Querying the Syntactic Reference Corpus of Medieval French (SRCMF) with TIGERSearch

Achim Stein

under construction, version of April 18, 2014

PDF version of this document.

Abstract

TIGERSearch is a software for querying syntactically annotated text corpora (tree graphs) and was developed at the Institut für Maschinelle Sprachverarbeitung (IMS) at the University of Stuttgart (Lezius, 2002). It includes a complete manual which however focuses mostly on phrase structure grammars. This document is a guide for querying corpora annotated with the SRCMF dependency model.

HINT: For more information about the corpus and the installation go to the SRCMF homepage.

Contents

1 Introduction

1.1 Installation of TIGERSearch corpora

Case 1: Installation using TIGERRegistry

If you have received the corpus in the TIGER XML format, you must use TIGERRegistry to install it. The TIGERSearch manual explains how to do this. If the XML file is in compressed gzip-format (*.xml.gz) you do not have to uncompress it.

Case 2: Copy a registered corpus

If you receive the corpus in a pre-registered format (normally as a compressed archive, e.g. a *.zip file), unpack this archive and move the resulting directory into the directory `TIGERCorpora` of your TIGERSearch installation (on Windows systems, the default is `C:\TIGERSearch\TIGERCorpora`).

The corpus will be visible in the corpus tree. (If TIGERSearch was active, you may have to re-fresh the corpus tree in TIGERSearch to see the new corpus.)

HINT: You may be able to infer much of the TS query language from the examples given in this document. If not, you should study chapter IV of the TS manual in order to get acquainted with the basics of the query language. The TS manual is included in the tool (click on the *Help* icon); PDF and HTML versions can be found in the `doc` subfolder in the TIGERSearch installation folder.

1.2 The SRCMF dependency model

You should consult the description of the SRCMF dependency model in order to get acquainted with the syntactic categories (see Stein und Prévost 2013 and SRCMF homepage). This document will focus on queries for a limited number of linguistic questions, without however providing detailed explanations of the categories as such.

Please keep in mind that the general strategy here is to use the syntactic annotation, i.e. the tree structure, as much as possible, and to rely as little as possible on the word-level annotation, i.e. part of speech categories and lemmata. This is because the syntactic annotation has been introduced and verified manually, whereas the word-level annotation

- can differ according to the origin of the text (BFM or NCA);
- may have been introduced automatically;
- may not have been verified.

Therefore, in some cases you may find that you can formulate more straightforward queries using word-level information, but if you do so be aware of the risk of retrieving erroneous occurrences or, which is more serious, of overlooking occurrences due to annotation errors.

1.3 Dependency graphs vs TIGER graphs

The following figure is a typical exemple for a dependency structure: the dependency between words is expressed by a labelled edge, where the label expresses the type of the relation (fig. 1):

The dependency annotation that can be represented in TIGER XML does not reflect all the details of the original corpus annotation. The TIGER XML format we refer to has been produced by the export function of the *Notabene* annotation tool (Mazziotta, 2010a,b). Since the current specification of TIGER XML requires words to be terminal nodes, the dependency structure is represented as follows (see fig. 2):

- Each TIGER graph has a top node, mostly Snt, for sentences.
- The dependency relation R between a governing node A and the dependent node B is represented in the TIGER graph by

