# BACO – A large database of text and co-occurrences

**Luís Sarmento**
Faculdade de Engenharia da Universidade do Porto (NIAD&R) and Linguateca
las@fe.up.pt

**Abstract**

In this paper we introduce a public resource named BACO (Base de Co-Ocorrências), a very large textual database built from the WPT03 collection, a publicly available crawl of the whole Portuguese web in 2003. BACO uses a generic relational database engine to store 1.5 million web documents in raw text (more than 6GB of plain text), corresponding to 35 million sentences, consisting of more than 1000 million words. BACO comprises four lexicon tables, including a standard single token lexicon, and three n-gram tables (2-grams, 3-grams and 4-grams) with several hundred million entries, and a table containing 780 million co-occurrence pairs. We describe the design choices and explain the preparation tasks involved in loading the data in the relational database. We present several statistics regarding storage requirements and we demonstrate how this resource is currently used.

## 1. Introduction

The use of generic relational databases for corpora search and linguistic purposes has already been experimented with for a few years (Davies, 2003), but has been attracting renewed attention with the increasing amount of textual data available, taken from large web crawls (see for example the Wacky project at (http://wacky.sslmit.unibo.it/). Given the current limitations of specific linguistically-motivated corpus query systems in dealing with "gigaword" corpora, researchers are turning their attention to generic database systems. Database systems may provide a very good option for corpus linguistics when dealing with very large amounts of raw text (i.e. non-annotated text), which is often the case when working with very large crawls or text collections. Recently, a large crawl of the Portuguese Web – the WPT03 collection - was made available to the scientific community (http://www.linguateca.pt/wpt03/). The WPT collection is one of the largest Portuguese document collections but has not yet been fully explored: efficiently searching its 15 GB of data is not simple. We thus decided to try to use a generic database system to see if it was possible to efficiently deal with such a large amount of data. After some simple but encouraging tests, we started BACO (Base de Co-Ocorrências), a database of text and co-occurrences.

## 2. Requirements and Design Choices

Developing BACO involved many design decisions regarding how data from the WPT03 collection should be pre-processed and stored in a generic database system. The first design decision was concerned with the granularity of the text representation i.e. which should be the smallest unit of text to be stored as an individual tuple in the database. The most common options usually considered are (i) storing one complete document per tuple, (ii) a sentence per tuple, or (iii) storing each individual token in one separate tuple. This choice has strong implications on the size (in number of tuples) of the table that stores the text. Choosing a finer-grained text unit will increase the number of tuples to be stored which usually leads to some degradation in search and retrieval performance, but it also allows one to build much more expressive queries that may operate over smaller text particles. But since for many NLP and IE applications (e.g.: finding definitions or semantic relations, retrieving answers in automatic QA systems) relevant information is often found inside the scope of a single sentence we decided that BACO would store text on the basis of sentences. Furthermore, this is a solution that represents a balance between query expressiveness and table size.

### 2.1 The Need for Auxiliary N-Gram Tables

In many NLP applications it is important to know which are the most significant collocations / lexical contexts of a given word. However, trying to obtain this information by querying the entire text base each time we need it is simply inefficient. We thus decided to include in BACO, (besides the traditionally one-word dictionary) several n-gram tables, i.e. 2-word, 3-word and 4-word lexicons, which may be directly queried for obtaining information about collocations or surrounding contexts, avoiding the need to search the entire collection. Since these are very simple tables, containing just a few columns, they may be efficiently indexed and queried. The downside of including these tables in BACO is that they involve replicating the same data several times. For example, the data in any N-gram table could be easily obtained from the data contained in the (N+1)-gram table. As we will describe later, this redundancy greatly increases the database storage requirements (the N-gram tables grew to approximately 18 GB, excluding indexes). Nevertheless, we think its worth sacrificing disk storage for the query speed that can be obtained in return.

