

# INASCO - an incremental associative memory for collocations

Tobias Rothenberger, Sinan Öz, Emin Tahirovic

JW Goethe-University Frankfurt am Main, Dept. of Computer Science

Robert-Mayer-Str. 11-15, 60486 Frankfurt am Main, Germany

Email: {tahirov, oez, rothenb}@cs.uni-frankfurt.de

## Abstract

Since collocations involving main actors represent a big part of the action in the text, examining them is a good way to get an idea about the direction in which the story develops. Creating an incremental memory for collocations which allows a user to decide at what moment he wants to get an insight of the story line is a task we try to fulfil in this paper. We save collocations in separated memory blocks broken down by actors. Through diverse priority functions we can form the aspect of *actuality* of a collocation and give an output consistent with these function values. Our intention is to enable user to reconstruct the recent events from the presented results.

## 1 Introduction

Constructing an incremental associative memory for collocations in a text seems to be an appropriate way of representing the structure of a text and of revealing the information the text is mainly about. In many cases it is useful to know in what direction the action in the text develops and who or what exerts the main influence on the action. The main idea behind the approach is to try to differentiate sentences on the basis of actors/characters that appear in them. We pursue “one actor per sentence” rule to avoid ambiguity, so that each sentence becomes a part of the knowledge base for just one actor. Each actor appearing for the first time receives its own data base. Later we may reconstruct the story line of the text from these knowledge bases and can find out more about single actors by examining interaction among them. Because of the incremental nature of our task we examine one sentence at the time subsequently updating our data base with the new information. In the first section we explain some concepts that are necessary to understand the further elaboration. Then, we give an architectural plan of the process flow that underlies INASCO. Third, we address the implementation, followed by the validation and improvement suggestions.

## 2 Some Terminological Clarifications

In order to understand the motivation and the subject itself, it would be helpful to take some time to examine more closely the specific concepts of the subject definition. Some of these concepts are a subject of research

in computational linguistics while others are from computer science. If we call something incremental, then we refer to its ability to gradually adjust to changes that happen over a period of time so that the present condition reflects these changes. The central concept and furthermore the main tool which will help us in managing our work is the concept of collocation. It is one of the most often stipulated concepts in the theory of natural languages and computational linguistics. The difficulty to reconcile all aspects of collocations in one single definition has occupied many natural language experts and computer linguists over a large period of time. At this point we offer one of all those definitions which gives a description of the collocation concept best suited to our work. “*The term collocation [...] is used for word combinations that are lexically determined and constitute particular syntactic dependencies such as verb-object, verb-subject, adjective-noun relations, etc.*” ([2]) Other definitions can be found in ([1], [3]). The memory concept we would like to present in this paper bears resemblance to canonical concept of the associative memory, in a sense that the derivation of a memory address, for a particular collocation, results from the collocation’s structure/formation itself. Questions similar to our task have already been raised in the domain of computer linguistics particularly with regard to *information* and *discourse* structure. Most of the work that has been done on this field is to present the semantical, temporal and psychological attributes of a discourse in a way that is interpretable by a machine. This differs from our task in the sense that features of our discourse analysis will be presented to a human user who will be hopefully able to get a good picture of the discourse considered. Jäger and Oshima ([4]) use a “T(opic)-tree” model to form a discourse structure acceptable for further machine interpretation. They denote the focus of a sentence as a topic that corresponds to our notion of “actor” or “character” - which has to be identified for each sentence.

## 3 Illustration of our approach

Given a natural language text<sup>1</sup>, we identify main characters (so-called “actors”) in order to understand how they interact. We concentrate on current events which

