

# Implementation of Collocation Extraction in Unitex

B. Burak Arslan

September 12, 2007

## Abstract

Collocation extraction is an elaborate problem in the field of corpus linguistics that requires both statistical and linguistic information in order to be successful. The problem is attracting more attention as the importance of collocations, (aka multi-word expressions) is recognized by the NLP community. In this report, I present the work that I reviewed, and the `Colloc` module that was the result of this work, by explaining reasons behind taken decisions during the implementation, and of course, what remains to be done.

## Résumé

L'extraction des collocations est un problème compliqué du domaine de la linguistique de corpus qui exige l'utilisation de l'information statistique autant que linguistique afin d'obtenir des résultats satisfaisants. Au fur et à mesure que l'importance des collocations (également Multi-Word Expressions ou MWE) est perçue par la communauté du TLN, ce problème attire davantage l'attention. Dans cette présentation de stage, je présente le travail que j'ai examiné, ainsi que le module `Colloc` qui en est le résultat, en expliquant les raisons des décisions prises pendant l'implémentation, et bien sûr, ce qu'il reste à faire.

## 1 Introduction

The term collocation (aka compound word or multi-word expression) is used to represent groups of words that either have slightly different meanings when used together, or have become idiosyncratic with heavy use. (e.g. we say “traffic lights”, not “traffic regulators”, nor “flow routers”) Their importance were first noticed by foreign language teachers and translators, as most of the time collocations are recognized as expressions that, when translated word-by-word, stand out as awkward at best, for a native speaker of the target language <sup>1</sup>.

---

<sup>1</sup>e.g. think about the direct translation of the collocation “dead serious” to any other language you know

Another important property of the collocations is the fact that they are language dependent and can only be learned by observing their occurrence in language use; they are otherwise not predictable[4]. So, collocations can't be generated from a dictionary of simple words, they can only be extracted from large corpora.

When looking for a more formal definition, one observes that many were proposed during the course of the research done on this topic.

Further study of the collocations in the pre-computing era showed that collocations have particular statistical distributions[6]. Basing on this information, collocations are defined to be "recurrent combinations of words that co-occur more often than expected by chance and that correspond to arbitrary word usages". While this purely statistical definition was seemingly correct, later research has shown that it is extremely difficult to extract collocations by purely statistical tests.[1].

According to a more recent work collocations should rather be defined as: "idiosyncratic interpretations that cross word boundaries"[3], which is a more accurate, but much more vague definition which is harder to track computationally.

There are many types of collocations which are generally classified basing on their behaviour in real-world corpora. We can say that these types vary between two extremes, based on their rigidness in form. Some collocations are simply words-with-spaces, which are simply words that always follow each other, and receive no morphosyntactic modification no matter the context. These types of collocations are the easiest to extract, especially when they are statistically distinguishable. On the other extreme are collocations that vary wildly in form depending on their context.

## 2 Design of a Collocation Extraction System

Generally it all boils down to two main issues to be resolved in order to implement a robust collocation extraction system:

1. There is no general form that defines all collocations.

This also means that one given collocation may be observed under different forms in a corpus. For example, the collocation "to make a decision" may appear as "*the decisions he made*", or "*he made an important decision*", or "*he is about to make a decision*". So, a collocation system should incorporate lexical and syntactical knowledge in order to prevent statistical information from fragmenting, and should be robust to such heavy alterations in the form.

2. As said above, not all collocations can be extracted by adopting purely statistical methods, so other properties of collocations should be exploited as well.

While it is accepted this sparseness is the result of the nature of the collocations, it is partly due to the existence of thematic collocations<sup>2</sup> as well.

---

<sup>2</sup>which are what makes up the *lingo* used by people that are somehow organized around a common social or professional activity, and are only encountered in corpora that share that particular theme

Basically, all of the collocation extraction systems that I reviewed work in three steps:

1. Identifying two-word collocation candidates
2. Eliminating them
3. Combining them to collocations that may span more words than two.

The defining steps being identification and elimination, the two may be undistinguishable. Two main approaches exist:

**Deep parsing:** Apply syntactic analysis to the text (using a deep parser like Fips[7]) and extract collocation candidates basing on relations between words [5]. This is a relatively complicated approach that requires a full-blown grammar and its parser for the target language.

**Shallow Parsing:** When working on raw text, it is always a good idea to do some preprocessing in order to normalize the data at hand. So, operations like marking sentence boundaries, lemmatization, POS-tagging etc. simplifies greatly the lives of processess that may follow[6]. These operations are called *shallow parsing* operations, because they don't seek to obtain the structure of the sentence(s).