### 2.2 The Co-occurrence Table

Information about the co-occurrence of words is also relevant for many NLP tasks. However, compiling the information about co-occurrences from very large text collections may become problematic in practice and it usually involves adopting some restrictions. Potentially, any word in the lexicon of a language may co-occur with any other word of that lexicon, so compiling information about word co-occurrences between all pairs of words from a N word lexicon may result in N^2 co-occurrence tuples of the form (w1, w2, count), a number much larger than can be handle by the usual desktop computer using simple approaches. In order to make this process feasible, and since we are not making use of any POS annotation nor stemming techniques, our option was to limit the lexical domain at stake by excluding some lexemes that

we believed would not carry enough information. Therefore, we decided that we would *only consider* co-occurrence pairs that *did not* include (i) any lexemes with less than 4 characters or more than 20 characters, (ii) any lexeme composed of digits, (iii) or any word from a list of 260 very frequent words (manually compiled) that included prepositions, quantifiers, adverbs and very frequent verbs. This allowed us to reduce the lexicon to be processed from about 6.8M lexemes to less that 4M. More importantly, it helped to significantly reduce the number of co-occurrence pairs by excluding very frequently used words that co-occur with a large portion of the lexicon.

The next design decision regards the scope in which co-occurrences are considered. One may opt for compiling document-scope co-occurrences pairs or sentence-scope co-occurrences pairs. We believe, however, that word co-occurrences at document level do not exhibit enough semantic cohesion, and this seems to be especially the case when considering web documents that may have many unrelated sections (e.g. headers, footers and navigation menus). Therefore, we decided to compile co-occurrence pairs inside the scope of a sentence, which is also compatible with the previous decision of storing the text on a sentence basis. Another important design option concerns to whether the information about the distance and/or relative position of the words in the co-occurrence pairs should be kept or not. To avoid data explosion and still keep some useful information we decided that BACO would just keep the information about word order but not the corresponding distance, meaning that we keep different co-occurrences counts for the pair (w1, w2) and for the pair (w2, w1) but we do not keep information about distance.

### 2.3 The database management system

Another practical design decision concerns the database management system to be used. We should remember that BACO is a read-only database: users will query the tables but will not need (nor will be allowed) to insert new tuples or to update values. Therefore, no complex transaction mechanism is required for BACO.

We decided to use the "of-the-shelf" open-source database system MySQL (http://www.mysql.org). Despite other possible options, our previous experience with the MySQL system showed it has a good performance for the kind of task we had in mind. Additionally, MySQL has a good user support and documentation, has versions in many different platforms and it provides database connectors for several programming languages.

### 2.4 Summary of BACO

The overall design of BACO includes:

1. the sentences table, containing all the text of the collection, split into sentences;
2. the metadata table, containing some relevant metadata originally provided in the WPT collection for all documents stored (url, language, etc);
3. the dictionary table, a list of tokens along with the information about the corresponding frequency and the number of documents in which they occur;
4. the n-grams tables. A table of 2-grams, a tables of 3-grams and a table of 4-grams, each keeping information about the n-gram and their frequency and number of documents in which they occur;
5. the co-occurrence table, storing information about pairs of words that co-occur in the scope of a sentence, and about their relative positions.

## 3. Preparing and Loading Data

BACO was generated from a non-annotated subset of the original WPT03 collection, which was obtained by removing possible duplicates based on title and document size comparisons [1]. This subset had roughly 6 GB, contained about 1.5 million documents corresponding to approximately 1000 million words. During the data preparation stage we used a 2.8 GHz Pentium IV machine with 2GB of RAM and 160Gb IDE hard-drive. The machine was running Linux Fedora Core 2. All pre-processing programs were written in Perl 5.6 and we used MySQL version 5.0.15 to store the data. Perl / MySQL interface was done using DBI with the DBD-MySQL driver version 2.9008.

### 3.1 Sentence and Metadata Tables

The 6 GB subset of WTP03 collection was transformed into a very simple relational scheme, by separating the meta-data of each document from its content. The text content was split into sentences, using PT::PLN tools (http://search.cpan.org/~ambs/Lingua-PT-PLN/). We thus obtained two text files in a tabular format that contained information for the two base tables of BACO: a large file containing all the sentences (35,575,103 sentences), and another table containing the metadata for each document (1,529,758 documents). Both files were loaded in the database and indexes were generated for each of the relevant fields. A full-text index was generated for the sentence table. From this moment on, and in order to ensure data consistency, we only used the data from these tables to generate the remaining tables.