<sup>1</sup>In this work, we focus on German sentences

takes place in the shorter past. The interpretation of the concept “shorter past” will be defined by a function in dependence to the position of the collocation in the text and the number of times where this collocation appears in the present moment. A detailed elaboration of the priority functions mentioned follows in chapter 4.4. The interaction between main characters can often reveal a coherence about the story line of the text. This brings us to the incremental nature of our work (see Figure 1) that permits an insight into the story line at an user-defined point of time. Each sentence has to pass a preprocessing step, which is done sentence-wise. Here, the sentence is brought to a form that supports the identification of the corresponding actor/character. In the next section, we will give a detailed description of the sentence form we aspire to. After having successfully identified the actor/character that the sentence is about, the information contained in the sentence becomes a part of the memory base that is set up for this particular actor. We keep in mind the information about a story line broken down by actors in the text. The user may decide when he wants to have in what direction the action in the text develops. He will be presented the most newsworthy entries of the memory base according to the already mentioned priority function, for all or for a couple of chosen actors. In this way, we can spare the user from reading the whole text in search for a particular spot or in his efforts to get a good picture of what is the text about. Our solution will be validated in the first step for fairy tales. These texts are simple regarding the linearity of the story line and the structure of the sentences. In particular, the tests and validation of the results will be carried out on the text of “Little Red Riding Hood” (Ger. “Rotkäppchen”) written by The Brothers Grimm.

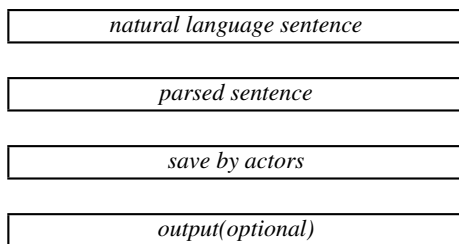


Figure 1: The process flow

## 4 Data and Implementation

### 4.1 Preprocessing

Before we can save a collocation in which a particular actor is involved, each sentence of the text has to go through a preprocessing step. We will assume that this step is completed before we start analyzing the active sentence. Implementation of the preprocessing step can be reali-

zed in cooperation of a stemmer, parser and a human supervisor who controls. We favor the opinion that this is not too much liberty taken in trying to rearrange the input text, so it can be successfully processed later, because our main concern concentrates on steps that follow. Nevertheless we would like to emphasize some of the problems that could turn up during preprocessing. Besides bringing the sentences on a unique form, which allows us easier identification of the character/actor in the sentence and the action he is involved in, we must decide how to deal with some phenomenon like main/subordinate clause, direct/reported speech, active/passive and so on. First, we need to bring all words to their basic form like *hatte*, *hat*, *hätte* to *haben*(stemming). Then, we identify compound sentences and break them down into simpler ones by a syntax parser. In the process, we count - with some loss of information - as a result of dissolving multiple sentences by erasing conjunctions and relying on some corrective actions from the human supervisor. In the case of the directed or reported speech, it would be advisable to embed the statements, which appear between quotations, into the story. This could be accomplished by erasing quotation marks and omitting the notation about the speaker. Interrogative sentences as such carry no information relevant for our task, because it is very hard to determine what is their contribution in forming the information structure of the text. For this reason it would be wise to leave them out as a part of the preprocessing step. We come to the problem of finding the unique form of the sentences. Using the stemmer, we denote each basic form of a word it's category. Subsequently, it would be possible to achieve an unique form for each sentence with a help of a particular Grammar combined with a human supervisor who would be able to take some corrective actions. Passive sentences could be transformed, by the same human supervisor in equivalent, active statements. The problem of detecting the correct reference between the pronouns in the text and actual actors/characters would also be a task for our human supervisor. Additionally we save the position of the sentence/collocation in the text with a help of a simple counter variable which is incremented with each sentence and appended at the end of each collocation. The following is an example, how the preprocessing step looks like. Following an original text like

```

Der Wolf legte sich wieder ins Bett
und fing an zu schnarchen. Der
Jaeger ging vorbei. Er dachte, die
alte Frau schnarcht so laut, da
muss ich einmal nachsehen. Er
trat ein. Im Bett lag der Wolf, den
er so lange gesucht hatte.
  
```

we dissolve in a first round the compound sentences and make simple sentences out of them.

```

Der Wolf legte sich wieder ins Bett.
  
```

Der Wolf fing an zu schnarchen.  
 Der Jaeger ging vorbei.  
 Er dachte.  
 Die alte Frau schnarcht so laut.  
 Ich muss einmal nachsehen.  
 Er trat ein.  
 Im Bett lag der Wolf.  
 Er hatte den Wolf so lange gesucht.

In the second step we bring the words to their basic form and mark their category. In addition the correct reference between the pronouns and the actors is established.

