and "Jesus Cristo."

- in the English Bible, "Jesus Cristo" is translated as "Christ Jesus" but in some books also as "Jesus Christ";
- in English, "Messiah" is used to translate "Jesus Cristo" while in Portuguese the probable translation ("Messias") is rarely used;

Table 4.7 shows another example of alignment, also from the Bible, for the adverbial phrase "a little". This is a more difficult and more interesting alignment:

- the correct (most common) translation is given with more than 50% of probability;
- there is an huge number of alignments with the *null* token, probably given that the Bibles are not direct translations and as such the original translators can ignore or add sometimes this adverbial phrase without changing the meaning. The first set of examples from table 4.6 shows some of these cases.
- there are some weird alignments as "me a", "your company", "e ," or "BEGIN Daqui", but with very small probabilities.
- some other alignments can be explained by different uses of the term (extracts from the corpus can be seen on table 4.6, second set of examples).

Bigger groups (three or more words) were tried but results were not really interesting: process time raised a lot, accordingly with the number of words we put together; the alignment matrix became too huge to be usable; multi-words were not found or, when found, on little quantities.

	Portuguese	English
1	David ficou de pé e tomou a palavra: "irmãos e povo meu, escutai-me um pouco."	Then King David got up and said: "My brothers and my people, listen to me."
	Vai e faz como disseste. Mas primeiro prepara um pãozinho com o que tens e traz-mo.	Go and do as you have said, but first make me a little cake of it and bring it to me;
2	Um pouco mais adiante, lançaram de novo a sonda e deu vinte e sete metros.	After a while, they measured it again and it was twenty-seven meters.
	Pouco depois, outro viu Pedro e disse: "Tu também és um deles."	A little later someone who saw him said, "You are also one of them!"
	Jesus chamou uma criança, colocou-a no meio deles,	Then, Jesus called a little child, set the child in the midst of the disciples,

Table 4.6: Examples of alignment for "um pouco" and "a little"

um pouco		a little	
a little	0.68	um pouco	0.54
(null)	0.19	(null)	0.27
a while	0.03	Pouco depois	0.06
me a	0.03	e ,	0.03
your company	0.02	uma criança	0.03
BEGIN Then	0.01	BEGIN Daqui	0.02

Table 4.7: Extract from the word-pairs alignment for "a little" and "um pouco" word pairs.

4.3 Simple Domain Specific Terminology Extraction

Word alignment of parallel corpora is a method to extract bilingual terminology and bootstrap dictionaries. Meanwhile, if the idea is to build a domain specific dictionary, the distinction of common lexicon from the domain specific one must be performed. On (Resnik and Melamed, 1997) is proposed a simple algorithm for a domain specific lexicon acquisition process:

1. *Run the automatic lexicon acquisition algorithm on a domain-*

specific parallel corpus.

2. *Automatically filter out "general usage" entries that already appear in a machine readable dictionary (MRD) or other general usage lexical resources.*

3. *Manually filter out incorrect or irrelevant entries from the remaining list.*

This algorithm was applied to part of the European Parliament (EP) corpus where "general usage" entries were removed using Jspell morphological analyzer dictionary(Almeida and Pinto, 1994).

Although the results from this process contain some interesting results (see table 4.8), more interesting ones could be obtained if we used a controlled list of words instead of a general morphological analyzer dictionary. Other reason for the low amount of results is that the EP corpus are transcriptions from the sessions of the Parliament, where a very general (and oral) language is used.

Other entries that remained in the dictionary were misspells, numbers, personal names and non-translated words. Some of them could be removed using automatically.

#	Word	Translation	Probability
10	Sabóia	Savoy	0.3825
		allowed	0.0182
7	estrategicamente	strategically	0.5762
		form	0.2128
		strategic	0.0936
8	multifuncionais	multifunctional	0.4010
17	unicef	unicef	0.8146
		bought	0.1498
4	geostratégicas	geostrateic	0.4010
3	intergovernamentalismo	intergovernmentalism	0.2951
		things	0.2158
4	democratas	extremists[2]	0.1757
		neo	0.1160
10	furanos	furanes	0.4006

Table 4.8: Extract from the result of the domain specific terminology extraction method.

4.4 Iterative Alignment

In (Ahrenberg, Andersson, and Merkel, 2000) an iterative method for word alignment is presented, based on the basic alignment method, but iterating the alignment over the corpus.

The method starts with a first alignment. Then, the dictionary is analyzed: words with high values of relationship are saved and removed from the original corpus. This new (smaller) corpus is re-aligned and this process iterated. The algorithm iterates until the source corpus becomes empty.

Our tests were done iterating until the dictionary had only one relationship with probability over a specific threshold (in the test presented here, 70% of translation quality). For the Tom Sawyer (TS) corpus, this process iterated 8 times.

Iter	# entries	quality
1	459	72%
2	120	54%
3	51	54%
4	23	75%
5	7	40%
6	5	40%
7	4	75%
8	4	50%
9	1	0%
	674	

Table 4.9: Dictionaries sizes and qualities for the Iterative Alignment

Table 4.9 shows the number of entries in the dictionary with quality higher than 70% for each iteration. The quality presented is the percentage of good translations on this dictionary. This value was obtained analyzing by hand the first 50 entries of each iteration dictionary.

As it could be seen in table 4.9, if we used the same threshold on the source corpus, we would get a dictionary of 459 elements, with about 128 bad translations. Using the iterative method we would obtain a 674 element dictionary, with about 224 incorrect translation. Then, with only one iteration we would get 331 good entries, while we get 450 with the iterative method.

[2]This is one example to study given the strange relationship.

4.5 Using Language specific pre-processing

As presented on the previous chapter, the basic alignment process is language independent because there is a first language dependent layer (which we named the pre-processing step). This layer can be used to perform operations on the source corpus to obtain more interesting results with the alignment process.

This section will present two different ways to pre-process corpora and a discussion on the obtained results.

4.5.1 English Genitives

As shown on table 4.1, there was an alignment between "Deus" and "God's". This is very common when aligning English texts and the tokenization process was not done correctly. One simple solution would be to split "God's" into three tokens: "God", "" and "s". This is not the correct way: we should use "God" and "'s". This one is better because we know that the apostrophe is part of the genitive. Meanwhile, this is not the best solution. Cases like "Jonas' boat" should not be split into "Jonas", "" and "boat", but into "Jonas" "'s" "boat". This would lead to better word alignment results.

In any case, we can always substitute "'s" by another token like "_GEN_" if we want it to be easy to find. This means that a sentence like

```
1 | It clearly is not , as Mr Berthu has said , in order to
2 | create a European super - state but more , as Mrs Berès
3 | has said , to make it clear to today's citizens and
4 | tomorrow's citizens that what we are in now is a Community
5 | of values : values based on democracy , freedom , equality ,
6 | solidarity and respect for diversity , values -- and I
7 | stress this to Mrs Garaud -- that unite Europeans across
8 | frontiers , from the north , the south , the east and the
9 | west of the Community .
```

could be substituted by

```
1 | It clearly is not , as Mr Berthu has said , in order to
2 | create a European super - state but more , as Mrs Berès
3 | has said , to make it clear to today 's citizens and
4 | tomorrow 's citizens that what we are in now is a Community
5 | of values : values based on democracy , freedom , equality ,
6 | solidarity and respect for diversity , values -- and I
7 | stress this to Mrs Garaud -- that unite Europeans across
```

```
8 | frontiers , from the north , the south , the east and the
9 | west of the Community .
```

This kind of input will not consider "tomorrow's" and "today's" as different words from "tomorrow" and "today." Also, "'s" will be considered as a different word, and, as such, will be aligned.

For example, for the European Parliament (EP) corpus this tokenized form will produce the results shown on table 4.10.

's	
0.37	da
0.22	do
0.08	a
0.04	de
0.03	o

Table 4.10: Results of alignment using correctly tokenized genitives

4.5.2 Infinitive verbs

The alignment between English and Portuguese verbs has a big problem: Portuguese has many more forms for each verb. Thus, the relations between the English few forms with the huge number of Portuguese forms[3] is very fragile (while in the opposite direction this is not noticed).

To solve this problem we used a morphological analyzer (Jspell (Simões and Almeida, 2001; Almeida and Pinto, 1994)) with a Portuguese dictionary, to stem Portuguese verbs.

For example, the following input

```
1 | Senhora Presidente , gostaria de saber se esta semana
2 | o Parlamento terá oportunidade de manifestar a sua inequívoca
3 | posição de descontentamento face à decisão , hoje tomada , de
4 | não renovar o embargo de armas destinadas à Indonésia , tendo
5 | em atenção que a grande maioria da assembleia apoiou o referido
6 | embargo quando este foi decretado .
```

would be substituted with[4]

```
1 | Senhora Presidente , gostar de saber se esta semana
2 | o Parlamento ter oportunidade de manifestar a sua inequívoca
```

[3]Normally, English has 4 form, while there are about 50 for Portuguese.

[4]Yes, this is the kind of text Tarzan would say!

```
3 | posição de descontentamento face à decisão , hoje tomar , de
4 | não renovar o embargo de armas destinar à Indonésia , tendo
5 | em atenção que a grande maioria da assembleia apoiar o referir
6 | embargo quando este foi decretar .
```

These changes on the source corpus will enhance the probability of translation for verbs from English to Portuguese.

While the probability between English to Portuguese translation raises, the inverse decreases. This happens because the occurrence number of the Portuguese verb becomes too high in relation to the different (although few) English verb forms.