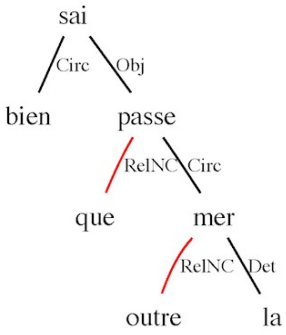

Figure 1: SRCMF dependency structure of *Bien sai que outre la mer passe*

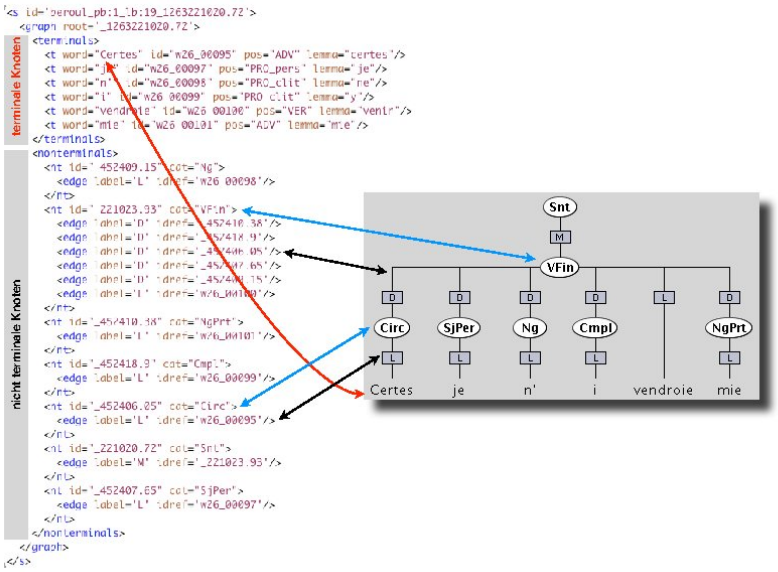

Figure 2: SRCMF dependency structures in TIGER

– a node expressing the relation R in its category value, e.g. VFin for the finite verb;
– an edge labelled L (for "lexical") pointing to the governing node A, e.g. the verb *vendroie*;
– one or more further edges labelled D (for "dependency") pointing to the dependent node(s), e.g. to the subject SjPer

2 TIGERSearch Settings and word-level annotation

Again: the documentation of TIGERSearch gives a detailed account of the possible configuration options. Here, we focus on what is interesting for the SRCMF.

2.1 Features of terminal nodes

The annotation of terminal nodes (i.e. at word level) includes a number of features, of which all may not be of interest. To select what you want to see, open the Graph Viewer window (e.g. by making a query or by klicking on the icon *Explore Corpus*) and select the Menu *Options–Display Options*. In the options window (see Figure 3), select or deselect the features according to your needs.

HINT: If you move the mouse over a terminal node, you still get a popup window with **all** the complete set of features, regardless of your option settings.

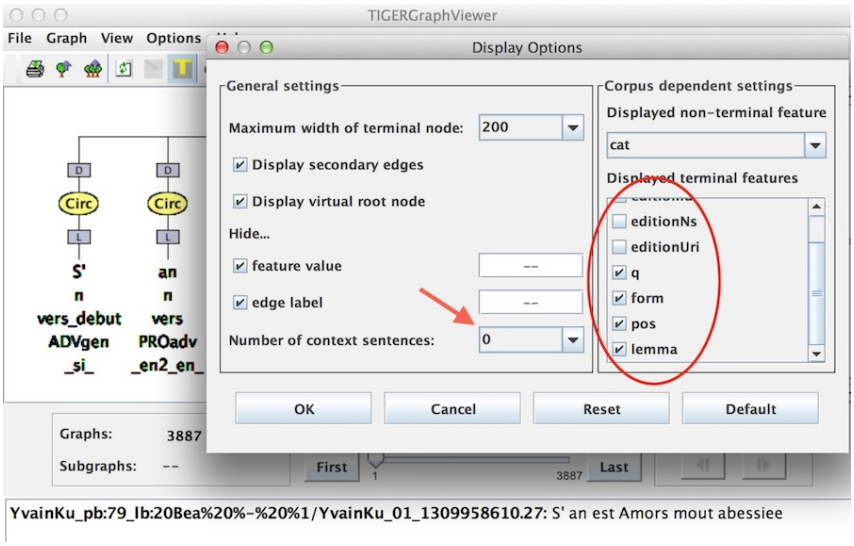

Figure 3: TIGER display options

You can also increase the context which is displayed in the bottom section of the Graph Viewer, by setting the option *Number of context sentences*.

