Using Masks, Suffix Array-based Data Structures and Multidimensional Arrays to Compute Positional Ngram Statistics from Corpora

Alexandre Gil*
Computer Science Department
New University of Lisbon
Caparica, Portugal
agil@pt.ibm.com

Gaël Dias
Centre of Mathematics
Beira Interior University
Covilhã, Portugal
ddg@di.ubi.pt

Abstract

This paper describes an implementation to compute positional ngram statistics (i.e. Frequency and Mutual Expectation) based on masks, suffix array-based data structures and multidimensional arrays. Positional ngrams are ordered sequences of words that represent continuous or discontinuous substrings of a corpus. In particular, the positional ngram model has shown successful results for the extraction of discontinuous collocations from large corpora. However, its computation is heavy. For instance, 4.299.742 positional ngrams (n=1..7) can be generated from a 100.000-word size corpus in a seven-word size window context. In comparison, only 700.000 ngrams would be computed for the classical ngram model. It is clear that huge efforts need to be made to process positional ngram statistics in reasonable time and space. Our solution shows $O(h(F)\ N \log N)$ time complexity where N is the corpus size and $h(F)$ a function of the window context.

1 Introduction

Many models have been proposed to evaluate word dependencies. One of the most successful statistical models is certainly the ngram model (Jelinek, 1990). However, in order to overcome its conceptual rigidity, T. Kuhn *et al.* (1994) have defined the polygram model that estimates the probability of an ngram by interpolating the relative frequencies of all its kgrams ($k \leq n$). Another way to account for variable length dependencies is the n-multigram model designed by Deligne and Bimbot (1995).

All these models have in common the fact that they need to compute continuous string frequencies. This task can be colossal when gigabytes of data need to be processed. Indeed, Yamamoto and Church (2000) show that there exist $N(N+1)/2$ substrings in a N-size corpus. That is the reason why low order ngrams have been commonly used in Natural Language Processing applications.

In the specific field of multiword unit extraction, Dias (2002) has introduced the positional ngram model that has evidenced successful results for the extraction of discontinuous collocations from large corpora. Unlikely previous models, positional ngrams are ordered sequences of tokens that represent continuous or **discontinuous** substrings of a corpus computed in a $(2.F+1)$-word size window context (F represents the context in terms of words on the right and on the left of any word in the corpus). As a consequence, the number of generated substrings rapidly explodes and reaches astronomic figures. Dias (2002) shows that Δ (Equation 1) positional ngrams can be computed for an N-size corpus in a $(2.F+1)$-size window context.

$$\Delta = (N - 2.F) \times \left(1 + F + \sum_{k=3}^{2.F+1} \sum_{i=1}^{F} \sum_{j=1}^{F} C_{j-1}^{i-1} C_{j}^{k-i-1} \right)$$

Equation 1: Number of positional ngrams

In order to illustrate this equation, 4.299.742 positional ngrams (n=1..7) would be generated from a 100.000-word size corpus in a seven-word size window context. In comparison, only 700.000 ngrams would be com-

* The authors want to thank Professor José Gabriel Pereira Lopes from the New University of Lisbon for his advices.

puted for the classical ngram model. It is clear that huge efforts need to be made to process positional ngram statistics in reasonable time and space.

In this paper, we describe an implementation that computes the Frequency and the Mutual Expectation (Dias *et al.* 1999) of any positional ngram with time complexity *O(h(F) N log N)*. The global architecture is based on the definition of **masks** that allow virtually representing any positional ngram in the corpus. Thus, we follow the *Virtual Corpus* approach introduced by Kit and Wilks (1998) and apply a **suffix-array**-like method, coupled to the Multikey Quicksort algorithm (Bentley and Sedgewick, 1997), to compute positional ngram frequencies. Finally, a **multidimensional array** is built to easily process the Mutual Expectation, an association measure for collocation extraction.

The evaluation of our C++ implementation has been realized over the CETEMPúblico[2] corpus and shows satisfactory results. For example, it takes 8.59 minutes to compute both frequency and Mutual Expectation for a 1.092.723[3]-word corpus on an Intel Pentium III 900 MHz Personal Computer for a seven-word size window context.