### 3.2 Dictionary and N-Grams Tables

Generating the dictionary table was achieved in a single pass over the entire sentence table. The whole lexicon fitted easily into RAM, and the frequency and document count for each entry in the lexicon were obtained using standard Perl hash tables. We obtained a simple text file in tabular format with information about 6,834,420 lexemes. This file was loaded into the database and the corresponding B-Tree indexes were generated. The 2-word lexicon however did not fit into RAM using Perl hash tables. We decided to perform several passes over the sentence table, each of them only collecting 2-grams in which the first word of the 2-gram had a certain number of characters. This is a simple standard way to divide the problem in several disjoint sub-problems, each of them fitting into memory. We performed 13 passes, each one taking approximately 90 minutes. We obtained 13 disjoint 2-gram lists that were concatenated, resulting in a file in tabular format containing 54,610,655 tuples. The file was loaded in the database system and the corresponding B-Tree indexes were generated, taking about 24 hours to complete.

For the 3-gram and 4-gram tables the previous approach was impractical. Since the number of distinct 3-grams and 4-grams is potentially much larger than the number of

[1] Thanks to Nuno Seco, nseco@dei.uc.pt

2-grams, we needed to divide the problem into many more sub-problems than before. However, we had no way of knowing in advance if a given partition would fit in memory or if it would be unbalanced and require further partitioning. We then opted for dividing the problem by calculating the n-gram list for D documents each time. After each iteration the n-grams obtained from a set of D documents were sorted alphabetically and then merged with the ones obtained in the previous iteration, which had been stored in file, also sorted alphabetically. We thus followed the same strategy of the external merge-sort algorithm. This strategy allowed us to incrementally obtain the n-grams counts for the entire collection. However, after processing several thousand documents the size of the temporary file grew to more than 500 Mb, which severely increased the time of merge routine between the tuples in RAM and those in file. To avoid unnecessary performance degradation, in those cases we simply restarted the process with an empty temporary file. In the end, we obtained several intermediate sorted files (4 for the 3-gram lexicon and 6 for the 4-gram lexicon) that were merged in a final step. Tuples were loaded in the database and corresponding B-Tree indexes were generated for all columns of these tables. Although we do not have precise numbers, index generation took more than 40 CPU hours for the 3-gram table and more than 130 CPU hours for the 4-gram table.

### 3.3 The Co-occurrence Table

For compiling the co-occurrence table we followed the same strategy as in the previous cases: we obtained tuples for a set of D documents that were then merged with the tuples obtained from the previous iteration, which had been stored sorted in a temporary file. Again, after several iterations, the temporary file reached gigabyte size and the merge operation became very slow, so the process was restarted with an empty temporary file. We obtained 18 intermediate files, which were then merged in a final operation into a single 15GB file. Obtaining the complete co-occurrence file took an enormous amount of time (by our standards): it took more than 3 weeks of CPU time for compiling the 18 intermediate files and about 48 CPU hours to merge them. Index generation (also B-Tree indexes for each column) took more than 60 CPU hours.

### 3.4 Statistics

Table 1 gives a brief view regarding the size and disk usage of each of BACO's tables.

| Table | # tuples (millions) | Table size (GB) | Index size (GB) |
|---|---|---|---|
| Metadata | 1.529 | 0.2 | 0.05 |
| Sentences | 35.575 | 6.55 | 5.90 |
| dictionary | 6.834 | 0.18 | 0.27 |
| 2-grams | 54.610 | 1.50 | 0.92 |
| 3-grams | 173.608 | 5.43 | 2.97 |
| 4- grams | 293.130 | 10.40 | 6.35 |
| Co-occurrence | 761.044 | 20.10 | 7.56 |
| BACO total | - | 44.4 | ~ 24 |

Table 1. Overview of size and disk usage of BACO.