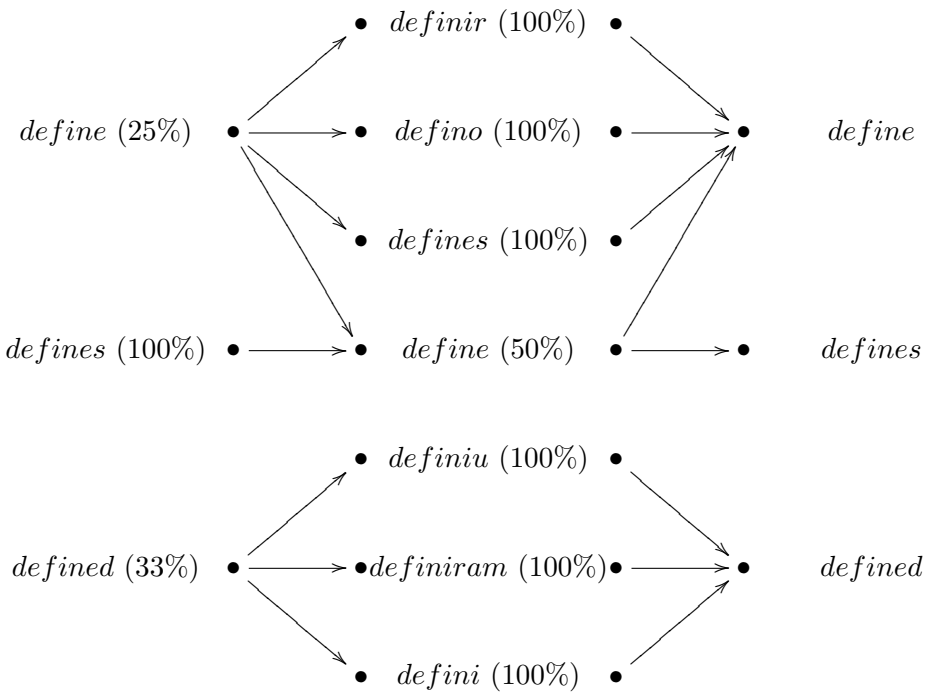

Figure 4.1: Alignment before Portuguese verb stemming

Figure 4.1 shows an alignment for the verb "definir/to define" and some of their forms[5].

[5]Probabilities in the diagram are not real. They were added without knowledge of possible occurrences counts. This mean they were calculated using $100/n$ where n is the number of relations which source is that word.

So, before the Portuguese verb stemming, we notice two distinct blocks, one for the past tense, and another with imperative, infinitive and present. Relation between present and past tense from English to Portuguese is very low (below 33%) although relation between Portuguese and English is very high for almost all forms.

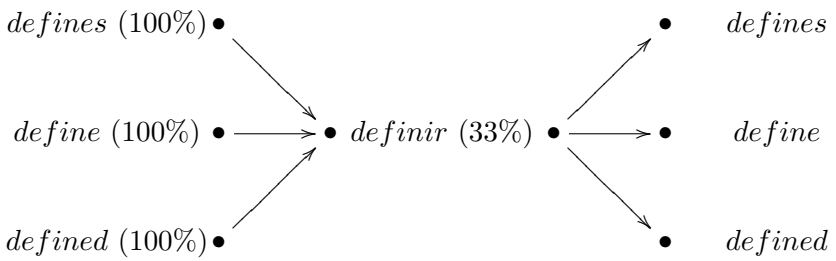

Figure 4.2: Alignment after Portuguese verb stemming

Figure 4.2 shows the same relationship after Portuguese verb stemming. At this point there is no distinct blocks, and probabilities have changed completely. English to Portuguese relationship becomes stronger, and relationship from Portuguese to English weakens a lot.

This is a good way to get better relationships between verbs from English to Portuguese. If we want to enhance relationships in both directions we have two choices: to stem verbs in both languages (as figure 4.3) or to do a selective stemming to different tenses.

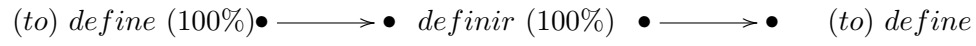

Figure 4.3: Alignment after Portuguese and English verb stemming

This last option should retrieve better results. For example[6], if we take care only of the past tense, and stem Portuguese and English verbs to the infinitive unless they are in the past tense and, in this case, "stem" them to a common past tense format, we obtain a relation as shown on figure 4.4[7].

[6]This example it very naive, given that there are more than one past tense, and the process should consider each tense separately.

[7]In figure 4.4 we used `/p` to denote the past tense.

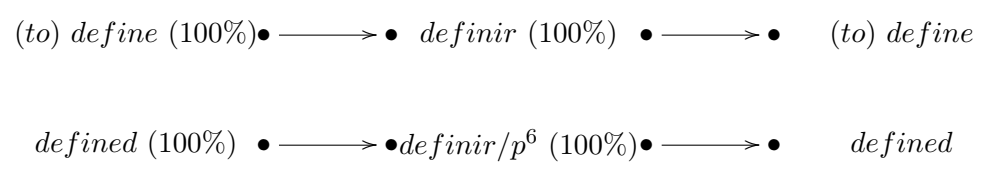

Figure 4.4: Alignment after Portuguese and English special verb stemming

4.6 Results Analysis

This chapter shows that the NATools alignment results are quite reliable, and that different ways of using NATools are available. Each one of these experiments gave interesting results leading to new ideas and new experiments.

Although some of the methods return some weird entries in the dictionaries, the true is that most of them can be explained by the corpus we were using. On these and other cases, these bad translations are interesting for linguistic and cognitive studies.

Finally, most of these experiments were possible given the pre-processing layer added to the original Twente-Aligner. This layer is very important giving the user the ability to transform the corpus and tune the way the alignment is done.

Chapter 5

NATools based Tools

A successful [software] tool is one that was used to do something undreamed of by its author.

S. C. Johnson

The NATools alignment results in two different objects: probabilistic dictionaries and parallel corpora. In this chapter, we will show that these objects are not the end of the road, but the beginning of several new problems(Tiedemann, 2003).

The possibility to use NATools probabilistic dictionaries as a resource for new tools and experiments is very important, and that is the reason why we think NATools probabilistic dictionaries as a data type with an associated algebra.

This chapter demonstrates how these objects can be used to perform various tasks:

- word search on parallel corpora
  use the sentence aligned text to compare sentences in different languages, using the web as interface;
- bilingual dictionary browsing
  provides ways to search words in the probabilistic dictionary, using a graphical interface with meaningful colors to show the results. The interface can be used to check where the word occurs in the corpora, and why it was aligned with some specific word;
- estimation of translation probability for parallel sentences

use the dictionary probabilities to look to a pair of string and give them a classification regarding their translation quality;

- word sequence alignment
  use the word aligned corpora and probabilistic dictionaries to build aligners for segment of words;

- multi-word term extraction
  use a multi-word term extractor for monolingual corpus and align the terms, extracting a multi-word term bilingual terminology;

- translation by example
  use the parallel corpora to search for examples of translations for a specific sentence, automating the translation;

Before explaining each of these items in detail, next section will explain the programmers interface (API) for writing programs based on NATools dictionaries.

5.1 NATools Programming Interface

NATools probabilistic Dictionaries can be used in different tools, it is crucial to build a robust API to simplify their manipulation.

The NATools alignment process creates dictionaries and store them on four files: two lexicon files including information about the corpus' words, and two probabilistic dictionary files, including alignment information in both directions (source to target and target to source languages).

$$NATDic(\mathcal{L}_\alpha, \mathcal{L}_\beta) = Lex_\alpha \times Lex_\beta \times \mathcal{D}(\mathcal{L}_\alpha, \mathcal{L}_\beta) \times \mathcal{D}(\mathcal{L}_\beta, \mathcal{L}_\alpha)$$

At the moment, a new dictionary file type is being developed to include in one unique file the two lexicons and the two probability dictionaries.

The interface with these files can be done with Perl or C programming languages. These interfaces are quite similar and, given that Perl use is simpler, we will show only its interface. In fact, the Perl interface is written over the C interface using the Perl facility named XS[1].

Next sections explain briefly how to use the NATools programming interface. First two sections show the API for each lexicon and dictionaries

[1] XS is an interface description file format used to create an extension interface between Perl and C code (or a C library) which one wishes to use with Perl.

access, and the last includes some examples of how to combine them to write programs.

5.1.1 Lexicon files' interface

Each lexicon files includes the words used in one of the corpora (source or target one), as defined on section 3.2.2:

$$Lex_\alpha = w_\alpha \rightharpoonup (id \times occurrenceCount)$$

The interface with Lexicon files is done using the `NAT::Lexicon` Perl module. This module provides the following API[2]:

- `$lex = NAT::Lexicon::open("file.lex")`
  Constructor for a `NAT::Lexicon` object, given one lexicon file. At the moment the Perl interface can open up to ten lexicon files at the same time. This limit will be removed on next releases.

- `$lex->close()`
  Closes the lexicon object and frees the used memory.

The main purpose of the lexicon file is to map words to identifiers and vice-versa. Following methods provides ways to access identifiers from words and vice-versa.

- `$lex->word_from_id(2)`
  This method returns the word whose identifier is passed as argument.

- `$lex->id_from_word("dog")`
  This is the dual method of `word_from_id`. Given a word from the lexicon file, returns its identifier;

- `$lex->sentence_to_ids("the dog bitten the lazy cat")`
  It is very common to have to translate each word from a sentence to identifiers. This method takes care of that, returning a list of identifiers;

- `$lex->ids_to_sentence(1, 3, 5, 3, 9, 5, 3)`
  Given a list of identifiers, this method returns the decoded sentence;

[2]Given that the occurrence count information is also available on the dictionary probability files, there is not a method to access to these value on this module. It could be added later if needed.

5.1.2 Dictionary mapping files interface

Each dictionary map file include the relationship between words from one language into another and its structure is, as defined on section 3.2.5:

$$\mathcal{D}(\mathcal{L}_\alpha, \mathcal{L}_\beta) = w_\alpha \rightharpoonup (occur \times w_\beta \rightharpoonup P\left(\mathcal{T}\left(w_\alpha\right) = w_\beta\right))$$

The interface to this file is done using the `NAT::Dict` Perl module. This module provides the following API[3]:

- `$dic = NAT::Dict::open("file.bin")`
  This is the constructor of `NAT::Dict` objects. It receives a filename of a dictionary file and returns a reference to an object. As with the Lexicon files, at the moment there can be up to 10 dictionary objects open at the same time.

- `$dic->close()`
  Closes the object, freeing the used memory.

- `$dic->exists($id)`
  This method is used to check if an identifier exists in the probabilistic dictionary. Although very uncommon, sometimes the alignment process discards some words.