This article is divided into four sections: (1) we explain the basic principles of positional ngrams and the mask representation to build the *Virtual Corpus*; (2) we present the suffix-array-based data structure that allows counting occurrences of positional ngrams; (3) we show how a multidimensional array eases the efficient computation of the Mutual Expectation; (4) we present results over different size sub-corpora of the CETEMPúblico corpus.

2 Positional Ngrams

In the specific field of multiword unit extraction, Dias (2002) has introduced the positional ngram model that has evidenced successful results for the extraction of discontinuous collocations from large corpora.

2.1 Principles

The original idea of the positional ngram model comes from the lexicographic evidence that most lexical relations associate words separated by at most five other words (Sinclair, 1974). As a consequence, lexical relations such as collocations can be continuous or discontinuous sequences of words in a context of at most eleven words (i.e. *5* words to the left of a pivot word, *5* words to the right of the same pivot word and the pivot word itself). In general terms, a collocation can be defined as a specific[4] continuous or discontinuous sequence of words in a (*2.F+1*)-word size window context (i.e. *F* words to the left of a pivot word, *F* words to the right of the same pivot word and the pivot word itself). This situation is illustrated in Figure 1 for the collocation *Ngram Statistics* that fits in the window context.

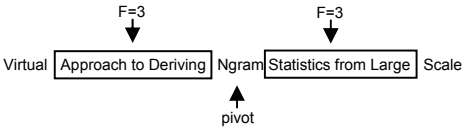

Figure 1: *2.F*-word size window context

Thus, as computation is involved, we need to process all possible substrings (continuous or discontinuous) that fit inside the window context and contain the pivot word. Any of these substrings is called a positional ngram. For instance, [Ngram Statistics] is a positional ngram as is the discontinuous sequence [Ngram ___ from] where the gap represented by the underline stands for any word occurring between Ngram and from (in this case, Statistics). More examples are given in Table 1.

Positional 2grams	Positional 3grams
[Ngram Statistics]	[Ngram Statistics from]
[Ngram ___ from]	[Ngram Statistics ___ Large]
[Ngram ___ ___ Large]	[Ngram ___ from Large]
[to ___ Ngram]	[to ___ Ngram ___ from]

Table 1: Possible positional ngrams

In order to compute all the positional ngrams of a corpus, we need to take into account all the words as possible pivot words.

Figure 2: One-window context for *F=3*

A simple way would be to shift the two-window context to the right so that each word would sequentially be processed. However, this would inevitably lead to duplications of positional ngrams. Instead, we propose a

[2] The CETEMPúblico is a 180 million-word corpus of Portuguese. It can be obtained at http://www.ldc.upenn.edu/.

[3] This represents 46.986.831 positional ngrams.

[4] As specific, we intend a sequence that fits the definition of collocation given by Dias (2002): "A collocation is a recurrent sequence of words that co-occur together more than expected by chance in a given domain".

one-window context that shifts to the right along the corpus as illustrated in Figure 2. It is clear that the size of the new window should be *2.F+1*.

This new representation implies new restrictions. While all combinations of words were valid positional ngrams in the two-window context, this is not true for a one-window context. Indeed, two restrictions must be observed.

Restriction 1: Any substring, in order to be valid, must contain the first word of the window context.

Restriction 2: For any continuous or discontinuous substring in the window context, by shifting the substring from left to right, excluding gaps and words on the right and inserting gaps on the left, so that there always exists a word in the central position *cpos* (Equation 2) of the window, there should be at least one shift that contains all the words of the substring in the context window.

$$cpos = \left\lfloor \frac{2.F+1}{2} \right\rfloor + 1$$

Equation 2: Central position of the window

For example, from the first case of Figure 2, the discontinuous sequence [A B _ _ E _ G] is not a positional ngram although it is a possible substring as it does not follow the second restriction. Indeed, whenever we try to align the sequence to the central position, at least one word is lost as shown in Table 2:

Possible shift	Central word	Disappearing words
[_ _ A B _ _ E]	B	G
[_ _ _ A B _ _]	A	E, G

Table 2: Shifting Substrings

In contrast, the sequence [A _ C _ E F _] is a positional ngram as the shift [_ A _ C _ E F], with C in the central position, includes all the words of the substring.