Note that not all the features shown in Figure 3 may be available in your version of the corpus, or in individual texts.

2.2 Some remarks on the "linguistic" features:

pos The part of speech information follows the *Cattex* annotation scheme. The pos tags, if present, of the 15 texts originally published as SRCMF were verified.

lemma If the text contains lemmas, they were introduced automatically using *TreeTagger* and Old French lexical information described in Stein (2003). These lemmas can be helpful for some users, but they were **not verified** and they may ambiguous, i.e. list more than one possible lemma forms, linked by an underscore. If a lemma is followed by a number, this number refers to a subentry in the Tobler-Lommatzsch dictionary.

q (quotation) The default value is `n`. In texts where direct speech is annotated, the value of the attribute `q` is `y` for each word contained in direct speech. This value was calculated automatically. If unknown, the value is `--`.

form According to the position of the word in the verse. The values are:
`vers_debut` for the first word of the verse.

vers for words within the verse.
vers_end for the last word of the verse.
-- if unknown.

3 TIGERSearch queries for SRCMF annotation

3.1 Basics of the TIGERSearch query language

Each node of the graph is an expression between angular brackets. This expression contains one or more attribute-value pairs, e.g. `cat=Obj` for the category Obj or, at the word level, `pos=NOMcom` for the part of speech tag. Two nodes can be concatenated by an operator, e.g. `>` for dominance in the syntactic structure, or by `.` (a dot) for precedence at word level. TS will ignore comments introduced by double slashes.

In order to find out about the attributes which are available in your corpus, you may consult the corpus information in the left part of the window, or use the preview function, symbolized by the looking glass in the top icons, and move the mouse over a node, or over a terminal symbol. This will display the information available at that node for a few seconds.

The values of the attributes have to be enclosed either by quotes (for fully specified values) or by slashes (for regular expressions). The following query (at word level) shows how to use attribute value structures, the precedence operator and comments:

```
// forms starting with "gran" followed by a noun
[word=/gran.*/] . [pos=/NOM.*/]
```

Since an operator combines exactly two node specifications, a series of three nodes *A, B, C* has to be splitted in two separate expressions using the 'and' symbol `&`, e.g. for the precedence relation: `A . B & B . C`

Furthermore, variables have to be introduced if both nodes `B` are meant to refer to the same word (which is normally intended). Variables are introduced attaching their name to a node specification `#name:[ ]` They can be referred to using their name again `#name` The following query finds forms of *demander*, followed by a determiner (*DET:...*) and a noun (*NOM*). The second expression re-uses the node labelled `#det`:

```
[word=/dist.*/] . #det:[pos=/DET.*/]
& #det . [pos=/NOM.*/]
```

Apart from referring to previous expressions, the variables are also necessary to address the node information in the statistics window.

Several attributes of one node can be specified using the `&` operator. The following example searches for words starting with *vil* which are nouns:

```
// nouns starting with "vil"
[word=/vil.*/ & pos=/NOM.*/]
```

3.2 The position of adjectives in the noun phrase

The following query retrieves nominal phrases Nom governing a modifier ModA. The precedence relation (line 4 or 5, comment out as needed) has to bear on the terminal nodes, which are attached by L-edges (lines 2-3):

```
#nom:[type="nV"] > #moda:[cat="ModA"]
& #moda >L #adj:[pos="ADJqua"]
& #nom >L #n:[]
& #adj . #n  // modifier precedes noun
//& #n . #adj     // modifier follows noun
```

HINT: A node specification may remain empty. This is faster than specifying an arbitrary feature value, e.g. `[word=/.*/]`. Just note that in the statistics window, the feature of an unspecified node will not appear automatically when you click on the 'Default' button. If you need the nouns in your statistics, use this version (which differs in line 3)

```
#nom:[type="nV"] > #moda:[cat="ModA"]
& #moda >L #adj:[pos="ADJqua"]
& #nom >L #n:[word=/.*/]
& #adj . #n  // modifier precedes noun
//& #n . #adj     // modifier follows noun
```

3.3 Position of the verb

Verb-second (V2): The following query attempts to retrieve verb-second contexts. It is probably not complete, but it can show some important strategies for TS queries. First of all, precedence is hard to specify at phrase level: something like `[cat="Circ"].[cat="VFin"]` will not produce the intended result.