- `$dic->size()`
  For some tasks it is important to know the size of the original corpus. This method returns that value, summing up the occurrence count for all corpus words.

- `$dic->occ($id)`
  Returns the number of occurrences for a specific word.

- `$dic->vals($id)`
  This is the method to access the translations for a specific word. It returns a map from word identifiers (from the target corpus) to its respective probability of translation.

- `$dic->add($dic2)`
  When the alignment is done by chunk the final dictionaries need to be summed up. This method is used for sum dictionary chunks. This method can not be used to add any pair of dictionaries, because the identifiers on each dictionary should represent the same word.

[3]Note that all the methods on this module use identifiers instead of the real words.

- `$dic->for_each( sub{...} )`
  This is an utility method, very powerful to process each word from a dictionary. It will cycle through all dictionary words and for each one of them call a given function, passed as argument. See example on page 85 for an example of use.

5.1.3 Programming with NATools dictionaries

This section shows some examples of programs using the NATools programming interface.

An interactive dictionary shell

For studying a probabilistic dictionary it is useful to have the ability to search for words instead of editing the dictionary with a word editor. With that in mind, the following example was created: a shell which reads words and prints all the information it knows about that word translations:

```
1  #!/usr/bin/perl -w

2  use NAT::Dict;
3  use NAT::Lexicon;

4  my $dic1 = NAT::Dict::open(shift());
5  my $lex1 = NAT::Lexicon::open(shift());

6  my $dic2 = NAT::Dict::open(shift());
7  my $lex2 = NAT::Lexicon::open(shift());

8  while(1) {
9    print "Word to search ::> ";
10   chomp(my $word = <>);
11   show_word($dic1,$lex1,$lex2,$word);
12   show_word($dic2,$lex2,$lex1,$word);
13 }

14 sub show_word {
15   my ($dic1,$lex1,$lex2,$word) = @_;
16   my $wid = $lex1->id_from_word($word);
17   if ($wid) {
18     my $count = $dic1->occ($wid);
19     my %probs = @{$dic1->vals($wid)};
20     print " Occurrence count: $count\n";
```

```
21      print " Translations:\n";
22      for (sort {$probs{$b} <=> $probs{$a}} keys %probs) {
23        next unless $_;
24        my $trans = $lex2->word_from_id($_);
25        print "  $trans => $probs{$_}\n";
26      }
27    }
28  }
```

This script starts by including the necessary modules and opening the lexicon and dictionary files. Then, it starts a loop that reads words and, if found ine one of the two dictionaries, a brief list of their translations is printed:

```
1   [ambs@holst test]$ ./shell Tom-PT.bin Tom-PT.lex \
2                              Tom-EN.bin Tom-EN.lex

3   Word to search ::> cão

4    Occurrence count: 17
5    Translations:
6     dog => 0.645013034343719
7     stray => 0.189520448446274
8     poodle => 0.133752107620239
9     floated => 0.0306419488042593

10  Word to search ::> gato

11   Occurrence count: 22
12   Translations:
13    cat => 0.909570515155792
14    cats => 0.0532888397574425
15    ripple => 0.0270212069153786
16    troubles => 0.0100118424743414

17  Word to search ::> casa

18   Occurrence count: 115
19   Translations:
20    house => 0.395296275615692
21    home => 0.293731182813644
22    (null) => 0.120877973735332
23    homeward => 0.0427895896136761

24  Word to search ::>
```

Creating a bilingual dictionary

The following example shows how to extract a bilingual dictionary from the probabilistic one: find words w_α and w_β such that $w_\beta \in \mathcal{T}(w_\alpha)$ and $w_\alpha \in \mathcal{T}(w_\beta)$ and above a given probability threshold.

The function receives two lexicon filenames and two probabilistic dictionary filenames. It returns a list of pairs (w_α, w_β).

```
1  sub calc_dic {
2    my ($dic1, $lex1, $dic2, $lex2) = @_;

3    $dic1 = NAT::Dict::open($dic1);
4    $lex1 = NAT::Lexicon::open($lex1);

5    $dic2 = NAT::Dict::open($dic2);
6    $lex2 = NAT::Lexicon::open($lex2);

7    my @DIC;
8    my $THRESHOLD = 0.7;

9    $dic1->for_each( sub {
10     my %data = @_;

11     my ($w1,$prob1) = @{$data{vals}};
12     return unless $prob1 >= $THRESHOLD;

13     my ($w2,$prob2) = @{$dic2->vals($w1)};
14     return unless $prob2 >= $THRESHOLD;

15     if ($w2 == $data{word}) {
16       push @DIC, [$lex1->word_from_id($data{word}),
17                   $lex2->word_from_id($w1)];
18     }
19   } );

20   return \@DIC;
21 }
```

Basically, the first set of lines opens the dictionary and the lexicon files, defines the variable to store the result, and define a threshold.

Follows a loop, iterating each word (w) on the source language dictionary. For each word, we get the translation (w_1) with bigger probability ($prob_1$). If its probability is bellow the threshold, we iterate to the next word. If its probability is above, we apply the same algorithm to the translation of w_1

(w_2). If both translation probabilities are above the threshold, we add them to the dictionary array.

5.2 Word search on Parallel Corpora

To consult parallel corpora in the web is a common task. There are several Internet sites where you can search parallel corpora(Frankenberg-Garcia and Santos, 2001).

The goal of this tool is to make NATools users able to publish their aligned corpus in the Internet with search capabilities.

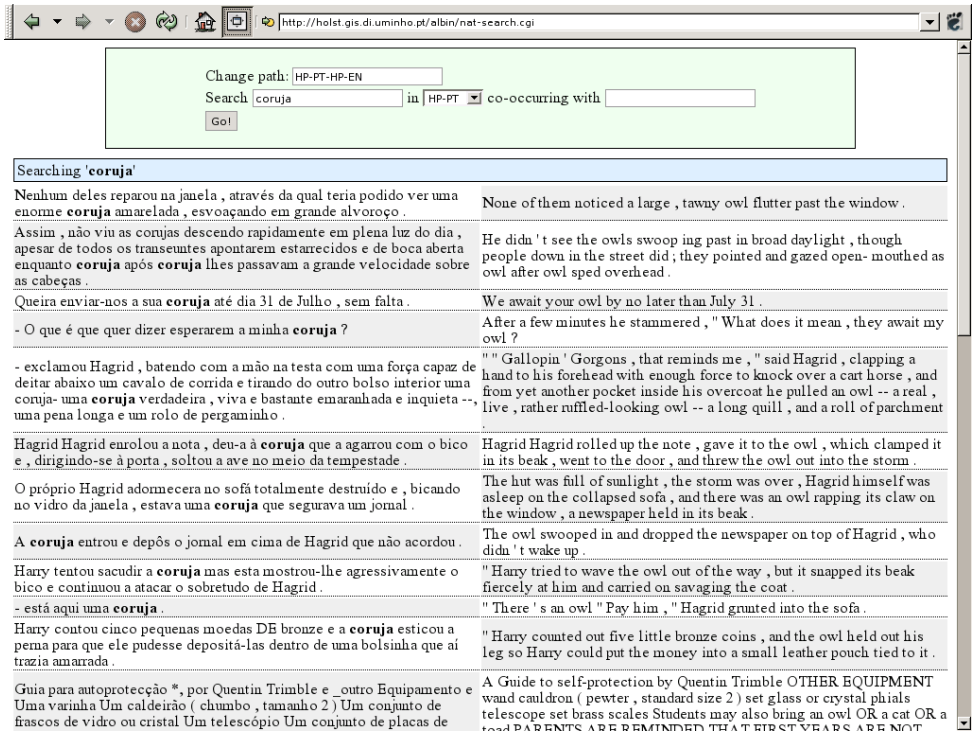

Figure 5.1: Searching for a word on a parallel corpus

The tool consists of a simple Perl CGI using NATools modules which search pairs of sentences where some word occur. It uses the lexicon and encoded corpora files only, which means you do not need to do all the alignment process to make the corpora search able:

$$(Lex_\alpha \times Crp_\alpha) \times (Lex_\beta \times Crp_\beta) \times (w_\alpha + w_\beta) \longrightarrow (s_\alpha \times s_\beta)^\star$$

Figure 5.1 is a screen-shot of this CGI script. The user is able to choose the corpus he wants to navigate, and search for a word (or sequence of words) in any of the two aligned languages.

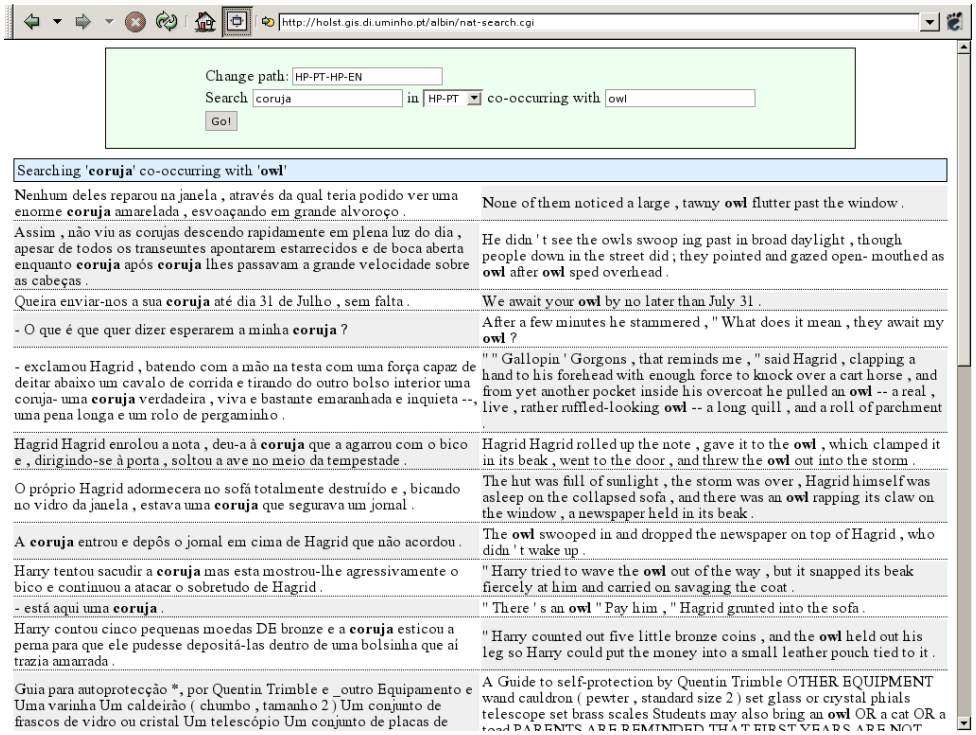

Figure 5.2: Searching for a words and probable translation on a parallel corpus

As shown on figure 5.2 it is possible to give a possible translation for the search query. This makes the CGI able to search for pairs of sentences where the word we are searching co-occurs with that possible translation:

$$(Lex_\alpha \times Crp_\alpha) \times (Lex_\beta \times Crp_\beta) \times (w_\alpha \times w_\beta) \longrightarrow (s_\alpha \times s_\beta)^\star$$