Basically, the first restriction aims at avoiding duplications and the second restriction simply guarantees that no substring that would not be computed in a two-window context is processed.

2.2 Virtual Representation

The representation of positional ngrams is an essential step towards efficient computation. For that, purpose, we propose a **reference representation** rather than an explicit structure of each positional ngram. The idea is to adapt the *suffix* representation (Manber and Myers, 1990) to the positional ngram case.

Following the suffix representation, any continuous corpus substring is virtually represented by a single position of the corpus as illustrated in Figure 3. In fact, the substring is the sequence of words that goes from the word referred by the position till the end of the corpus.

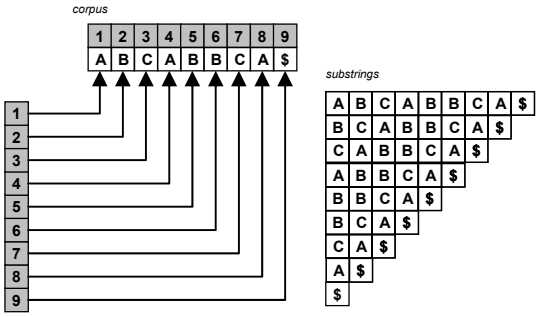

Figure 3: Suffix Representation[5]

Unfortunately, the suffix representation can not directly be extended to the specific case of positional ngrams. One main reason aims at this situation: a positional ngram may represent a discontinuous sequence of words. In order to overcome this situation, we propose a representation of positional ngrams based on **masks**.

As we saw in the previous section, the computation of all the positional ngrams is a repetitive process. For each word in the corpus, there exists an algorithmic pattern that identifies all the possible positional ngrams in a *2.F+1*-word size window context. So, what we need is a way to represent this pattern in an elegant and efficient way.

One way is to use a set of masks that identify all the valid sequences of words in a given window context. Thus, each mask is nothing more than a sequence of 1 and 0 (where 1 stands for a word and 0 for a gap) that represents a specific positional ngram in the window context. An example is illustrated in Figure 4.

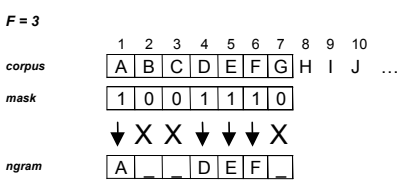

Figure 4: Masks

[5] The $ symbol stands for the end of the corpus.

Computing all the masks is an easy and quick process. In our implementation, the generation of masks is done recursively and is negligible in terms of space and time. In table 3, we give the number of masks *h(F)* for different values of *F*.

F	h(F)
1	4
2	11
3	43
4	171
5	683

Table 3: Number of masks

In order to identify each mask and to prepare the reference representation of positional ngrams, an array of masks is finally built as in Figure 5.

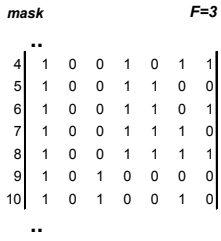

Figure 5: Masks Array

From these structures, the virtual representation of any positional ngram is straightforward. Indeed, any positional ngram can be identified by a position in the corpus and a given mask. Taking into account that a corpus is a set of documents, any positional ngram can be represented by the tuple $\{\{id_{doc}, pos_{doc}\}, id_{mask}\}$ where id_{doc} stands for the document id of the corpus, pos_{doc} for a given position in the document and id_{mask} for a specific mask. An example is illustrated in Figure 6.

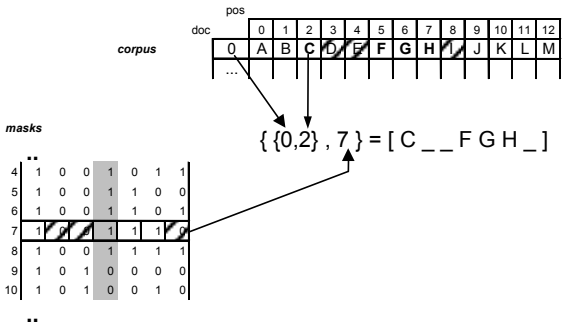

Figure 6: Virtual Representation

As we will see in the following section, this reference representation will allow us to follow the *Virtual Corpus* approach introduced by Kit and Wilks (1998) to compute ngram frequencies.