A safe solution to this problem is to specify precedence only at terminal level (for words). We therefore have to determine the boundaries of a category at word level. To do this, we use the `>@l` and `>@r` operators, which link a node to its leftmost or rightmost terminal nodes.

Solution 1: top-down. In the following query, we search for a sentence-initial category node `#init_cat` [line 4] and define its boundaries [lines 5-6]: `#init_cat` directly precedes the verb if its rightmost terminal node (its last word) directly precedes the verb [line 7]. We use the `>@l` operator again to ensure that the leftmost node of `#init_cat` is also the leftmost node of VFin, and thus exclude further preceding categories:

```
// V2 with verb initial phrases except SjPer, Apst, Ng
#vfin:[cat="Snt" & type="VFin"]  // In a main clause..
& #vfin >L #v:[] // ...get the verb node...
& #vfin > #init_cat:[cat!=/(SjPer|Apst|Ng)/]  //... exclude unwanted initial categories
& #init_cat >@l  #init_cat_init:[]  // define the border nodes of #init_cat
& #init_cat >@r  #init_cat_end:[]
& #init_cat_end . #v  // The last word of #init_cat precedes the verb
& #vfin >@l  #init_cat_init  // The first node of VFin is the left border of #init_cat
```

The previous query is an example for a top-down approach: it specifies the nodes under VFin and goes down to the word level. However, it misses occurrences where the initial category is discontinuous, i.e. only part of the category precedes the verb, as in:

> Mais *Dex* plevis ma loiauté *Qui sor mon cors mete flaele S' onques fors cil qui m' ot pucele Out m' amistié encor nul jor.*

To include these cases, it is safer to procede bottom-up, i.e. determine the word which precedes the verb (here: *Dex*), and go up to its topmost dominating category under VFin.

Solution 2: bottom-up. In the following query, we do the following: (a) specify the path from Snt over VFin to the verb, lines 1-2; (b) determine the word which precedes the verb, line 3, and (c) specify the path up to the highest category `#init_cat` which dominates it under VFin, excluding the categories for subjects, parenthesis and negation which are not relevant for V2 contexts, lines 4-5. Finally we ensure that there is no preceding category by stating that the leftmost node of the VFin, i.e. its first word, is also the leftmost node of `#init_cat`, lines 6-7. Note that this does not exclude sentence-initial conjunctions, like *Et...*, since thy are governed by Snt.

```
// V2 with verb initial phrases except SjPer, Apst, Ng
#vfin:[cat="Snt" & type="VFin"]  // In a main clause..
& #vfin >L #v:[] // ...get the verb node...
& #before_v:[] . #v  // ...then the word preceding the verb...
& #init_cat:[cat!=/(SjPer|Apst|Ng)/] >* #before_v  //...and its category...
& #vfin > #init_cat  // ...directly under VFin (#init_cat).
& #vfin >@l  #vfin_init:[]  // VFin-initial, if the leftmost node of VFin...
& #init_cat >@l  #vfin_init   // ...is also the leftmost node of #init_cat
```

Note that in a sample of 3.300 sentences, the bottom-up solution gives us 90 more occurrences of V2 than the top-down solution.