0. Wolf (N) legen (V) Bett (N)
1. Wolf (N) anfangen (V) schnarchen (N)
2. Jaeger (N) gehen (V) vorbei (N)
3. Frau (N) sein (V) alt (Adj)
4. Frau (N) schnarchen (V) laut (Adj)
5. Jaeger (N) nachsehen (V) Haus (N)
6. Jaeger (N) eintreten (V) Haus (N)
7. Bett (N) liegen (V) Wolf (N)
8. Jaeger (N) suchen (V) Wolf (N)

In the next section we will clarify how the collocation “*Frau sein alt*” and its position number was generated.

## 4.2 Collocation form

After the preprocessing step, and as a result of the unique sentence/collocation structure, we are able to assign each sentence to one particular actor. At this point, all sentences are built in a similar way, where the canonical sentence/collocation structure is made up of four parts: a sentence starts with a subject representing the actor (whom the sentence is assigned to), followed by the verb in the sentence, which indicates the relation between the actor and the third part of the collocation. The third part is a generalized notion of an object in the sentence, which is involved in some way with the actor. It is normally a noun or an adjective depending on the way, how the sentence/collocation in question is generated in the preprocessing step. At the end of each collocation, there is a position counter which supports the process of representing collocations in a correct order. The collocation that ends with an adjective was created from a word combination *ADJECTIVE + NOUN* in the original text. The *ADJECTIVE* describes one particular feature of the *NOUN* and in the preprocessing step this yields on a new collocation in which *NOUN* becomes the *actor* and *ADJECTIVE* an *object* of the collocation. The word *sein* takes on the linking role (middle part) in the new collocation and represents it as a fact. The new collocation inherits the collocation number of the sentence, which it was a part of and thus preserves the time flow of the text. This way we achieve the unique sentence structure that we need, and keep the original semantical value of the text.

(“alte Frau”) ⇒ (“*Frau sein alt*”)

## 4.3 Saving information

Having the structure of collocation explained, we focus on how to represent this information in an appropriate way, possibly in combination with a good visualization of the recent events and in the scope of the given situation. For this, we keep a *dynamic list* of each *actor* who appears in text. Each *actor* has one main memory block and one priority list for each priority function. Priority lists save collocations sorted by respective priority function enabling output in accordance with the particular function (see Figure 2). These functions depend, in general, on the present moment and the repetition behavior of a particular collocation. We will discuss priority functions in the next chapter. First we identify the *actor* in the currently processed collocation. After that, it is easy to prove if this is the first appearance in the text. This can be easily done by contacting the dynamic collective *actor* list. If so, a new memory block is allocated for this *actor*. If the actor already exists in the memory, we then use the *verb* (second part) of the collocation as a key to save this collocation in his main memory block. It is possible that this *verb* appears in relation with this *actor* again: then a list for it already exists in the memory. Such list is kept for each *verb* and all we have to do now is to examine if the *object* (third part) can be found in the list. If so, we detected a recurrence of the current collocation. Further steps would be to recompute the priority functions and to sort the priority lists according to the new values. We preserve this way the correct order in the lists which we later use in output for chosen actors.

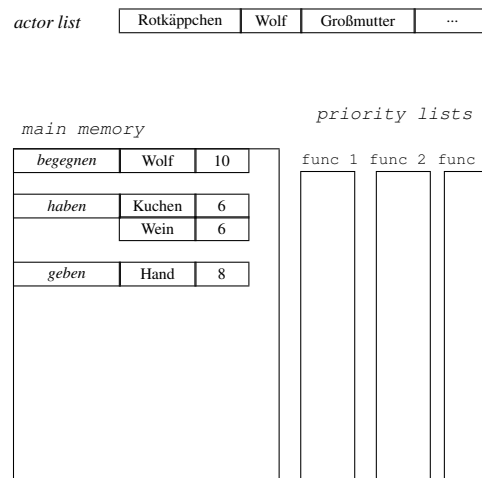


Figure 2: The state of the memory block for the actor “Rotkäppchen” for  $c = 10$ ,  $c$  being the current sentence number