3 Computing Frequency

With the *Virtual Corpus* approach, counting continuous substrings can easily and efficiently be achieved. After sorting the suffix-array data structure presented in Figure 3, the count of an ngram consisting of any n words in the corpus is simply the count of the number of adjacent indices that take the n words as prefix. We illustrate the *Virtual Corpus* approach in Figure 6.

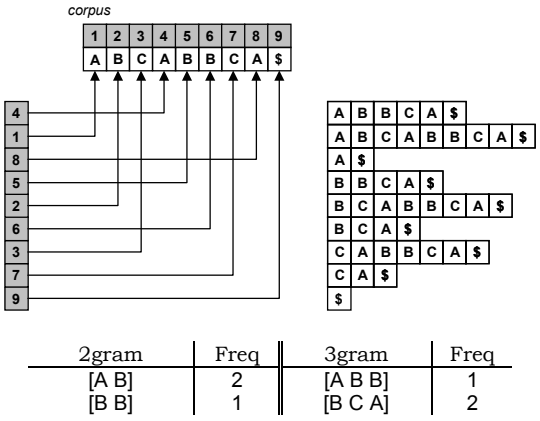

Figure 6: *Virtual Corpus* Approach

Counting positional ngrams can be computed exactly in the same way. The suffix-array structure is sorted using lexicographic ordering for each mask in the array of masks. After sorting, the count of a positional ngram in the corpus is simply the count of adjacent indices that stand for the same sequence. We illustrate the *Virtual Corpus* approach for positional ngrams in Figure 7.

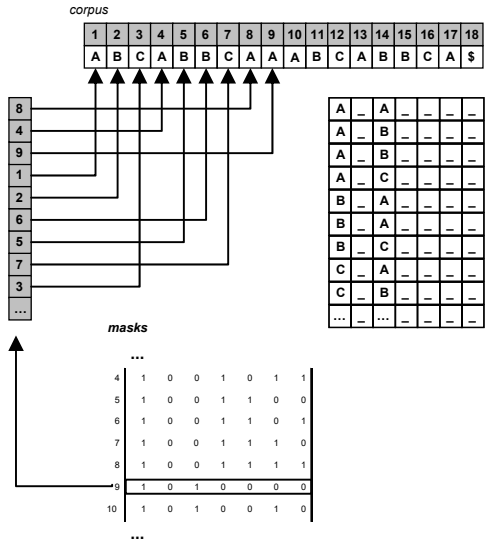

Figure 7: *Virtual Corpus* for positional ngrams

The efficiency of the counting mainly resides in the use of an adapted sort algorithm. Kit and Wilks (1998) propose to use a bucket-radixsort although they acknowledge that the classical quicksort performs faster for large-vocabulary corpora. Around the same perspective, Yamamoto and Church (2000) use the Manber and Myers's algorithm (1990), an elegant radixsort-based algorithm that takes at most $O(N \log N)$ time and shows improved results when long repeated substrings are common in the corpus.

For the specific case of positional ngrams, we have chosen to implement the Multikey Quicksort algorithm (Bentley and Sedgewick, 1997) that can be seen as a mixture of the Ternary-Split Quicksort (Bentley and McIlroy, 1993) and the MSD[6] radixsort (Anderson and Nilsson, 1998).

The algorithm processes as follows: (1) the array of string is partitioned into three parts based on the first symbol of each string. In order to process the split a pivot element is chosen just as in the classical quicksort giving rise to: one part with elements smaller than the pivot, one part with elements equal to the pivot and one part with elements larger than the pivot; (2) the smaller and the larger parts are recursively processed in exactly the same manner as the whole array; (3) the equal part is also sorted recursively but with partitioning starting from the second symbol of each string; (4) the process goes on recursively: each time an equal part is being processed, the considered position in each string is moved forward by one symbol.

In Figure 8, we propose an illustration of the Multikey Quicksort taken from the paper (Bentley and Sedgewick, 1997). The pivot is chosen using the median method.