While shallow parsing approach is simpler to adopt, because it makes use of less linguistic information, it generates more bogus data. Smadja, in his work[6], introduces limits such as defining the "neighborhood of a word" to a small number in order to cope with this fact. Seretan and Wehrli argue, in [5], that this limit is too low. They should be right, because in fact, such limits are rather artificial that are put because of practical reasons (limits of modern personal computers) and not algorithmic ones.

I chose to take the path of shallow parsing by making use of Unitex' various facilities, but tried not to introduce any artificial constraints for simplifying memory management.

## 2.1 Implementation Details

At first, the raw corpus is preprocessed via Unitex' various tools. These tools apply the following operations on the raw corpus:

1. Normalization
2. Tokenization
3. Detection of Sentence Boundaries
4. Lemmatization/POS Tagging

5. Identification of MWE's of the sort "words-with-spaces". Thanks to the lemmatization operation, this process can also capture MWE's that may receive a few lexical alterations. (i.e. the collocation *part of speech* that may be seen like *parts of speech* in some places is also recognized) Colloc tries its best to ignore these combined states when encountered in the text automata in order to eliminate invalid pairs they cause.

This string of operations results in a text automata that results in a kind of matrix where for every word  $w_i$ , there is an interpretation  $s_{i,j}$  with  $j$  mostly bigger than 1.

Colloc commutatively combines, than counts these different interpretations two by two, never combining two different interpretations of the same word. This is where the memory needs may be more than typical; so some precautions are needed to keep the computing environment sane. Colloc contains a user-configurable solution which is what I call **compacting**: Assuming that a  $N$  phrase subset of the corpus exhibits the same statistical properties as the whole corpus itself, Colloc may delete, every  $N$  sentences, combinations that have frequency values below

$$(i - s) \frac{t}{e}$$

where

$i$  is the current number of the sentence

$s$  is the number of the first sentence

$t$  is the threshold value

$e$  is the number of the last sentence

This algorithm deletes entries that will supposedly be below the user-specified threshold in the end of the computation, thus resulting in a much more efficient memory usage. Note that this is optional behaviour that is needed to be enabled via the command line arguments passed to the Colloc executable.

Another precaution is to ignore part of the information that comes from the DELA dictionary entries. Split in levels 0 through 3 in the following manner, it helps the user keep the memory usage to a minimum by discarding information that is not needed.

**Level 0** Keeps only lemmatized form:

eg. Paris

**Level 1** lemmatized form with POS:

eg. Paris.N

**Level 2** lemmatized form with POS and additional semantic info:

eg. Paris.N+PR+DetZ+Toponyme+Ville+IsoFR

**Level 3** Full DELA form:

eg. Paris.N+PR+DetZ+Toponyme+Ville+IsoFR:ms:fs

## 2.2 What remains to be done

Actually, what is done until today in this project barely scratches the face of the collocation extraction problem. Complex linguistic operations are needed to be applied to corpora in order to produce competitive results with other work on collocation extraction.

**Detection of boundaries of subordinate clauses** A first step towards obtaining safer word pairs is to combine words that are only in the same subordinate clause in a complex sentence. According to my observations, collocations do not seem to cross subordinate boundaries in a random complex sentence, if they are not the simple sentences themselves. While the problem of detection of the boundaries of subordinate clause(s) in a complex sentence remains mostly an open problem, experiences with a restricted corpus with a simpler subordinate boundary detection system deserves attention. Such a system would drastically reduce the number of bogus combinations, increasing the success rate of the `Colloc` module and also resulting in remarkable performance gains.

**Unification** Even after eliminating as much bogus pairs as possible, one will still end up with invalid word pairs. By invalid, I mean pairs like *le table* or pairs that are statistically very significant like *le le*, *le du* which are out-and-out wrong. Such pairs are possible, as we are combining *all* of the words of a sentence in pairs. So, applying linguistic operations like unification will result in pairs of much higher quality.

**Disambiguation** Combinatorial explosion while processing candidates is also due to the fact that more than one interpretation exists for a surface string. So, elimination of incorrect or inappropriate parses will also help increase the efficiency of the program. For example, the pair *le.DET la.DET* is invalid, (which occurs quite frequently in a French corpus) whereas *le.DET la.N* should be kept.