## 4.4 Presenting the results

### 4.4.1 Output

The output is user defined, thus user can decide when the results are to be presented and which activities of which actors is to be seen. The priority lists kept for each *actor*

are a feature that are closely related to the output process. The presentation of the results (showing recent/current collocations broken down by actors) occurs directly from the priority lists. Each priority list has its own priority function by which the collocations are sorted. Dependent on users decision, items for chosen actors are outputted from priority lists. The actual sequence of the presented items depends on the function user chooses. However, not all collocations, where an actor is involved in, will be presented in an output result. Since we try to model the current events in the story line, it is better to present just the most *recent* collocations. In attempt to be able to provide the user insight into each actor over the entire storyline we decided to be less wise and to give a user an opportunity to see at least 5 entries. In addition we define a threshold value  $\Delta = 0.007$  (this is the value which allows the output of 7 most recent collocations according to the priority function, which factors no repetitions in its evaluation) and display additionally those entries (collocations), in actors priority list, that lie above the threshold  $\Delta$ . The results for a particular actor are presented in a separate window, whereas the font size in which the results are written depends on the value evaluated by the priority function (more actual collocations are written in a bigger font size). This should provide a better visualization of the results.

#### 4.4.2 Priority functions

With the idea of priority functions we try to model a concept of oblivion in the program. With the help of the values computed by priority functions we obtain an order among the collocations. For further elaboration it is important to define the notion of *recent* or *current* events. Observing repetition of a particular collocation could increase the importance of the collocation for the story line. Besides chronological features of the collocations, we considered the number of repetitions as a meaningful feature in determining the priority of the collocation. This means that the collocations which have several occurrences in the text, should receive a larger priority than those which occur only once. Of course there are many other features which could seem reasonable in calculating the priority of the collocation, like appearance of a main character in it. We incorporated the impact of repetitions in the calculation of two of our priority functions.

All priority functions should share some important properties in order to model the story line primarily in accordance with its chronological aspects. Such function must be defined on a subset  $D$  of the set of natural numbers  $\mathbb{N}$ . It must be continuous and monotonously decreasing:

$$\forall x, y \in D : x < y \Rightarrow f(x) > f(y).$$

The program provides the user with three priority func-

tions from which he can choose when he decides to see the current output:

1. Let  $\vec{x}^k \in \mathbb{N}^d$  be a dynamic vector of all sentence numbers in which the collocation  $k$  appears,  $d$  is at most the number of all sentences in the text

$$F(c, \vec{x}^k) = \sum_i 0.5^{c-x_i^k}$$

where  $c$  the number of the sentence which is currently examined. We can see that this function incorporates a repetition characteristic of a collocation in the calculation of the priority. The most recent occurrences of the collocation carry more weight in the sum than those which lie far in the past. Because we chose the geometric progression the earlier occurrences will be "*forgotten*" very fast. If a occurrence happened in a very near past it has a relatively big influence on the whole sum. The coefficient of the geometric progression remains 0.5 all through the process.

2. For the next priority function is just the number of repetitions important ( $d$  the dimension of the vector  $\vec{x}^k$  of the first function). In this case we increase with each repetition the coefficient of the geometric progression. The priority is calculated in dependence of the current coefficient  $\hat{a}$ , number of the currently examined sentence  $c$  and the number  $l_k$  of most recent sentence in which a collocation  $k$  appeared.

$$\hat{a} = \sum_{i=1}^d 0.5^i,$$

$$F(\hat{a}, c, l_k) = \hat{a}^{c-l_k},$$

Since we increase the coefficient, with each repetition, priority of the collocations that experience repetitions is uprated. We can see that the step of increase for the coefficient is calculated as a sum of a geometric progression with a start coefficient 0.5. This function rates repetitions higher than the first one since with each repeated occurrence the coefficient changes permanently and the chronological order of repetitions plays no part any more.

3. The third function is the simplest of all three. The repetitions are considered irrelevant. Priority depends on the current sentence number  $c$  and the number  $l_k$  of the last sentence with a particular collocation  $k$  observed.

$$F(c, l_k) = 0.5^{c-l_k},$$

This function provides a way to measure relevance of repetition for calculating the priority of collocations.

On the following example we will show how all three functions calculate the priority :

Exp  $c = 20$

...  
*Wolf sein böse* (sentence 05) - 1. occurrence  
 ...  
*Wolf sein böse* (sentence 15) - 2. occurrence  
 ...  
*Wolf sein böse* (sentence 17) - 3. occurrence  
 ...