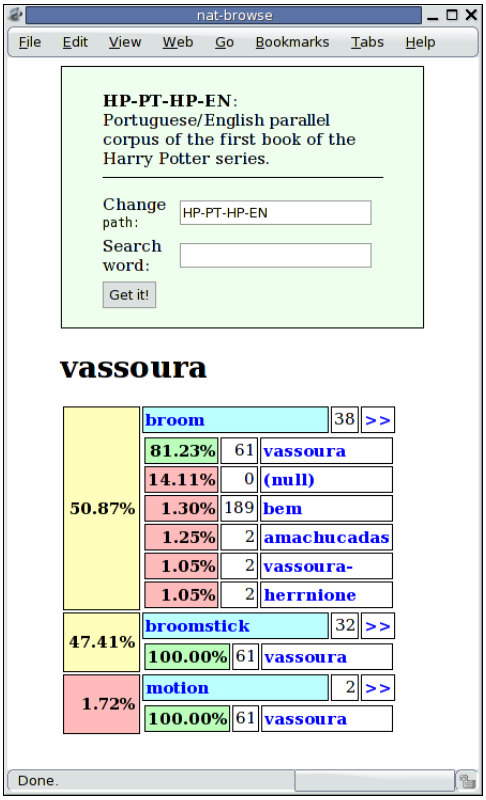

Figure 5.3: Dictionary navigation CGI

5.3 Dictionary browsing

This tool is a browser of dictionaries, very useful to study them, and to check their translations. Basically, it shows an entry from the dictionaries at two levels: the probable translations for the word that the user searched, and probable translations for those translations.

Figure 5.3[4] shows a screen-shot for this navigation CGI where the entry for the Portuguese word "vassoura" (broom) is presented (on the HP parallel corpus).

This CGI has the following properties:

- for each word, translations are sorted by the translation probability,

[4]This example shows two weird entries. The first, "vassoura-" was not tokenized. Then, "herrnione" seems an OCR problem (it should be Hermione, the name of Harry Potter friend).

and with different color levels (green, yellow and red). This makes it easier to visually identify the most probable translations;

- word search is done on both languages which means that if some word appears on both languages, the CGI will show one entry for each language;
- the script shows two levels of the dictionary: for each possible translation of the searched word the CGI shows their possible translations;
- if the word being analyzed appear as a possible translation for any of their possible translations, then that possible translation is presented on a different color;
- it is possible to follow a link ($\boxed{>>}$) directly to the CGI presented on previous section, to search the corpus for pairs of sentences where the word and one of the possible translations appear. Integration between different CGIs make each one of them more interesting.

5.4 Estimation os Translation Probability

The translation quality is a very hard property to measure(Santos, Maia, and Sarmento, 2004). Meanwhile, we can use the probabilistic dictionaries to calculate an estimation of translation probability. This process cannot be seen as the evaluation of the correctness of the translation, but a probabilistic measure.

Our estimative function ($\mathcal{Q}_{\mathcal{T}}$) between two languages $\mathcal{L}_\alpha$ and $\mathcal{L}_\beta$ can be prototyped as:

$$\mathcal{Q}_{\mathcal{T}} : \mathcal{D}(\mathcal{L}_\alpha, \mathcal{L}_\beta) \times \mathcal{D}(\mathcal{L}_\beta, \mathcal{L}_\alpha) \times s_\alpha \times s_\beta \longrightarrow \mathcal{Q}$$

To define our *translation probability measure* we need to define first the *unidirectional translation probability.*

Definition 8 *The* ***unidirectional translation probability*** *will be represented by* $\mathcal{P}\left(\mathcal{T}\left(s_\alpha\right) = s_\beta\right)$ *for two different sentences;* s_α *and* s_β*.*

Each sentence is a sequence of words. Then,

$$s_\alpha = w_{\alpha,1} w_{\alpha,2} \ldots w_{\alpha,n} \quad \textit{and} \quad s_\beta = w_{\beta,1} w_{\beta,2} \ldots w_{\beta,m}$$

On following formulas, consider $\mathcal{D}$ *as being the dictionary* $\mathcal{D}(\mathcal{L}_\alpha, \mathcal{L}_\beta)$*. This means that for a word* w_α*,* $\pi_2(\mathcal{D}(w_\alpha))(w_\beta)$ *is the probability of* w_β *being a translation of* w_α*.*

Thus, we can define the probability of s_β being a translation of s_α as

$$\frac{1}{n}\sum_{i=1}^{n} max\left(\{\pi_2(\mathcal{D}(w_{\alpha,i}))(w_{\beta,j}) : w_{\beta,j} \in dom(\pi_2(\mathcal{D}(w_{\alpha,i}))) \wedge w_{\beta,j} \in s_\beta\}\right)$$

◇

Definition 9 *Given two sentences s_α and s_β, the **translation probability measure** $\mathcal{Q}_\mathcal{T}$ is computed using the probability of s_α being a translation of s_β and vice-versa: $\mathcal{P}\left(\mathcal{T}\left(s_\alpha\right) = s_\beta\right)$ and $\mathcal{P}\left(\mathcal{T}\left(s_\beta\right) = s_\alpha\right)$.*

$$\mathcal{Q}_\mathcal{T}\left(s_\alpha, s_\beta\right) = \frac{\mathcal{P}\left(\mathcal{T}\left(s_\alpha\right) = s_\beta\right) + \mathcal{P}\left(\mathcal{T}\left(s_\beta\right) = s_\alpha\right)}{2}.$$

◇

$\mathcal{Q}$	English	Portuguese
0.261751	vocês os dois - eu mando-vos uma coruja .	Thanks , said Harry , I ' ll need something to look forward to .
0.673549	" Have you got your own broom ?	- Tens a tua vassoura ?
0.659810	" Harry glanced down at his broom .	Harry deitou um olhar à sua vassoura .
0.599914	" Stick out your right hand over your broom , " called Madam Hooch at the front , " and say ' Up !	- Estendam a vossa mão direita ao longo da vassoura –, gritou Madame Hooch ,- e digam De pé .

Table 5.1: Translation probability measure samples

Table 5.1[5] shows some examples of translation probability measures. This results can be used for very different purposes:

- classify, validate and test candidate parallel pairs in the automatic extraction of parallel corpora presented on section 2.2.2;
- classify automatically generated TMX files, such that translators can use that knowledge to reject some translations.

[5]Notice that the worst translation has 26% of probability given that the current algorithm is using punctuation as words. So, for small sentences like the on in the example, with lot of punctuation, values are not so low as expected.

- classify sub-sentences to make word sequence alignment, statistical translation and multi-word term translation dictionaries extraction.

As some of the developed tools need information about translation probability of aligned sentences, it was developed a tool to create a file with this information for each pair of sentences.

This tool signature can be described as:

$$(Crp_\alpha \times Lex_\alpha) \times (Crp_\beta \times Lex_\beta) \times \mathcal{D}(\mathcal{L}_\alpha, \mathcal{L}_\beta) \times \mathcal{D}(\mathcal{L}_\beta, \mathcal{L}_\alpha) \longrightarrow \mathcal{Q}^\star$$

5.5 Word sequence alignment

Word sequence alignment is deeply related with the translation probability measure proposed on previous section. With this word sequence alignment we do not expect to extract relationships between all word sequences in the corpus, but instead, we want to have an algorithm to align a small sequence of words.

This means we want a tool with the following signature[6]:

$$ws - align : (C_\alpha \times L_\alpha) \times (C_\beta \times L_\beta) \times \mathcal{D}(\mathcal{L}_\alpha, \mathcal{L}_\beta) \times \mathcal{D}(\mathcal{L}_\beta, \mathcal{L}_\alpha) \times w_\alpha^\star \longrightarrow w_\beta^\star$$

In fact, we want something more. Like with the word alignment, with word sequence alignment we would prefer as a result a mapping from word sequences to their translation quality:

$$(C_\alpha \times L_\alpha) \times (C_\beta \times L_\beta) \times \mathcal{D}_{\alpha,\beta} \times \mathcal{D}_{\beta,\alpha} \times w_\alpha^\star \longrightarrow \left(w_\beta^\star \rightharpoonup \mathcal{P}(\mathcal{T}(w_\alpha^\star) = w_\beta^\star)\right)$$

The algorithm is based on the sentence alignment of the corpus and in the translation quality algorithm. Given the parallel corpus, the alignment dictionaries and a sequence of words to align, we:

1. search on the corpus (which language is the same with the word sequence) a set of sentences where the word sequence occurs;
2. use the sentence alignment to get each sentence translation;
3. at this point we have a sequence of pairs of sentences: $(s_\alpha, s_\beta)^\star$ where[7] $w_\alpha^\star \sqsubset s_\alpha$;

[6] for simplicity, and to show the formula in one line only, Crp was abbreviated by C, Lex by L and dictionaries $\mathcal{D}(\mathcal{L}_\alpha, \mathcal{L}_\beta)$ by $\mathcal{D}_{\alpha,\beta}$.

[7] here we use this subset operator ($\sqsubset$) to represent the word sequence inclusion, meaning the sequence of words appears in the sentence in that order.