V3 structures are often said to be counter-evidence for the V2 hypothesis. The following query searches for any kind of sentence-initial phrase (XP: anything except Apst), followed by the Subject in second position and the verb in third position. First, we specify the phrases which depend the verb node VFin. Then the `>@l` and `>@r` operators are used to determine phrase boundaries, and these boundaries are used to define the linear precedence of the phrases. Again, the last statement ensures that XP is the first phrase:

```
// V3  XP-Subject-Verb
#vfin:[cat="Snt" & type="VFin"]  // In a main clause..
& #vfin >L #v:[]
& #vfin > #subject:[cat="SjPer"]
& #vfin > #init_cat:[cat!="Apst"]  // exclude Apst
// determine the left and right phrase boundaries
& #subject >@l  #subject_init:[]  // first word of Subject
& #subject >@r  #subject_end:[]  // last word of Subject...
& #init_cat >@l  #init_cat_init:[]
& #init_cat >@r  #init_cat_end:[]
// define the precedence using the boundaries
& #subject_end . #v
& #init_cat_end . #subject_init
& #vfin >@l #init_cat_init
```

A very similar query can be used to search for verb-final structures. The following query retrieves sentences with the order SOV. Here, category boundaries are specified for both the subject and the object:

```
// Subject-Object-Verb final
#vfin:[cat="Snt" & type="VFin"]  // In a main clause..
& #vfin >@r #v:[]
& #vfin > #subject:[cat="SjPer"]
& #vfin > #object:[cat="Obj"]
// determine the left and right phrase boundaries
& #subject >@l  #subject_init:[pos!=/PROper.*/]  // first word of Subject
& #subject >@r  #subject_end:[]  // last word of Subject...
& #object >@l  #object_init:[pos!=/PROper.*/]
& #object >@r  #object_end:[]
// define the precedence using the boundaries
& #subject_end . #object_init
& #object_end . #v
```

Compare with a MCVF query:

```
// V2 after 1-word or constituent
#ip:[cat="IP"]
& ( #ip > #pos1:[pos!=/(PON|NEG)/]
     | #ip > #xp1:[cat="PP"] & #xp1 >@r #pos1:[word!=/[\*\,].*/] )
& #ip > #v:[pos=/(V.*)/]
& #ip >SBJ #sbj:[cat="NP"]
& #pos1 .* #v // leave room for clitics
& #v .* #sbj
```

3.4 Order of direct and indirect objects

This query looks complex, but it isn't: there are many repitions. The goal is to find out about the order of the direct (Obj) and indirect (Cmpl) objects, with respect to the verb, i.e. the following six orders:

1. Verb - Obj - Cmpl
2. Verb - Cmpl - Obj
3. Obj - Verb - Cmpl
4. Cmpl - Verb - Obj
5. Obj - Cmpl - Verb
6. Cmpl - Obj - Verb

Only the main verb is relevant, i.e. if an auxiliary is present, we consider the position of the objects relative to the **main** verb.

See the comments in the query for more detailed explanations. The last six lines specify the six patterns enumerated above: uncomment the lines for the pattern you want.

```
#vfin:[type="VFin"]
& #vfin:[cat="Snt"]  // restrict to main clauses
&
(
#vfin:[dom!=/.*Aux.*/]
& #vfin >L #2verb:[] // simple main verb
& #vfin >L #1verb:[] // (only for var export)
|
#vfin >L #1verb:[] // auxiliary
& #vfin > #aux:[cat=/Aux.*/]
& #aux >L #2verb:[]  // complex main verb
)
& #vfin > #obj:[cat="Obj" & type!="VFin"] // exclude clauses
& #vfin > #cmpl:[cat="Cmpl" & type!="VFin"]
& #cmpl > #relnc:[cat="RelNC"]
& #relnc > #cmplrel:[]
& #obj >@l #lobj:[] // first word
& #obj >@r #robj:[] // last word
& #obj >L #objh:[pos!="PROper" & pos!="\-\-"] // head word
& #cmpl >@l #lcmpl:[]
& #cmpl >@r #rcmpl:[]
& #cmpl >L #cmplh:[pos!="PROper" & pos!="\-\-"]
//& #2verb .* #lobj & #robj .* #lcmpl // uncomment one
//& #2verb .* #lcmpl & #rcmpl .* #lobj
//& #robj .* #2verb & #2verb .* #lcmpl
//& #rcmpl .* #2verb & #2verb .* #lobj
//& #robj . #lcmpl & #rcmpl .* #2verb
& #rcmpl . #lobj & #robj .* #2verb
```

4 SRCMF annotation for other corpora

4.1 Le Monde

The annotation of this corpus was made with the mate tools parser (Bohnet, 2010), trained on relatively small corpus, and was not verified. The TIGER format was produced by conversion of the parser output (using the script *conll2tiger*).