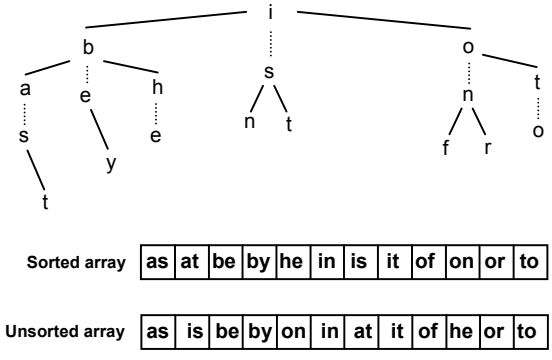

Figure 8: Sorting 12 two-letter words.

[6] MSD stands for Most Significant Digit.

Different reasons have lead to use the Multikey Quicksort algorithm. First, it performs independently from the vocabulary size. Second, it shows $O(N \log N)$ time complexity in our specific case. Third, Anderson and Nilsson (1998) show that it performs better than the MSD radixsort and proves comparable results to the newly introduced Forward radixsort.

Counting frequencies is just a preliminary step towards collocation extraction. The following step attaches an association measure to each positional ngram that evaluates the interdependency between words inside a given sequence. In the positional ngram model, Dias *et al.* (1999) propose the Mutual Expectation measure.

4 Computing Mutual Expectation

4.1 Principles

The Mutual Expectation evaluates the degree of rigidity that links together all the words contained in a positional ngram ($\forall n, n \geq 2$) based on the concept of Normalized Expectation and relative frequency.

Normalized Expectation

The basic idea of the Normalized Expectation is to evaluate the cost, in terms of cohesiveness, of the loss of one word in a positional ngram. Thus, the Normalized Expectation measure is defined in Equation 3 where the function $k(.)$ returns the frequency of any positional ngram[7].

$$NE([p_{11}u_1 \ldots p_{1i}\,u_i \ldots p_{1n}\,u_n]) = \frac{k([p_{11}u_1 \ldots p_{1i}\,u_i \ldots p_{1n}\,u_n])}{\frac{1}{n}\left(k([p_{22}u_2 \ldots p_{2i}u_i \ldots p_{2n}u_n]) + \sum_{i=2}^{n} k\left(\left[p_{11}w_1 \ldots \hat{p}_{1i}\ \hat{u}_i \ldots p_{1n}\,u_n\right]\right)\right)}$$

Equation 3: Normalized Expectation

For that purpose, any positional ngram is defined algebraically as a vector of words $[p_{11}\ u_1\ p_{12}\ u_2\ \ldots\ p_{1n}\ u_n]$ where u_i stands for any word in the positional ngram and p_{1i} represents the distance that separates words u_1 and u_i[8]. Thus, the positional ngram [A _ C D E _ _] would be rewritten as [0 A +2 C +3 D +4 E] and its Normalized Expectation would be given by Equation 4.

[7] The "^" corresponds to a convention used in Algebra that consists in writing a "^" on the top of the omitted term of a given succession indexed from 1 to n.

[8] By statement, any p_{ii} is equal to zero.

$$NE([0\,A+2\,C+3\,D+4\,E]) = \frac{k([0\,A+2\,C+3\,D+4\,E])}{\frac{1}{4}\begin{pmatrix} k([0\,A+2\,C+3\,D])+ \\ k([0\,A+2\,C+4\,E])+ \\ k([0\,A+3\,D+4\,E])+ \\ k([0\,C+1\,D+2\,E]) \end{pmatrix}}$$

which is equivalent to

$$NE([A\_C\,D\,E\_\_]) = \frac{k([A\_C\,D\,E\_\_])}{\frac{1}{4}\begin{pmatrix} k([A\_C\,D\_\_\_])+ \\ k([A\_C\_E\_\_])+ \\ k([A\_\_D\,E\_\_])+ \\ k([C\,D\,E\_\_\_\_]) \end{pmatrix}}$$

Equation 4: Normalized Expectation example

Mutual Expectation

One effective criterion for multiword lexical unit identification is frequency. From this assumption, Dias *et al.* (1999) pose that between two positional ngrams with the same Normalized Expectation, the most frequent positional ngram is more likely to be a collocation. So, the Mutual Expectation of any positional ngram is defined in Equation 5 based on its Normalized Expectation and its relative frequency.

$$ME([p_1 u_1 \ldots p_i u_i \ldots p_n u_n]) = p([p_1 u_1 \ldots p_i u_i \ldots p_n u_n]) \times NE([p_1 u_1 \ldots p_i u_i \ldots p_n u_n])$$

Equation 5: Mutual Expectation

In order to compute the Mutual Expectation of any positional ngram, it is necessary to build a data structure that allows rapid and efficient search over the space of all positional ngrams. For that purpose, we propose a multidimensional array structure called **Matrix**[9].