4. for each pair we will use a sliding window algorithm to retrieve the portion of the target language which is the best translation of the sequence we want to align:

   (a) calculate all possible subsequences from the target sentences with similar size (for three different window sizes: same number of words from the source sequence, one more word and one less word) with the sequence we want to align;

   (b) calculate the translation probability between each one of these sequences and the original sequence, choosing the one with better quality;

This alignment can be used as a simple translation tool as figure 5.4 shows — an interaction with a translation shell using the UEP corpus.

```
1  ==> difficult situation
2  Using 6 occurrences (0.732864 seconds)
3    situation difficile       - 0.8025
4    situation très difficile  - 0.8025
5    situation aussi difficile - 0.8025

6  ==> sentenced to death
7  Using 1 occurrences (0.214145 seconds)
8    condamné à mort - 0.443333333333333

9  ==> final version
10 Using 7 occurrences (0.843922 seconds)
11   version définitive - 0.5075
12   définitive         - 0.09
13   définitif          - 0.0875
```

Figure 5.4: Word sequence alignment example

By default, the script searches all the corpus for occurrences; this can lead to much time of search. To solve this, the corpus is previously ranked (using the automatic sentence evaluation method) and only n samples are searched on the corpus (sorted by translation quality).

This functionality will be added to a Distributed Translation Memory system(Simões, Almeida, and Guinovart, 2004) such that computer assisted translation tools can use them to facilitate the translators work.

5.6 Multi-word terminology extraction

Although we had discussed a very simple method for multi-word (biword) bilingual terminology extraction on section 4.2, in this section we will present a more general and interesting tool to extract bilingual multi-word terminology.

Although there are known efficient methods for monolingual multi-term terminology extraction, multilingual terminology extraction is still a problem.

5.6.1 Monolingual multi-word term extraction

For the described method we will use a monolingual terminology extractor presented at (Dias, 2002) and with a C++ implementation discussed in (Gil, 2002). It is included in the senta project(SENTA, 2003). The extractor uses an hybrid algorithm of the "mutual expectation association measure" and the "GenLocalMaxs" algorithms.

Looking to the extractor as a black box, it uses a monolingual text and extracts a set of word sequences, the respective sequence occurrence counter and a measure value for mutual expectation:

$$mwuextractor : text \longrightarrow (MEvalue \times occurrence \times word^{\star})^{\star}$$

In fact, this extractor gives something more than word sequences: it returns patterns. This means that the word sequence can contain *gaps*:

$$mwuextractor : text \longrightarrow (MEvalue \times occurrence \times (word + gap)^{\star})^{\star}$$

For the purpose of this thesis the entries which contain gaps will be removed (ignored) as entries with small length (in characters).

Note that the multi-word terms obtained are extracted regarding to their statistical properties. We cannot see all extracted multi-word terms as true linguistic terms.

5.6.2 Multilingue multi-word term extraction algorithm

The algorithm we will use can be described as:

1. align the original parallel corpora and create the translation dictionary as before;

2. use the monolingual terminology extract in the source corpora and filter it, removing terms with gaps and small length entries;

3. use the word sequence alignment method (described on section 5.5) to extract from the corpora possible translations from the original multi-word unit;

Another option could be first to detect multi-word units, join them in a single word (as done before on section 4.2) and align the corpus. Although this is possible, has big problems: the corpus need a re-alignment and matrix sizes grow.

So, the first option has some advantages:

- corpora is already aligned;
- multi-word term detector needs to be ran only in one of the corpus;
- the translation detection and extraction will be done with knowledge about the languages, which should retrieve better results.

5.6.3 Results and evaluation

First test was done with the Bible. Although the extraction of multi-word terms is more important for technical corpora, the Anacom corpus is too noisy and European Parliament corpus are too big to be used directly by the `senta` extractor.

The extractor was run on the Portuguese version of the bible for extraction of multi-word units of maximum size five. The extractor returns all units with more than two occurrences with size greater or equal to two and smaller than five: found 22 531 units.

From the units found we need to remove gaps, as they cannot be translated[8]. Removing gaps, we remain with 7 295 entries. Figure 5.5 shows the amount of different units found on relation to the number of times their appear in the corpus.

From the figure we can check that there are a lot of units (about 3719) which occur only two times in the corpus. Meanwhile, there is a small number of units that occur a lot in the corpus.

[8] In fact, we could translate the string and maintain gaps. It would result on translations with placeholders (parameterized translations). This is an idea for future work.

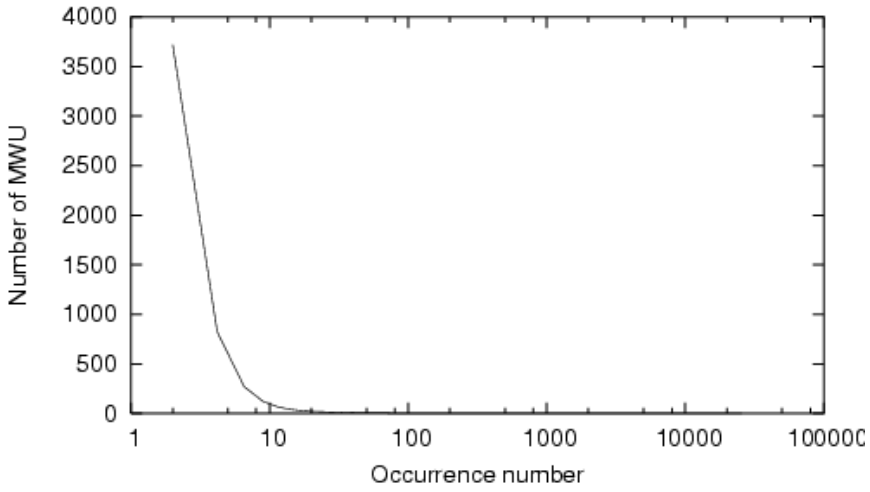

Figure 5.5: Relation between the number of units found and their occurrence number on the corpus

We can say that many of the units found are not real multi-word terms given the small number of occurrences on such a big corpus. This lead to the filtering of the list of multi-word units removing units with too low occurrence.

If we remove multi-word units which occur less than five times in the corpus we remain with 1 856 entires (about 25% of the original list). Although this process removed a lot of noisy entries, they still exist.

To clean this list we can look to the found strings. Many of them (specially noisy ones) are very small like “e ,” or “e o”. These entries can be removed by string size (remove entries with less than some specific number of characters).

Looking to the specific list of the Bible the size chosen was three characters. This process removed 21 entries.

The word sequence alignment took about 2 minutes for these two thousand entries (about 0.07 seconds for entry). Some of these translations are very good (specially those which really are multi-word terms). Table 5.2 shows some good and bad translations. From this table we can detect some problems on the sequence alignment algorithm:

- some really weird translations, like (*13*);
- articles disappearance, like samples (*6*) and (*7*);
- verbs disappearance, like sample (*8*);

- good translations with extra words, like (*19*);

	Portuguese (*input*)	English (*output*)
(*1*)	todos os	all the
(*2*)	até ao	until
(*3*)	entre vós	among you
(*4*)	ninguém pode	no one can
(*5*)	aqueles que	those who
(*6*)	deste mundo	world
(*7*)	os judeus	jews
(*8*)	deus é	god
(*9*)	instintos egoístas	flesh
(*10*)	pedras preciosas	precious stones
(*11*)	estas coisas	these things
(*12*)	dez mil	ten thousand
(*13*)	ao seu	educating
(*14*)	espírito santo	holy spirit
(*15*)	coisa consagrada a Javé	thing dedicated to yahweh
(*16*)	no deserto do Sinai	the desert of sinai
(*17*)	para sempre e eternamente	forever and ever
(*18*)	é caso de lepra	is a case of leprosy
(*19*)	perfume agradável a Javé	sweet-smelling offering to yahweh
(*20*)	a minha voz suplicante	the voice of my pleading

Table 5.2: Multi-word terms translations

5.7 Statistical machine translation

> *Existing translations contain more solutions to more translation problems than any other existing resource.*
>
> *(Isabelle, 1992)*

Statistical machine translation is based on the idea that portions of any sentence can be found on other texts, specially, on parallel ones. We can say this is not the real truth, but happens for most of the cases.

Relying on this idea, the statistical translation aims to divide a sentence on small chunks (three, four or more words) and search on parallel corpus for those sequence occurrence. Found them, the word sequence alignment algorithm can be used to determine the corresponding translations. Optimally,

the translation for those chunks (with overlapping words) can be composed together to form acceptable sentence translations. Of course the good translations observed on section 5.6 occurred because the word sequences appear in the corpus, and most cases, more than one time.

With this in mind, we developed a statistical translator prototype which will be discussed in this section.

For each sentence s_α in the text we want to translate, we split it into its constituent words (or tokens, as punctuation is considered a word in this case): $w_{\alpha,1} \dots w_{\alpha,n}$.

Then, until there are no words to translate, we take a sequence of k words $w_{\alpha,i} \dots w_{\alpha,i+k-1}$ (normally 3 or 4) starting with $i = 0$ and try to find that sequence on the base corpus we are using for the translation. This process was explained in detail on section 5.5.

If the sequence is found, its aligned segment is added to the translation, and we restart the process with $i = i + k$. If not found, we take a smaller size segment ($k = k - 1$) and retry the alignment. This process is done until we find a segment to align (in the last case, when we find a word to align).

Figure 5.6 shows a translation example using as base corpora ten percent of the Portuguese/English european parliament corpus.

The example shows how this method works. First we split the sentence into words. Then, a sliding window with size of three words starts trying to translate our original message. In the example, this was done translating "no parlamento europeu". As this sequence was found on the corpus with more than 0.5 percent of quality, the translation was used and we passed to the next sequence: "existe uma grande". When translating this one, the corpus based translation had quality bellow 0.5 percent, it was discarded and the window shortened (to "existe uma"). The process continues for the rest of the sentence.