1.

$$F_1 = 0.5^{20-5} + 0.5^{20-15} + 0.5^{20-17}$$

$$F_1 = 0.5^{15} + 0.5^5 + 0.5^3$$

$$F_1 \approx 0.15628$$

2.

$$F_2 = (0.5^1 + 0.5^2 + 0.5^3)^{20-17}$$

$$F_2 = 0.875^3$$

$$F_2 \approx 0.66992$$

3.

$$F_3 = 0.5^{20-17}$$

$$F_3 = 0.5^3$$

$$F_3 = 0.125$$

We can already see on this example how repetitions exert a different impact on the respective function evaluation.

## 5 Test and Validation

In this section we present some outputs of the program in order for you to understand better the difference between the priority functions and to visualize the functionality of the program. We will examine the preprocessed text of the german fairytale "Rotkäppchen". In the parenthesis by each actor are priority functions which were used to calculate the priority for the table, with  $c$  we express current sentence number (present moment).

$c=10$

Rotkäppchen		
Priority	Verb	Object
1.0	kommen	Wald
0.25	geben	Hand
0.063	haben	Wein
0.063	haben	Kuchen

Dirne ( $F_1$ )

Priority	Verb	Object
0.016	heissen	Rotkäppchen
0.02	sein	süß
0.02	sein	klein

Großmutter ( $F_1$ )

Priority	Verb	Object
0.5	wohnen	draussen
0.5	wohnen	Wald
0.125	sein	schwach
0.125	sein	krank
0.008	schenken	Kappe

$c=21$

Blumen ( $F_1$ )

Priority	Verb	Object
0.516	sein	schön
0.031	stehen	ringsumher

Blumen ( $F_2$ )

Priority	Verb	Object
0.75	sein	schön
0.031	stehen	ringsumher

Here we can compare the impact of unequal influence of repetition for two priority functions of the same actor. Collocation "*Blumen sein schön*" appeared in 15. as well as in 20. sentence.

$$F_1(21, (15, 20)) < F_2(0.75, 21, 20)$$

We can see here how the second function is more influenced by repetitions. The priorities for "*Blumen stehen ringsumher*" are the same since this collocation occurred only once so far.

$c=22$

Blumen ( $F_1$ )

Priority	Verb	Object
0.258	sein	schön
0.016	stehen	ringsumher

Blumen ( $F_2$ )

Priority	Verb	Object
0.563	sein	schön
0.016	stehen	ringsumher

Blumen ( $F_3$ )

Priority	Verb	Object
0.25	sein	schön
0.016	stehen	ringsumher

We see all three functions for the same actor here. The third function obviously calculates priority without regard to repetition since

$$F_3(22, 20) < F_1(21, (15, 20)) < F_2(0.75, 21, 20)$$

$c=32$

Blumen ( $F_1$ )

Priority	Verb	Object
0.0	sein	schön
0.0	stehen	ringsumher

Blumen ( $F_1$ )

Priority	Verb	Object
0.32	sein	schön
0.0	stehen	ringsumher

Großmutter ( $F_1$ )

Priority	Verb	Object
0.063	sein	schwach
0.063	rufen	—
0.0	wohnen	draussen
0.0	wohnen	Wald
0.0	sein	krank

Großmutter ( $F_2$ )

Priority	Verb	Object
0.316	sein	schwach
0.063	rufen	—
0.0	wohnen	draussen
0.0	wohnen	Wald
0.0	sein	krank

$F_1$  “forgets” collocations which occurred more than once faster than  $F_2$

Rottkäppchen ( $F_1$ )

Priority	Verb	Object
0.031	bringen	Wein
0.031	bringen	Kuchen
0.004	geraten	tief
0.004	geraten	hinein
0.004	geraten	Wald

Wolf ( $F_1$ )

Priority	Verb	Object
1.0	anziehen	Kleider
0.5	verschlucken	Großmutter
0.5	gehen	Bett
0.125	drücken	Klinke
0.016	klopfen	Tür
0.008	gehen	geradewegs
0.008	gehen	Haus
0.008	gehen	Großmutter

$c=36$

Großmutter ( $F_1$ )