Storage requirements are huge and data expansion in relation to the size of the collection (around 6GB) is close to a factor of 11. There is a tremendous amount of redundancy involved in BACO. Nevertheless, and as we will show in the next sections, this seems to be an effective tradeoff in exchange for higher performance and simplicity in querying the data.

## 4. Using and Experimenting BACO

BACO is especially suited for three types of queries, namely (i) pattern matching over non-annotated text, (ii) context search using a 1 to 3-word window and (iii) queries regarding co-occurrence data. We have been using BACO in several tasks where there is a practical need to efficiently perform these types of queries.

### 4.1 Helping to Build a Gazetteer

In one recent project we used BACO to find examples of proper names (anthroponyms, toponyms, organizations, etc.) for our online gazetteer REPENTINO (Sarmento et al., 2006). The search was conducted by scanning the sentence table with an array of archetype lexical patterns (e.g.: "X is located in Y") for specific classes of names. Sentences matching these patterns were then parsed to extract the corresponding candidates (i.e "X" or "Y"). After manual validation, positive candidates were stored in REPENTINO database. Similarly, since names of several organizations and events sometimes follow a very regular structure with very speicifc words (e.g.: "Association for *" or "* Club"), we queried BACO for sentences matching those patterns, which again were parsed and subjected to manual validation. This procedure allowed us to find many interesting examples. This was feasible because of the speed provided by the database engine, which allowed us to experiment with many patterns over the whole collection with very reasonable processing times, usually taking from 10 to 150 seconds depending on the frequency of the words in the query and the number of sentences to be retrieved from disk. Differences in query times are related to very practical issues regarding full-text indexes. Full-text indexes do not usually include words with less than four characters because it is assumed that such words are too frequent. In those cases indexes would be almost useless because an enormous number of tuples would still have to be fetched from disk after locating them in the index, thus decreasing performance to nearly the level of a complete sequential scan over the whole text base (which takes approximately 3 minutes on our machine). A less severe, yet still problematic, case occurs for longer but still very frequently occurring words. In those cases MySQL does find the words on the index but a very large number of tuples still needs to be retrieved. This yields search times of more than two minutes. Inversely, performance greatly improves (5-20 seconds) for cases where the search expression includes less frequent occurring words.

### 4.2 Finding Co-hyponyms

In another experiment, we used BACO to automatically expand a set of co-hyponyms given by the user. In other words, starting from a set of elements named seeds, which belong to a certain class, for example ("yellow", "orange", "black") we wanted to find "similar" elements to expand that initial set of seeds, for example ("red", "green", "pink"…). A well-known example of such a function is the Google Sets, available at http://labs.google.com/sets. We were interested in exploring very simple methods,

which would make use of the large amount of information available. One of the methods developed was based on the fact that similar elements occur in the same lexical context. Thus, finding words similar to the ones belonging to the initial set of seeds would be achieved by looking for words that co-occur in the same contexts. In our experiment, for each of the seed words we used the 4-gram table to obtain a list of contexts, which was composed of the three preceding words (there were other possibilities but we only considered this specific 3-word context). From all contexts obtained searching the 4-gram table for all the seed words, we excluded all contexts that co-occurred with a very large number of words (250) because those would probably be very generic contexts. From the remaining contexts we chose only those contexts that co-occurred simultaneously with at least a certain number of seed words. This can be seen as a simple "quality" parameter because contexts that co-occur with all the seed words should be strongly semantically related to them and, therefore, more productive for our goal. Using the selected 3-word contexts (usually more than 20), we again searched the 4-gram table to obtain words that occurred in those contexts. Although no formal evaluation was conducted, we can report (Sarmento, 2006) that results are usually a large set of words containing many "similar" words (Table 2). Each word is given a figure of merit based on the number of contexts in common with the seed words, so that an appropriate cut-off threshold can be established. There are obviously many limitations to this approach but since the whole search procedure usually takes less than 5 minutes, it may be seen as a possible semi-automatic aid for building lexical-semantic resources.