Analyzing this example, we can see:

- the first "no" was lost, given that its translation has a big number of possible translations (so, a low translation probability);
- the 0.5 percent threshold for translation was too high, at least for the translation of "existe uma grande". In the other hand, we could not lower it too much, or the wrong translation for "grande crise de" would be chosen;
- some windows should be detected and translated before. For example, the translation of "grande crise de identidade" could be better if "crise

```
1  no parlamento europeu existe uma grande crise de identidade

2  no parlamento europeu...
3          european parliament - 0.6215775385499

4  existe uma grande...
5          there was a great - 0.370103897837301

6  existe uma...
7          there exists a - 0.505804251879454

8  grande crise de...
9          crisis here - 0.342034876346588

10 grande crise...
11         crisis - 0.602234557271004

12 de identidade...
13         identity - 0.727038472890854

14 european parliament there exists a crisis identity
```

Figure 5.6: Example-Based Machine Translation example

de identidade" was translated all together, and "grande" translated independently.

Although there are a lot of problems in this kind of translation, better results can be obtained if we:

- enlarge the corpus size — at the moment we need to implement a better corpus search engine bacuase the present one is consuming too much memory;

- diversify the corpus types — this example was based only on the European parliament corpus; if we add some other corpora like Harry Potter or Tom Sawyer the vocabulary known will enlarge and make translations better.

5.8 Tools development conclusions

This chapter shown how easy is to write applications using probabilistic translation dictionaries, using the NATools programmers interface. The tools presented are, them self, written using the NATools probabilistic dictionaries API.

The described tools (some of them in prototype stage) show that the word alignment is not the end of the road, but the beginning of new and different roads of research.

Chapter 6

Conclusions

A conclusion is simply the place
where someone got tired of thinking.

During this dissertation work was done in the re-engineering of Twente-Aligner, analysis of its application and usage of their resulting probabilistic dictionaries using an application programmers interface. From this work, we conclude:

- open-source tools are very useful resources for re-usability, to learn new technologies and to be used as a cooperative development ambient during time.
- NATools is a tools distributed with documentation, installation instructions and scripts, and a programmer interface toolkit to manage probabilistic dictionaries. The distribution tarball includes command based tools, programmer interface modules and tool prototypes.
- the re-engineering lead to a speed-up on real corpora alignment. The use of slicing for big corpora makes its alignment possible: align small chunks and at the end sum-up the resulting dictionaries.
- translation dictionaries obtained by NATools contain information not present on traditional translation dictionaries. This probabilistic information is important in some areas, like disambiguation on cross-language information retrieval. NATools probabilistic dictionaries can be combined on traditional translation dictionaries, or used instead of

them. Also, they are a good bootstrap method for new translation dictionaries.

- the pre-processing step added to the alignment process is very important. It can be used to apply some transformations to the source corpora, like to pass it through a morphological analyzer to enhance alignment results.
- probabilistic translation dictionaries can be used for different tasks like:
  - bilingual multi-word term extraction and creation of specific knowledge area translation dictionaries;
  - measure the probability of two sentences being a translation of each other — although not a precise measure, it it good enough to rank translations;
  - segment alignment can be done using the measure of translation probability. This is a promising technique for "by example" automatic translation systems;
  - to use these dictionaries on new applications is easy using the programmers interface modules;

Future Work

Many things can be done both to enhance and use alignment results. We can divide the future work basically on three different topics: work being done, future work to enhance the aligner, and future work based on the aligner results:

- at the moment we are working on three different ideas.
  - A new file type is being developed to include lexicon and dictionary files, such that most tools usage will be simplified;
  - The programmers interface is being enriched with new functionality; for example, a `for_each` method with alteration capabilities — not only to create results based on the dictionary, but change the dictionary;
  - Probabilistic translation dictionaries will be available as a web-service on a mega-dictionary project;
  - They will be also used in the Distributed Translation Memory web-services to search translations at word sequence level.

- new techniques can be used to enhance the alignment process and results.
  - When aligning, use not only words but also their morphological analysis. This can be helpful in two ways: first, when the same word has two different meanings, depending on their morphological analysis, the aligner can distinguish them; second, it is possible to align words (or count their co-occurrence) only if they match relatively the morphological analysis;
  - The use of parsing techniques can be used to identify constructions where the words can be glued before alignment to extract multi-word expressions.
- probabilistic dictionaries can be used in new and exciting projects like word-sense disambiguation, sentence alignment or machine translation.

Bibliography

Abaitua, Joseba. 2000. Tratamiento de corpora bilingües. In *La ingenieria lingüística en la sociedad de la información*, July.

Ahrenberg, Lars, Mikael Andersson, and Magnus Merkel, 2000. *Parallel text processing: Alignment and Use of Translation Corpora*, volume 13 of *Text, Speech and Language Technology*, chapter 5 — "A Knowledge lite approach to word alignment", pages 97–116. Kluwer Academic Publishers.

Almeida, J. João and Ulisses Pinto. 1994. Jspell — um módulo para análise léxica genérica de linguagem natural. In *Actas do Congresso da Associação Portuguesa de Linguística*.

Almeida, J. João, Alberto Simões, José Castro, Bruno Martins, and Paulo Silva. 2003. Projecto TerminUM. In *CP3A 2003 – Workshop em Corpora Paralelos, Aplicações e Algoritmos Associados*, pages 7–14. Universidade do Minho.

Almeida, José João, José Alves Castro, and Alberto Manuel Simões. 2002. Extracção de corpora paralelo a partir da web: construção e disponibilização. In *Actas da Associação Portuguesa de Linguística*.

Almeida, José João, Alberto Simões, Diana Santos, and Paulo Rocha. 2004. `Lingua::PT::Segmentador`. `http://www.cpan.org/modules/by-module/Lingua/Lingua-PT-Segmentador-0.01%.tar.gz`.

Almeida, José João, Alberto Manuel Simões, and José Alves Castro. 2002. Grabbing parallel corpora from the web. In *Sociedade Española para el Procesamiento del Lenguaje Natural*, 29, pages 13–20, Sep.

Caseli, Helena Medeiros. 2003. Alinhamento sentencial de textos paralelos português-inglês. Master's thesis, ICMC-USP, February.

Danielsson, Pernilla and Daniel Ridings. 1997. Practical presentation of a "vanilla" aligner. In *TELRI Workshop in alignment and exploitation of texts*, February.

Darwin, Ian. 1997. `File::MMagic`. `http://www.cpan.org/modules/by-module/File/File::MMagic-1.21.tar.gz`.

de Castro, José Alves. 2004. `Lingua::Identify`. `http://www.cpan.org/modules/by-module/Lingua/Lingua-Identify-0.01.tar.g%z`.

Dias, Gaël. 2002. *Extraction Automatique d'Associations Lexicales à Partir de Corpora*. Ph.D. thesis, New University of Lisbon (Portugal) and University of Orlëans (France), 17 December.

Frankenberg-Garcia, Ana and Diana Santos, 2001. *Apresentando o COMPARA, um corpus português-inglês na Web*. Cadernos de Tradução, Universidade de São Paulo.

Gale, William A. and Kenneth Ward Church. 1991. A program for aligning sentences in bilingual corpora. In *Meeting of the Association for Computational Linguistics*, pages 177–184.

Gil, Alexandre Nuno Capinha. 2002. Extracção eficiente de padrões textuais utilizando algoritmos e estruturas de dados avançadas. Master's thesis, Universidade Nova de Lisboa – Faculdade de Ciências e Tecnologia.

Grefenstette, Gregory. 1995. Comparing two language identification schemes. In *JADT 1995, 3rd International Conference on Statistical Analysis of Textual Data*.

Harris, B. 1988. *Are you bitextual?*, volume 7. Language Technology.

Hiemstra, Djoerd. 1998. Multilingual domain modeling in twenty-one: automatic creation of a bi-directional lexicon from a parallel corpus. Technical report, University of Twente, Parlevink Group.

Hiemstra, Djoerd. August 1996. Using statistical methods to create a bilingual dictionary. Master's thesis, Department of Computer Science, University of Twente.

IMS Corpus Workbench. 1994-2002. http://www.ims.uni-stuttgart.de/projekte/CorpusWorkbench/.

Isabelle, Pierre. 1992. Bi-textual aids for translators. *Proceedings of the 8th Annual Conference of the UW Centre for the New OED and Text Research*, pages 1–15.

ISO 639. 1992. *Language Codes*. International Organization for Standardization.

Koehn, Philipp. 2002. Europarl: A multilingual corpus for evaluation of machine translation. Draft, Unpublished.

König, Oliver Christ & Bruno M. Schulze & Anja Hofmann & Esther. March 8, 1999 (CQP V2.2). *The IMS Corpus Workbench: Corpus Query Processor (CQP): User's Manual*. Institute for Natural Language Processing, University of Stutgart.

Melamed, I. Dan. 1999. Bitext maps and alignment via pattern recognition. *Computational Linguistics*, 25(1):107–130.

Melamed, I. Dan. 2000. Models of translational equivalence among words. *Computational Linguistics*, 26(2):221–249.

Overture Systems, Inc. 2004. Altavista. `http://www.altavista.com`.

Resnik, Philip. 1998. Parallel strands: A preliminary investigation into mining the web for bilingual text. In L.Gerber D. Farwell and E. Hovy, editors, *Machine Translation and the Information Soup (AMTA-98)*. Lecture Notes in Artificial Intelligence 1529, Springer.

Resnik, Philip and I. Dan Melamed. 1997. Semi-automatic acquisition of domain-specific translation lexicons. In *7th ACL Conference on Applied Natural Language Processing*, pages 340–347.

Rocha, Paulo Alexandre and Diana Santos. 2000. CETEMPúblico: Um corpus de grandes dimensões de linguagem jornalística portuguesa. In *Actas do V Encontro para o processamento computacional da língua portuguesa escrita e falada (PROPOR'2000)*, pages 131–140. Atibaia, São Paulo, Brasil, 19 a 22 de Novembro.

Santos, Diana and Eckhard Bick. 2000. Providing internet access to portuguese corpora: the ac/dc project. In *Second International Conference on Language Resources and Evaluation, LREC 2000*, pages 205–210, Athens, May-June.

Santos, Diana, Belinda Maia, and Luís Sarmento. 2004. Gathering empirical data to evaluate mt from english to portuguese. In *Workshop on the Amazing Utility of Parallel and Comparable Corpora*, Lisboa, Portugal, May.