4.2 Matrix

The attentive reader will have noticed that the denominator of the Normalized Expectation formula is the average frequency of all the positional (n-1)grams included in a given positional ngram. These specific positional ngrams are called positional sub-ngrams of order n-1[10]. So, in order to compute the Normalized Expectation and *a fortiori* the Mutual Expectation, it is necessary to access efficiently to the sub-ngrams frequencies. This operation is done through the Matrix.

[9] The Matrix also speeds up the extraction process that applies the GenLocalMaxs algorithm (Gaël Dias, 2002). We do not present this algorithm due to lack of space.

[10] In order to ease the reading, we will use the term sub-ngrams to denote positional sub-ngrams of order *n-1*.

However, to understand the Matrix itself, we first need to show how the sub-ngrams of any positional ngram can be represented.

Representing sub-ngrams

A sub-ngram is obtained by extracting one word at a time from its related positional ngram as shown in Figure 9.

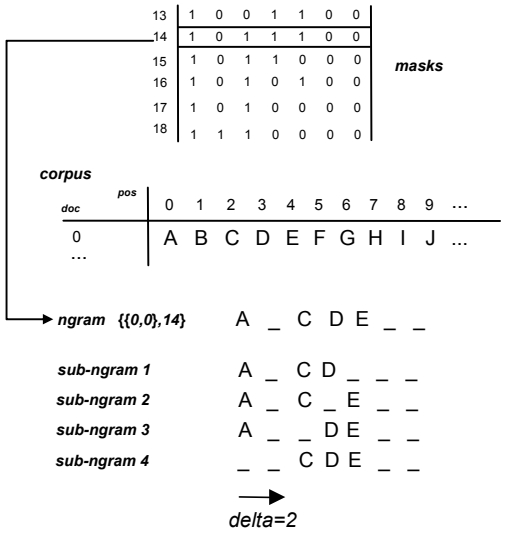

Figure 9: Sub-ngrams

By representing a sub-ngram, we mean calculating its virtual representation that identifies its related substring. The previous figure shows that representing the first three sub-ngrams of the positional ngram {{*0,0*},*14*} is straightforward as they all contain the first word of the window context. The only difficulty is to know the mask they are associated to. Knowing this, the first three sub-ngrams would respectively be represented as: {{*0,0*},*15*}, {{*0,0*},*16*}, {{*0,0*},*13*}.

For the last sub-ngram, the situation is different. The first word of the window context is omitted. As a consequence, in order to calculate its virtual representation, we need to know the position of the first word of the substring as well as its corresponding mask. In this case, the position in the document of the positional sub-ngram is simply the position of its related positional ngram plus the distance that separates the first word of the window context from the first word of the substring. We call *delta* this distance. The obvious representation of the fourth sub-ngram is then {{*0,2*},*18*} where the position is calculated as 0+(*delta*=2)=2.

In order to represent the sub-ngrams of any positional ngram, all we need is to keep track of the masks related

to the mask of the positional ngram and the respective *deltas*. Thus, it is clear that for each mask, there exists a set of pairs {id_{mask}, *delta*} that allows identifying all the sub-ngrams of any given positional ngram. Each pair is called a submask and is associated to its upper mask[11] as illustrated in Figure 10.

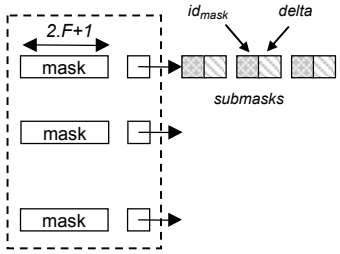

Figure 10: Submasks

Now that all necessary virtual representations are well-established, in order to calculate the Mutual Expectation, we need to build a structure that allows efficiently accessing any positional ngram frequency. This is the objective of the Matrix, a 2-dimension array structure.