| Seeds | Result sets obtained (top 20) |
|---|---|
| fiat<br>renault<br>toyota | ford (79), opel (75), peugeot (70), volkswagen (64), seat (63), bmw (62), honda (57), audi (55), mercedes (55), citroen (54) … |
| morango<br>laranja<br>banana | maçã (19), pêssego (14), ananás (13), tomate (13), chocolate (13), café (13), amêndoa (12), cenoura (11), limão (11), cereja (11) … |

Table 2. Obtaining "similar" elements using BACO

### 4.3 Clustering Contexts

In another recent experiment we tried to extend the "one sense per collocation" principle (Yarowsky, 1993) to investigate whether the information regarding co-occurrences (not collocations) among words was enough to identify possible word senses. The idea is that for a given sense of a polysemic word, the words that co-occur with the polysemic word under that sense are expected to co-occur among themselves significantly, thus forming contextual clusters. Following this idea, we used BACO to obtain information about co-occurrences and used several clustering algorithms available from the R Statistic Package (http://www.r-project.org) to process that information and to produce some meaningful clusters. For a given polysemic word, BACO is queried to find all possible co-occurrences, and the values of their frequency. Then, BACO is queried again to obtain information about the co-occurrences among each of those words. After calculating the appropriate word association values – using Pointwise Mutual Information (Church and Hanks, 1990) - we are able to obtain a square co-occurrence matrix ready to be processed by the clustering procedures. We have been experimenting with co-occurrence matrixes ranging from 50 to 400 columns, for several polysemic words. Given the amount of information involved, building the co-occurrence matrix can take more than one hour because many thousands of queries are performed. Clustering data similarly takes a long time. Preliminary results show that this clustering strategy leads to some meaningful clusters although the large number of parameters involved in the process makes it difficult to control the final result. An important result is the confirmation that the quality of results greatly depends on the amount (and diversity) of data available. Because of storage limitations, during the initial stages of development the co-occurrence information used corresponded to only a small sample (less the 5%) of the WPT collection. Initial results were disappointing but, interestingly, they immediately improved as soon as we ran exactly the same program over the complete co-occurrence information.

## 5. Conclusion

In this paper we have presented BACO, a large database of text and co-occurrences. We explained the various design decisions taken, we described the pre-processing tasks required for creating the database and we presented several statistics related to BACO. Our tests confirm that generic database systems are a good alternative for dealing with gigabyte text collections, despite the complex pre-processing tasks and system tuning required. We demonstrated the usefulness of BACO in three different tasks, namely in constructing gazetteers, in finding semantically related elements and in implementing word-sense disambiguation techniques based in clustering algorithms.

## 6. Acknowledgments

This work was partially supported by grant SFRH/BD/23590/2005 from Fundação para a Ciência e Tecnologia (Portugal), co-financed by POSI. The author also wishes to thank Luís Cabral (Luis.M.Cabral@sintef.no) for his help in Perl scripting and generic technical support.

## 7. References

Church, Kenneth and Patrick Hanks. 1990. Word association norms, mutual information, and lexicography. Computational Linguistics, 16(1), pp. 22–29.

Davies, Mark. 2003. Relational n-gram databases as a basis for unlimited annotation on large corpora. Proc. of the Workshop on Shallow Processing of Large Corpora 2003, Lancaster. pp. 23-33.

Sarmento, Luís. 2006. A expansão de conjuntos de co-hipónimos a partir de colecções de grandes dimensões de texto em Português. Actas de 1ª Conferência em Metodologias de Investigação Científica. Janeiro 2006. Porto, Portugal.

Sarmento, Luís, Ana Sofia Pinto and Luís Cabral. 2006. REPENTINO – A wide-scope gazetteer for Entity Recognition in Portuguese. 2006. To appear in the Proceedings of the PROPOR Conference 2006.

Yarowsky, David. 1993. One Sense Per Collocation. Proceedings of the ARPA Human Language Technology Workshop.