Santos, Diana, Paulo Rocha, Luís Sarmento, Alberto Simões, and Luís Costa. 2004. Linguateca — centro de recursos distribuído para a língua portuguesa. `http://www.linguateca.pt`.

SENTA. 2003. Software for the extraction of n-ary textual associations (senta). `http://senta.di.ubi.pt/`.

Simões, Alberto. 2003. Alinhamento de corpora paralelos. In *CP3A 2003 – Workshop em Corpora Paralelos: aplicações e algoritmos associados*, pages 71–77. Universidade do Minho. Braga, Jun. 2003.

Simões, Alberto and José João Almeida. 2004. `XML::TMX`. `http://www.cpan.org/modules/by-module/XML/XML-TMX-0.06.tar.gz`.

Simões, Alberto, José João Almeida, and Xavier Gomez Guinovart. 2004. Memórias de tradução distribuídas. In José Carlos Ramalho and Alberto Simões, editors, *XATA2004 - XML, Aplicações e Tecnologias Associadas*, pages 59–68, February.

Simões, Alberto M. and J. João Almeida. 2003. Natools – a statistical word aligner workbench. *SEPLN*, September.

Simões, Alberto Manuel and José João Almeida. 2001. `jspell.pm` – um módulo de análise morfológica para uso em processamento de linguagem natural. In *Actas da Associaçáo Portuguêsa de Linguística*, pages 485–495.

Tiedemann, Jörg. 1997. Automatical lexicon extraction from aligned bilingual corpora. Technical report, University of Magdeburg.

Tiedemann, Jörg. 2003. *Recycling Translations - Extraction of Lexical Data from Parallel Corpora and their Application in Natural Language Processing*. Ph.D. thesis, Studia Linguistica Upsaliensia 1.

Tiedemann, Jörg and Lars Nygaard. 2003. Opus - an open source parallel corpus. In *The 13th Nordic Conference on Computational Linguistics*, Reykjavik. University of Iceland.

Tiedemann, Jörg and Lars Nygaard. 2004. Opus - an open source parallel corpus — webpage. `http://logos.uio.no/opus/`, April.

TRADOS, Ireland Ltd, 2001. *Advanced Alignment Guide*. Dublin, Ireland, July.

Véronis, Jean, editor. 2000. *Parallel Text Processing: alignment and use of translation corpora*, volume 13 of *Text speech and language technology*. Kluwer Academic Publishers.

Yaacov Choueka, Ehud S. Canley and Ide Dagan, 2000. *Parallel text processing: Alignment and Use of Translation Corpora*, volume 13 of *Text, Speech and Language Technology*, chapter 4 — "A comprehensive bilingual word alignment system — aplication to disparate languages: Hebrew and English", pages 69–98. Kluwer Academic Publishers.

Zipf, George Kingsley. 1932. Selective studies and the principle of relative frequency in language.

Appendix A

Mathematical Notation

> *The good Christian should beware of mathematicians and all those who make empty prophecies. The danger already exists that mathematicians have made a covenant with the devil to darken the spirit and confine man in the bonds of Hell.*
>
> *St. Augustine*

This appendix is a quick introduction to the mathematical notation and calculus behind the data representations used in this thesis.

Data or object types are normally represented by capitalized strings. For example, Lex represents an object (in fact, a file which we can see as an object).

To combine data types we have different operators:

- the product of data types ($Lex \times Crp$) represents the aggregation of these simple data types on a compound one, with more than one field. We can see this operator as `structs` from the C language.

  Product is associative:

$$\begin{aligned}(LexA \times Crp) \times LexB &\equiv LexA \times (Crp \times LexB) \\ &\equiv LexA \times Crp \times LexB\end{aligned}$$

  When we have a data type like $Lex \times Lex \times Lex \ldots \times Lex$ we can abbreviate using Lex^n, where n is the number of items in the product. In cases where n can change (like lists, or sequences) we use a star ($Lex^\star$).

Over this construct it is possible to use a function named π_i where i is an integer. This function returns the i-th type of a product construction. For example:

$$\begin{aligned} \pi_1(LexA \times LexB \times LexC) &\equiv LexA \\ \pi_3(LexA \times LexB \times LexC) &\equiv LexC \end{aligned}$$

- the co-product is represented by the sum operator (+) and is an aggregation of data-types where only some one of the types can exist at a time. See it as the `union` construct of the C language.

  $LexA + LexB$ means a data type which holds one value of the data type $LexA$ or one value of the data type $LexB$. It is associative:

$$\begin{aligned} LexA + (LexB + LexC) &\equiv (LexA + LexB) + LexC \\ &\equiv LexA + LexB + LexC \end{aligned}$$

- the finite function is represented by an incomplete arrow ($\rightharpoonup$) and is a map between two data types. It can be seen as an hash table or an associative array (where indices are not necessarily integers).

  $ID \rightharpoonup LexA$ maps identifiers to objects of type $LexA$. If we have $Map \equiv ID \rightharpoonup LexA$, then:

$$\begin{aligned} dom(Map) &\equiv ID \\ ran(Map) &\equiv LexA \end{aligned}$$

  and $Map(id)$ is the value of type $LexA$ associated with an identifier id of type ID.

  Finite functions are not associative:

$$ID \rightharpoonup (Lang \rightharpoonup Lex) \not\equiv (ID \rightharpoonup Lang) \rightharpoonup Lex.$$

  Although the operator is not associative, we normally associate at the right, which means:

$$ID \rightharpoonup (Lang \rightharpoonup Lex) \equiv ID \rightharpoonup Lang \rightharpoonup Lex.$$

- the NULL data-type (or any other data-type which has only one value) is represented by 1. With this in mind, we can write the following rule:

$$1 \times A \equiv A$$

  This data-type is very used to represent a C pointer. In fact, $A + 1$ represents an object of type A (pointer pointing to an A object) or a 1 (pointer pointing to NULL).

Some rules can be proven using this calculus. Since some of them are useful to understand the mathematics used in this document, they are reproduced here:

- product is distributive over the co-product:

$$A \times B + A \times C \equiv A \times (B + C)$$

- a sequence is map from the position to the element in that position

$$A^n \equiv \mathbb{N}_n \rightharpoonup A$$

- a product as key for a finite function can be changed to a finite function:

$$A \times B \rightharpoonup C \equiv A \rightharpoonup (B \rightharpoonup C)$$

Appendix B

Software Documentation

Documentation is like sex: when it is good, it is very, very good; and when it is bad, it is better than nothing.

Dick Brandon

This section includes documentation for commands and modules developed and which are included in NATools. Notice that this tool is being developed continuously: so, for up-to-date documentation check the distribution tarball or the CVS web-site (start at `http://natura.di.uminho.pt`).

B.1 Software Installation

NATools is distributed in a tarball as GPL open-source software. It is mainly written in C and Perl with configuration and installation processes written in autoconf and automake, as usual on gnu software.

B.1.1 Requirements

NATools uses a shell configuration script and a makefile oriented build system. You would need also a C compiler and Perl.

Necessary C libraries include:

- common C math library;
- gnu zlib library for access to compresses files, available from `http://www.gnu.org`;

- glib-2.0 library for some data-structures, code portability and future unicode support. You can find it at `http://www.gtk.org`.

- and a bunch of Perl modules, all available from CPAN: `http://www.cpan.org`;

B.1.2 Compilation

The compilation process should be simple, as it is handled at all by automake and autoconf. The following steps should do the task:

```
1 tar zxvf NATools-x.x.tar.gz
2 cd NATools-x.x
3 ./configure
4 make
5 make check
6 make install
```

To the `configure` script it is possible to use a set of switches which can be viewed using `./configure --help`.

B.2 nat-this

The terminology extractor top-level script to apply to a TMX file.

Synopsis

```
1 nat-this file.tmx
```

Description

This script is a top level layer to the `nat-these` script, which aligns directly a TMX file. The script extract the languages to two separate files and calls `nat-these`.

See also

`nat-these`, NATools documentation;

Copyright

```
1 | Copyright (C)2002-2003 Alberto Simoes and Jose Joao Almeida
2 | GNU GENERAL PUBLIC LICENSE (LGPL) Version 2 (June 1991)
```

B.3 nat-pre

A pre-processor for parallel texts, counting words, checking sentence numbers, and creating auxiliary files.

Synopsis

```
1 | nat-pre <crp-text1> <crp-text2> <lex1> <lex2> <crp1> <crp2>
```

Description

This tools is integrated with `nat-these` command, and is not intended to be used directly by the user. It is an independent command so that we can use it inside other programs and/or projects.

The tool objective is to pre-process parallel corpora texts and create auxiliary files, to access directly corpus and lexicon information.

The `crp-text1` and `crp-text2` should be in text format, and should be sentence aligned texts. Each one of these texts should contain lines with the single character `$` as sentence separator. As the text is aligned, the number of sentences from both text files should be the same.

To use it, if you have the aligned text files `txt_PT` and `txt_EN`, you would say:

```
1 | nat-pre txt_PT txt_EN txt_PT.lex txt_EN.lex txt_PT.crp txt_PT.lex
```

Where the `.lex` files are *lexical* files and `.crp` files are *corpus* files.

If you process more than one pair of files, giving the same *lexical* file names, identifiers will be reused, and lexical files expanded.

Internals

Corpus and lexical files are written on binary format, and can be accessed using NATools source code. Here is a brief description of their format:

lexical files

these files describe words used on the corpus. For each different word (without comparing cases) it is associated an integer identifier. This file describe this relation.

The format for this file (binary) is:

```
number of words (unsigned integer, 32 bits)

'words number' times:
  word identifier (unsigned integer, 32 bits)
  word occurrences (unsigned integer, 32 bits)
  word (character sequence, ending with a null)
```

If you need to access directly these files you should download the NATools source and use the `src/words.[ch]` functions.

corpus files

these files describe corpora texts, where words were substituted by the corresponding integer identifier.

The binary format for this *gzipped* file is:

```
corpus size: number of words (unsigned integer, 32 bits)

corpus size times:
  word identifier (unsigned integer, 32 bits)
  flags set (character, 8 different flags)
```

If you need to access directly these files you should download the NATools source and use the `src/corpus.[ch]` functions.