Priority	Verb	Object
0.004	sein	schwach
0.004	rufen	—
0.0	wohnen	draussen
0.0	wohnen	Wald
0.0	sein	krank

Großmutter ( $F_2$ )

Priority	Verb	Object
0.1	sein	schwach
0.004	rufen	—
0.0	wohnen	draussen
0.0	wohnen	Wald
0.0	sein	krank

Rottkäppchen ( $F_1$ )

Priority	Verb	Object
1.0	vorkommen	seltensam
1.0	eintreten	Stube
0.5	wundern	sich
0.25	aufmachen	Weg
0.125	herumlaufen	Blumen

We can recognize the geometric progression of the priorities calculated by  $F_1$  for the actor “Rottkäppchen”. None of these collocations occurred more than once. The last of them (or the first by priority) occurred in the sentence number 32. since,

$$F_1(32, (32)) = 0.5^{32-32} = 0.5^0 = 1.0$$

Wolf ( $F_1$ )

Priority	Verb	Object
0.063	zuziehen	Vorhang
0.063	legen	Bett
0.063	aufsetzen	Haube
0.063	anziehen	Kleider
0.031	verschlucken	Großmutter
0.031	gehen	Bett
0.008	drücken	Klinke

## 6 Improvement possibilities

The preprocessing step is obviously a pretty vague step in our approach and there are quite a few actions which provide a more confident preprocessing step without a human supervisor. We concentrate more on the output and presentation of the results in order to have a performing and functional program at the end; as a consequence, the preprocessing actions suffered some lack of attention. Other point needed to emphasize is the dynamic allocation of the *actor* memory blocks. In long texts with a large number of characters this is a waste of memory resources. Thus we suggest the introduction of a *garbage collector* which deletes memory blocks of characters whose priority list appears very *out of date*. This is consistent with the incremental nature of our task. We can also do without three priority list and create them dynamically from current sentence number and the list saving the sentence numbers in which the collocation appeared, when needed. The threshold value could be left for the user to adjust according to his wishes. This way the user could decide how far in the past he wants his “view” to reach.

With a graph as a visualization possibility for the output, we would achieve more intuitive description of relations between the characters in the text. A memory base (thesaurus) with main actors in it would be useful in order to differentiate the behavior of the priority functions for these actors compared with those who appear only marginally in the storyline. This would allow us to know *a priori* the main actors of the story so we could make it harder for the priority function to forget them or perhaps remember them forever.

## 7 Conclusions

In this paper, we introduce a simple way of revealing the action/story line of the natural language text through recovering relations (collocations) between the actors incrementally. Bringing all sentences on a unique form allows us identification of the actor in a particular sentence (one actor per sentence). We save the collocation the actor is involved in together with a sentence number in a memory block allocated for this actor. Priority function sorts the collocations in actors memory block by their actuality. User then decides when he wants to see in what direction the story line develops. He is presented with the most actual collocations actors are involved in, from which he can conclude how the actors interact with each other.

This could be a very useful tool in examining long texts in order to find a particular spot in a story line. Given the time limits of the computer science practical course we did our best to achieve the best performance of the program and each help or advice in improving the lacking aspects are more than welcome.

## A Acknowledgement

Die vorliegende Arbeit fand im Rahmen des Praktikums *Text Mining and Retrieval* im Sommersemester 2006 an der JW Goethe-Universität Frankfurt am Main unter der Leitung von Prof. Dr. Christoph Schommer<sup>2</sup> statt.

## B References

- [1] T. Jäger, D. Oshima : *Towards a Dynamic Model of Topic-Marking*, Stanford University.
- [2] B. Krenn : *CDB - A Database of Lexical Collocations*, Austrian Research for Artificial Intelligence.
- [3] D. Lin : *Extracting Collocations from Text Corpora*, Department of Computer Science, University of Manitoba.
- [4] G. Meyer, M. Läuter, U. Quasthoff: *Learning Relations using Collocations*, Leipzig University, Computer Science Department.

---

<sup>2</sup>University of Luxembourg, Campus Kirchberg, Dept. of Computer Science and Communication, 6 Rue Coudenhove-Kalergi; 1359 Luxembourg, Luxembourg, Email: christoph.schommer@uni.lu