**Smarter thresholding** Thresholding is an effective elimination method when dealing with frequency data. An analysis of the first 30.485 sentences of the `lm94`<sup>3</sup> corpus shows that 85% of 4.241.598 pairs have frequencies  $\leq 3$ .<sup>4</sup>

The weakness of thresholding is its mercilessness. While this method is a very simple and effective way of eliminating most of the pairs that can not be considered as collocations one way or the other, one should either:

1. Adopt a reliable method of determining this threshold which should especially tune itself according to the size of the corpora at hand, or
2. Not rely on thresholding at all and use it as just as a simple filtering mechanism.

---

<sup>3</sup>Le Monde '94, a 843.000 sentence corpus of modern French

<sup>4</sup>An environment for R[2] is saved in `Unitex-C++/build` directory, in case one seeks to conduct further statistical analysis. Also see the script `make_environment` that comes with this document which creates this environment from scratch.

**Combining pairs** There exist collocations that span more than two words. So, a combination algorithm should be implemented in order to detect such collocations. Note that the current data structure that holds collocation candidates needs to be extended for this purpose because the information about the pairs' origins is currently discarded.

## 3 Other Contributions

During my time in Marne-la-Vallée University, I have made several other contributions to the Unitex project.

### 3.1 Memory Management

A great deal of my time was spent on optimizing memory usage. I have tried many libraries for this goal, and now two of them are used in Unitex. One is Judy and the other is BerkeleyDB.

Both are associative arrays of some sort, performing mapping between key-value pairs (which are arbitrary-length binary data). The key difference is that Judy is optimized to be fast, and for in-memory use, while BerkeleyDB is designed as an embedded database that has data security as a first priority instead of speed. BDB's interesting feature was its possibility to be configured as an in-memory associative array, which, once a pre-defined memory limit was reached, started to cache data to disk. While it kept the operating system quite happy, it did not result in a noticeable performance increase. BDB is a complex library with many configuration options, so it may be possible to further optimize the BDB configuration in order to obtain more subtle performance gains. That's why I did not drop the BDB support, and actually implemented a thin abstraction layer on top of Judy and BDB, (available in the form of `array_*` functions in `Array.{h,cpp}`) that uses only one of them depending on a compilation flag given in the `Makefile`.

Another contribution of mine was a read only file buffer that is designed to work on large files. This got required in `Freq`, my introductory project to Unitex (cf next section). With its `buffer_*` functions, one can access an arbitrarily large file, making sure that the memory usage is never more than was specified initially when initializing the buffer. It is optimized to reduce file reads in cases where sequential access to file is needed. It can be found in `Buffer_ng.{h,cpp}`. When the `Buffer_ng.cpp` file is compiled with `-DBUFFER_NG_TEST` option as a standalone executable, it results in a binary that demonstrates the behaviour of `Buffer_ng` in different cases. Instead of using direct system calls, I have used standard file functions in order to increase portability (maybe sacrificing some performance in \*nix systems).

### 3.2 Frequency Computation

I was introduced to the Unitex project by implementing a frequency computation module. It takes as input the `concord.ind` file produced by the `Locate` module and the `text.cod` file produced by the tokenizer, it displays the frequency of all the tokens in the vicinity of the given token(s) in the `concord.ind` file.

## References

- [1] Timothy Baldwin and Aline Villavicencio. Extracting the unextractable: a case study on verb-particles. In *COLING-02: proceeding of the 6th conference on Natural language learning*, pages 1–7, Morristown, NJ, USA, 2002. Association for Computational Linguistics.
- [2] R Development Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2007. ISBN 3-900051-07-0.
- [3] Ivan A. Sag, Timothy Baldwin, Francis Bond, Ann A. Copestake, and Dan Flickinger. Multiword expressions: A pain in the neck for nlp. In *CICLing*, pages 1–15, 2002.
- [4] Violeta Seretan, Luka Nerima, and Eric Wehrli. Extraction of multi-word collocations using syntactic bigram composition. In *Proceedings of the Fourth International Conference on Recent Advances in NLP (RANLP-2003)*, pages 424–431, Borovets, Bulgaria, 2003.
- [5] Violeta Seretan and Eric Wehrli. Accurate collocation extraction using a multilingual parser. In *Proceedings of the 21st International Conference on Computational Linguistics and 44th Annual Meeting of the Association for Computational Linguistics*, pages 953–960, Sydney, Australia, July 2006. Association for Computational Linguistics.
- [6] Frank Smadja. Retrieving collocations from text: Xtract. *Comput. Linguist.*, 19(1):143–177, 1993.
- [7] Eric Wehrli. Fips, a “deep” linguistic multilingual parser. In *ACL 2007 Workshop on Deep Linguistic Processing*, pages 120–127, Prague, Czech Republic, June 2007. Association for Computational Linguistics.