The flags used are:

1. x1
   the word appeared all in UPPERCASE;
2. x2
   the word appeared Capitalized;

Two other files are created also, named `.crp.index` which index offsets for sentences on corpus files.

See also

NATools documentation;

Copyright

```
1 Copyright (C)2002-2003 Alberto Simoes and Jose Joao Almeida
2 Copyright (C)1998 Djoerd Hiemstra

3 GNU GENERAL PUBLIC LICENSE (LGPL) Version 2 (June 1991)
```

B.4 nat-initmat

Initialize a sparse matrix with words co-occurrence.

Synopsis

```
1 nat-initmat <crp1> <crp2> [<exc1> <exc2>] <matrix>
```

Description

This tool is used internally by `nat-these` and is not intended to be used independently. Basically, this tool takes two corpora files created by `nat-pre` and allocates a sparse matrix, where rows indexes correspond to word identifiers on the source corpus, and column indexes correspond to word identifiers on the target corpus. Cells count the words co-occurrence on the same sentence. The `matrix` file is then created with the matrix information.

Optionally, you can pass to the system two exclude lists, as returned by the `nat-words2id` tool. This words will be ignored, and counting will not be done for them.

The matrix is saved and can be processed later by EM-Algorithm methods IPFP (`nat-ipfp`), Sample A (`nat-samplea`) and Sample B (`nat-sampleb`).

See also

nat-words2id, nat-pre, NATools documentation

Copyright

```
1 Copyright (C)2002-2003 Alberto Simoes and Jose Joao Almeida
2 Copyright (C)1998 Djoerd Hiemstra

3 GNU GENERAL PUBLIC LICENSE (LGPL) Version 2 (June 1991)
```

B.5 nat-ipfp

One of the three possible EM-Algorithm implementations of NATools

Synopsis

```
nat-ipfp <steps> <crp1> <crp2> <mat-in> <mat-out>
```

Description

This program is not intended to be used independently. It is used internally by `nat-these` script.

IPFP is an iterative method for the EM-Algorithm. To use it, you must supply the number of times the method should iterate, both corpus files (created by `nat-pre`), the sparse co-ocurrence matrix file (created by `nat-initmat`) and the file name where the enhanced matrix should be placed.

See also

NATools, nat-samplea, nat-sampleb

Copyright

```
Copyright (C)2002-2003 Alberto Simoes and Jose Joao Almeida
Copyright (C)1998 Djoerd Hiemstra

GNU GENERAL PUBLIC LICENSE (LGPL) Version 2 (June 1991)
```

B.6 nat-samplea

One of the three possible EM-Algorithm implementations of NATools

Synopsis

```
nat-samplea <steps> <crp1> <crp2> <mat-in> <mat-out>
```

Description

This program is not intended to be used independently. It is used internally by `nat-these` script.

Sample-a is an iterative method for the EM-Algorithm. To use it, you must supply the number of times the method should iterate, both corpus files (created by `nat-pre`), the sparse co-ocurrence matrix file (created by `nat-initmat`) and the file name where the enhanced matrix should be placed.

See also

NATools, nat-ipfp, nat-sampleb

Copyright

```
1 Copyright (C)2002-2003 Alberto Simoes and Jose Joao Almeida
2 Copyright (C)1998 Djoerd Hiemstra
3 GNU GENERAL PUBLIC LICENSE (LGPL) Version 2 (June 1991)
```

B.7 nat-sampleb

One of the three possible EM-Algorithm implementations of NATools

Synopsis

```
1 nat-sampleb <steps> <crp1> <crp2> <mat-in> <mat-out>
```

Description

This program is not intended to be used independently. It is used internally by `nat-these` script.

Sample-b is an iterative method for the EM-Algorithm. To use it, you must supply the number of times the method should iterate, both corpus files (created by `nat-pre`), the sparse co-ocurrence matrix file (created by `nat-initmat`) and the file name where the enhanced matrix should be placed.

See also

NATools, nat-ipfp, nat-samplea

Copyright

```
1 | Copyright (C)2002-2003 Alberto Simoes and Jose Joao Almeida
2 | Copyright (C)1998 Djoerd Hiemstra
3 | GNU GENERAL PUBLIC LICENSE (LGPL) Version 2 (June 1991)
```

B.8 nat-mat2dic

A translator from co-occurrence matrices to a dictionary file.

Synopsis

```
1 | nat-mat2dic <mat-in> <matdic-out>
```

Description

This command is not intended to be used independently. It is used in conjunction with `nat-ipfp`, `nat-samplea` or `nat-sampleb` and `nat-post`.

Translates the co-occurrence matrix after the application of EM-Algorithm. The dictionary is an intermediary format used by `nat-post` to write the dictionary on a readable format.

See also

nat-initmat, nat-ipfp, nat-samplea, nat-sampleb and remaining NATools documentation.

Copyright

```
1 | Copyright (C)2002-2003 Alberto Simoes and Jose Joao Almeida
2 | Copyright (C)1998 Djoerd Hiemstra
3 | GNU GENERAL PUBLIC LICENSE (LGPL) Version 2 (June 1991)
```

B.9 nat-postbin

A translator from dictionary file to the Perl readable format.

Synopsis

```
nat-postbin <matdic-in> <lex1> <lex2> <out-dic1> <out-dic2>
```

Description

This command is not intended to be used independently.

`nat-postbub` reads the dictionary created with `nat-mat2dic` and lexicon files created by the `nat-pre` tool, to write two dictionary files on a very small binary format.

The internal format can change a lot, so please use NATools to manage these files.

The format is a gzipped binary file. First 32 bits unsigned integer is the number of entries in the dictionary. Follows a sequence of MAXENTRY size, with pairs of translation identifier (32 bits unsigned integer) and probability (32 bits float).

See also

nat-pre, nat-mat2dic and remaining NATools documentation.

Copyright

B.10 nat-css

Corpus Search Sentence utility.

Synopsis

```
 nat-css [-q <rank>] <lex1> <crp1> <lex2> <crp2> [<sent_nr> | all]
```

Description

`nat-css` is used after the alignment process (using, for example, `nat-these` tool). Its objective is to search sentences where one word occurs.

The tool has two methods of use:

```
nat-css [-q <rank>] <lex1> <crp1> <lex2> <crp2>
```

where the tool opens the two lexicon files and the two corpora files in interactive mode. The user writes a word or a word sequence and the tool finds its occurrences on the corpus, printing the sentence from `crp1` where it occurs, and aligned sentence from `crp2`. If the `rank` is provided, the tool prints the ranking or quality of the alignment, too.

The other method of operation is:

```
nat-css [-q <rank>] <lex1> <crp1> <lex2> <crp2> (<sent_nr> | all)
```

where the tool prints the pair of sentences number `sent_nr` or all the sentences (if the `all` option is used). Again, if `rank` is used, the tool prints the ranking or quality of the alignment.

See also

nat-rank, nat-these, NATools documentation;

Copyright

B.11 NAT::Lexicon

Perl extension to encapsulate NATools Lexicon files

Synopsis

```
use NAT::Lexicon;

$lex = NAT::Lexicon::open("file.lex");

$word = $lex->word_from_id(2);

$id = $lex->id_from_word("cavalo");
```

```
5  @ids = $lex->sentence_to_ids("era uma vez um gato maltez");
6  $sentence = $lex->ids_to_sentence(10,2,3,2,5,4,3,2,5);
7  $lex->close;
```

Description

This module encapsulates the NATools Lexicon files, making them accessible using Perl. The implementation is based on OO philosophy. First, you must open a lexicon file using:

```
1  $lex = NAT::Lexicon::open("lexicon.file.lex");
```

When you have all done, do not forget to close it. This makes some memory frees, and is welcome for the process of opening new lexicon files.

```
1  $lex->close;
```

Lexicon files map words to identifiers and vice-versa. Its usage is simple: use

```
1  $lex->id_from_word($word)
```

to get an id for a word. Use

```
1  $lex->word_from_id($id)
```

to get back the word from the id. If you need to make big quantities of conversions to construct or parse a sentence use `ids_to_sentence` or `sentence_to_ids` respectively.

See also

See perl(1) and NATools documentation.

AUTHOR

Alberto Manuel Brandao Simoes, <albie@alfarrabio.di.uminho.pt>

Copyright AND LICENSE

B.12 NAT::Dict

Perl extension to encapsulate Dict interface

Synopsis

```
use NAT::Dict;

$dic = NAT::Dict::open("file.bin");

$dic->save($filename);
$dic->close;

$dic->add($dic2);

$dic->size();

$dic->exists($id);
$dic->occ($id);
$dic->vals($id);

$dic->for_each( sub{ ... } );
```

Description

The Dict files (with extension `.bin`) created by NATools, are mapping from identifiers of words on one corpus, to identifiers of words on another corpus. Thus, all operations performed by this module uses identifiers instead of words.

You can open the dictionary using

```
$dic = NAT::Dict::open("dic.bin");
```

Then, all operations are available by methods, in a OO fashion. After using the dictionary, do not forget to close it using

```
$dic->close().
```

The `add` method receives a dictionary object and adds it with the current contents. Notice that both dictionaries need to be congruent relatively to word identifiers. After adding, do not forget to save, if you with, with

```
$dic->save("new.dic.bin");
```

The `size` method returns the total number of words on the corpus (the sum of all word occurrences). To get the number of occurrences for a specific word, use the `occ` method, passing as parameter the word identifier.

To check if an identifier exists in the dictionary, you can use the `exists` method which returns a boolean value.

The `vals` method returns an hash table of probable translations for the identifier supplied. The hash contains as keys the identifiers of the possible translations, and as values their probability of being a translation.

Finally, the `for_each` method makes you able to cycle through all word on the dictionary. It receives a funcion reference as argument.

```
$dic->for_each( sub{ ... } );
```

Each time the function is called, the following is passed as `@_`:

```
word => $id , occ => $occ , vals => $vals
```

where `$id` is the word identifier, `$occ` the result of calling `occ` with that word, and `$vals` is the result of calling `vals` with that word.

See also

See perl(1) and NATools documentation.

AUTHOR

Alberto Manuel Brandao Simoes, <albie@alfarrabio.di.uminho.pt>

Copyright AND LICENSE