The annotation differs from SRCMF model in the following respects:

- Determiners are distinguished from modifiers by the category `Det`
- The sentence node is `Snt` by default. (No distinction between **Snt** and `nSnt` is made.)
- Nodes governing verbal heads contain attributes indicating the main verb, `vform` for the form and `vlemma` for the lemma. The main verb is
  - the simple verb
  - the main verb (`Aux`) in the case of complex verbs.
- Terminal nodes have attributes

– for the original part of speech and lemma markup (produced by the TreeTagger): `pos`, `pmor`, `lemma`

– for part of speech and lemma predicted by the *mate tools*: `ppos`, `pmor`, `plemma`

The attributes `mor` and `pmor` result from splitting the original pos tags in the (a) part of speech and (b) the list of morphological features.

The SRCMF queries quoted above can also be applied to this corpus. In addition, the verb annotation can be used to facilitate certain queries. The following example retrieves the direct objects of the verb *prononcer*, regardless of the complexity of the verb form, e.g. for *elle a prononcé* as well as for *elle prononça*:

```
[vlemma="prononcer"] > #obj:[cat="Obj"]
& #obj > #n:[]
```

References

[Bohnet 2010] Bohnet, Bernd: Top Accuracy and Fast Dependency Parsing is not a Contradiction. In: *Proceedings of the 23rd International Conference on Computational Linguistics (Coling 2010)*. Beijing, China : Coling 2010 Organizing Committee, August 2010, S. 89–97. – URL http://www.aclweb.org/anthology/C10-1011

[Lezius 2002] Lezius, Wolfgang: *Ein Suchwerkzeug für syntaktisch annotierte Textkorpora (German)*. Stuttgart : Institut für Maschinelle Sprachverarbeitung (IMS), 2002 (University of Stuttgart Arbeitspapiere des Instituts für Maschinelle Sprachverarbeitung (AIMS), vol. 8, no. 4)

[Mazziotta 2010a] Mazziotta, Nicolas: Building the 'Syntactic Reference Corpus of Medieval French' using NotaBene RDF Annotation Tool. In: *Proceedings of the 4th Linguistic Annotation Workshop (LAW IV)*, URL www.aclweb.org/anthology/W/W10/W10-1820.pdf, 2010

[Mazziotta 2010b] Mazziotta, Nicolas: Logiciel NotaBene pour l'annotation linguistique. Annotations et conceptualisations multiples. In: *Recherches qualitatives. Hors-série 'Les actes'* 9 (2010), S. 83–94

[Stein 2003] Stein, Achim: Étiquetage morphologique et lemmatisation de textes d'ancien français. In: Kunstmann, Pierre (Hrsg.) u. a.: *Ancien et moyen français sur le Web: Enjeux méthodologiques et analyse du discours*. Ottawa : Les Éditions David, 2003, S. 273–284

[Stein und Prévost 2013] Stein, Achim ; Prévost, Sophie: Syntactic annotation of medieval texts: the Syntactic Reference Corpus of Medieval French (SRCMF). In: Bennett, Paul (Hrsg.) ; Durrell, Martin (Hrsg.) ; Scheible, Silke (Hrsg.) ; Whitt, Richard (Hrsg.): *New Methods in Historical Corpora*. Tübingen : Narr, 2013 (Corpus Linguistics and International Perspectives on Language, CLIP Vol. 3), S. 275–282. – ISBN 978-3-8233-6760-4