2-dimension Array Structure

Searching for specific positional ngrams in a huge sample space can be overwhelming. To overcome this computation problem, two solutions are possible: (1) keep the suffix array-based data structure and design optimized search algorithms or (2) design a new data structure to ease the searching process. We chose the second solution as our complete system heavily depends on searching through the entire space of positional ngrams[12] and, as a consequence, we hardly believe that improved results may be reached following the second solution.

This new structure is a 2-dimension array where lines stand for the masks *ids* and the columns for the positions in the corpus. Thus, each cell of the 2-dimension array represents a given positional ngram as shown in Figure 11. This structure is called the Matrix.

The frequency of each positional ngram can easily be represented by all its positions in the corpus. Indeed, a given positional ngram is a substring that can appear in different positions of the corpus being the count of these positions its frequency. From the previous suffix array-based data structure, calculating all these positions is straightforward.

Calculating the Mutual Expectation is also straightforward and fast as accessing to any positional ngram can be done in $O(1)$ time complexity. We will illustrate this reality in the next section.

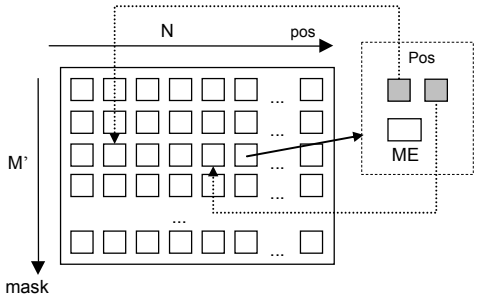

Figure 11: The Matrix

The illustration of our architecture is now complete. We now need to test our assumptions. For that purpose, we present results of our implementation over the CETEMPúblico corpus.

5 Experiments

We have conducted a number of experiments of our C++ implementation on the CETEMPúblico Portuguese corpus to derive positional ngram statistics (Frequency and Mutual Expectation). The experiments have been realized on an Intel Pentium 900 MHz PC with 390MB of RAM. From the original corpus, we have randomly defined 5 different size sub-corpora that we present in Table 4.

corpus	01	02	03	04	05
Size in Mb	0.7	3.1	5.3	6.7	8.8
# of words	114.373	506.259	864.790	1.092.723	1.435.930
# of ngrams[13]	4.917.781	21.768.879	37.185.712	46.986.831	61.744.732

Table 4: Sub-corpora

For each sub-corpus we have calculated the execution time of different stages of the process: (1) the tokenization that transforms the corpus into a set of integers; (2) the preparation of the mask structure and the construction of the suffix-array data structure; (3) the sorting of the suffix-array data structure and the creation of the Matrix; (4) the calculation of the ME. The results are given in Table 5.

[11] The upper mask is the mask from which the submasks are calculated. While upper masks represent positional ngrams, submasks represent sub-ngrams.

[12] In fact, this choice mainly has to do with the extraction process and the application of the GenLocalMaxs algorithm.

[13] The window context of the experiment is F=3.

corpus	01	02	03	04	05
Tokeniz.	0:00:01	0:00:04	0:00:08	0:00:09	0:00:17
Masks/Suffix	0:00:04	0:00:14	0:00:25	0:00:31	0:00:40
Matrix	0:00:35	0:03:23	0:06:16	0:08:11	0:11:12
ME	0:00:00	0:00:03	0:00:06	0:00:08	0:00:10
total	0:00:40	0:03:44	0:06:55	0:08:59	0:12:19

Table 5: Execution Time in (hh:mm:ss)

The results clearly show that the construction of the Matrix and the sort operation over the suffix-array data structure are the most time consuming procedures. On the contrary, the computation of the Mutual Expectation is quick due to the direct access to sub-ngrams frequencies enabled by the Matrix. In order to understand the evolution of the results, we present, in Figure 12, a graphical representation of the results.

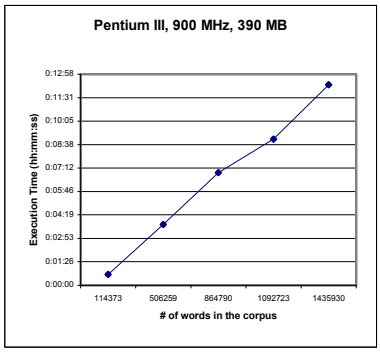

Figure 12: Evolution of execution time

The graphical representation illustrates a linear time complexity. In fact, Alexandre Gil (2002) has proved that, mainly due to the implementation of the Multikey Quicksort algorithm, our implementation evidences a time complexity of $O(h(F)\ N \log N)$ where N is the size of the corpus and $h(F)$ a function of the window context.

6 Conclusion

In this paper, we have described an implementation to compute positional ngram statistics based on masks, suffix array-based data structure and multidimensional arrays. Our C++ solution shows that it takes 8.59 minutes to compute both frequency and Mutual Expectation for a 1.092.723-word corpus on an Intel Pentium III 900 MHz for a seven-word size window context. In fact, our architecture evidences $O(h(F)\ N \log N)$ time complexity. To some extent, this work proposes a response to the conclusion of (Kit and Wilks, 1998) that claims that "*[...] a utility for extracting discontinuous co-occurrences of corpus tokens, of any distance from each other, can be implemented based on this program [The Virtual Corpus Approach]*".

References

Alexandre Gil. 2002. *Extracção eficiente de padrões textuais utilizando algoritmos e estruturas de dados avançadas*. Master Thesis, New University of Lisbon, Portugal.

Arne Anderson and Stefan Nilsson. 1998. *Implementing Radixsort*. ACM Journal of Experimental Algorithmics, Vol. 3. citeseer.nj.nec.com/79696.html

Chunyu Kit and Yorick Wilks. 1998. *The Virtual Approach to Deriving Ngram Statistics from Large Scale Corpora*. International Conference on Chinese Information Processing Conference, Beijing, China, 223-229. citeseer.nj.nec.com/kit98virtual.html.

Gaël Dias, Sylvie Guilloré, and José Lopes. 1999. *Language Independent Automatic Acquisition of Rigid Multiword Units from Unrestricted Text corpora*. Traitement Automatique des Langues Naturelles, Institut d'Etudes Scientifiques, Cargèse, France, 333-339. www.di.ubi.pt/~ddg/publications/taln1999.ps.gz

Gaël Dias 2002. *Extraction Automatique d'Associations Lexicales à partir de Corpora*. PhD Thesis. New University of Lisbon (Portugal) and University of Orléans (France). www.di.ubi.pt/~ddg/publications/thesis.pdf.gz

John Sinclair. 1974. *English Lexical Collocations: A study in computational linguistics*. Singapore, reprinted as chapter 2 of Foley, J. A. (ed). 1996, John Sinclair on *Lexis and Lexicography*, Uni Press.

Jon Bentley and Robert Sedgewick. 1997. Fast Algorithms for Sorting and Searching Strings. 8th Annual ACM-SIAM Symposium on Discrete Algorithms, New Orléans. citeseer.nj.nec.com/bentley97fast.html.

Jon Bentley and Douglas McIlroy. 1993. Engineering a sort function. *Software - Practice and Experience*, 23(11):1249-1265.

Mikio Yamamoto and Kenneth Church. 2000. Using Suffix Arrays to Compute Term Frequency and Document Frequency for All Substrings in a corpus. *Association for Computational Linguistics*, 27(1):1-30. www.research.att.com/~kwc/CL_suffix_array.pdf

Sabine Deligne and Frédéric Bimbot. 1995. *Language Modelling by Variable Length Sequences: Theoretical Formulation and Evaluation of Multigrams*. ICASSP-95. Detroit, Michigan, 1:169-172. citeseer.nj.nec.com/deligne95language.html

T. Kuhn, H. Nieman, E.G. Schukat-Talamazzini. 1994. Ergodic Hidden Markov Models and Polygrams for Language Modelling. ICASSP-94, 1:357-360. citeseer.nj.nec.com/kuhn94ergodic.html

Udi Manber and Gene Myers. 1990. *Suffix-arrays: A new method for on-line string searches*. First Annual ACM-SIAM Symposium on Discrete Algorithms. 319-327. www.cs.arizona.edu/people/udi/suffix.ps