Chunk Parsing in Corpora

Günther Görz, Günter Schellenberger
University of Erlangen-Nuremberg, Chair of Computer Science 8 (AI)

Abstract
After a brief introduction to the concept of “chunk”, we describe the expectations in (“shallow”) chunk parsing from a syntactic and semantic perspective. For syntactic chunking, the task is separated in two steps, segmentation and prototyping. For segmentation, a sequence of words in a corpus is annotated by “IOB” tags (for inside, outside, and beginning of chunk), where the B tags are augmented by a POS (part of speech) tag. This has to be done either manually or by using an already annotated reference corpus in a semi-automatic fashion. An annotated corpus can be used for training a chunker by well-known techniques from pattern recognition. Setting up transformation-based learning as an iterative process can lead to precision and recall rates of about 94% on unseen corpora. For semantic chunking, additional information about the contents of phrases is required, e.g., by tagging noun phrases with person, location, etc. Experiences with tasks to identify thematic role fillers for verbs as agent, patient, or theme by shallow parsing are still significantly less successful than parsing with “full” syntax.

1 What on Earth is a Chunk?

1.1 Basic Features of Chunks
A fundamental analytical task in Natural Language Processing (NLP) is the segmentation and labeling of texts. In a first step, texts are broken up into sentences as sequences of word forms (tokens). “Chunking” in general means to assign a partial structure to a sentence. “Tagging” assigns labels to the tokens which represent word specific and word form specific information such as the word category and morphological features. Chunk parsing regards sequences of tokens and tries to identify structural relations within and between segments.

Chunk parsing as conceived by Abney [1] originated from a psycholinguistic motivation:

“I begin with an intuition: when I read a sentence, I read it a chunk at a time. For example, the previous sentence breaks up something like this:

(1) [I begin] [with an intuition]: [when I read] [a sentence], [I read it] [a chunk] [at a time]

These chunks correspond in some way to prosodic patterns. It appears, for instance, that the strongest stresses in the sentence fall
one to a chunk, and pauses are most likely to fall between chunks. Chunks also represent a grammatical watershed of sorts. The typical chunk consists of a single content word surrounded by a constellation of function words, matching a fixed template...There is psychological evidence for the existence of chunks...

In the context of corpus linguistics, chunk parsing is regarded as an efficient and robust approach to parsing at the cost of not trying to deal with all of language. Hence, a good coverage on given corpora can be achieved, also in the presence of errors as it is the case with (transcribed) speech. Although the problem is defined on the semantic-pragmatic level, it can be captured automatically only by syntactic means — notice the analogy to the problem of collocations. Chunks are understood as “non-overlapping regions of text, usually consisting of a head word (such as a noun) and the adjacent modifiers and function words (such as adjectives or determiners).” [3]. Technically, there are two main motivations for chunking: to locate information — for information retrieval — or to ignore information, e.g., to find evidence for linguistic generalizations in lexicographic and grammatical research.

The grouping of adjacent words into a single chunk, i.e., a subsequence, should be faithful regarding the meaning of the original sentence.

The sentence

1. The quick brown fox jumps over the lazy dog.

will be represented as: ((FOX) (JUMPS) over (DOG)) where (FOX) and (DOG) represent the noun chunks (“the quick brown fox”) with head “fox” and (“the lazy dog”) with head “dog”, respectively. In a similar manner, (JUMPS) is a one-word verb chunk headed by “jumps”.

The results of chunk parsing are shorter and easier to handle, e.g., for computer aided natural language tools. Briefly, chunk parsing follows a divide-and-conquer principle as illustrated in the following commutative diagram:

\[
\begin{align*}
\begin{array}{c}
 w_1, w_2, \ldots, w_n \xrightarrow{m(\cdot)} LF(\overrightarrow{w_{i,1}}, \\ \overrightarrow{w_{i,2}}, \\ \overrightarrow{w_{i,3}}) \\
\uparrow \downarrow \\
chunk(\cdot) = \text{unchunk}(\cdot) \\
\end{array}
\end{align*}
\]

In this diagram, \(LF(x, y, \ldots)\) represents some logical form, e.g. a predicate like \(jumps-over(\text{who}, \text{over-\text{whom}})\), which can be much easier extracted from the compact sequence on the lower left side of the diagram. Chunking serves as a normalization step; the meaning of the original sentence can be derived from that of the chunked result by substituting the chunks (bottom line) with their origin. Because \(l \ll n\), the implementation of \(m(\cdot)\) is easier.

### 1.2 Chunk Parsing and Full Parsing

In view of examples as (1) above it is intuitive to think of chunk parsing as intermediate step towards full parsing. The type and head of a chunk are taken as non-terminal symbol and literal substitute, respectively, in a recursive process. After the first stage of base segmentation, adjacent chunks and isolated
Proposition 1  **Chunks have a much more simple internal structure than the sequence of chunks inside higher level constructions, including sentences.**

In order to support the intentions mentioned above, some constraints are imposed on chunks:

Proposition 2  **Chunks . . .**

1. never cross constituent boundaries;
2. form true compact substrings of the original sentence;
3. are implemented based on elementary features of the word string, like the POS tags or word types, avoiding deep lexical or structural parameterization incorporated in their implementation;
4. are not recursive.

Rule (3) is required to allow for fast and reliable construction of chunking structures based on their simple nature. A closer inspection of rule (4), which gives a formal definition of simplicity, shows some consequences:

- Chunks do not contain other chunks;
- Recursive rules like the following are not allowed (NC: Noun Chunk):
  \[ NC \rightarrow \text{Det} \ NC \]
- Recursive rule systems, e.g., (where ADVC: Adverbial Chunk)
  \[ ADVC \rightarrow \text{Adj} \ ADVC? \]
  \[ NC \rightarrow \text{Det} \ ADVC? \ N \]
  can be ‘flattened’ to regular structures like
  \[ NC \rightarrow \text{Det} \ \text{Adj}? \ \text{Adj}? \ \text{Adj}? \ N \]
  (‘?’ indicates that the preceding element is optional)

1.3  **Use of Chunks in Spoken Language Processing**

Our work in the area of speech processing systems, including dialogue control, gave us additional motivation for chunk parsing. In the special case of (transcribed) speech corpora, usually we don’t have many well-formed sentences and grammatically perfect constituents. A lot of additional ambiguities come up, because the usually differences in spelling cannot be found, e.g. “May” vs. “may” or the German verb “essen” vs. the German proper noun (cityname) “Essen”. Furthermore, there are no punctuation marks, including marks for the beginning and end of sentences, which again raises many reading ambiguities. This means that dialogue systems have to follow multiple different paths of interpretation. Therefore, search spaces tend to become much bigger and we are facing time and memory limitations due to combinatorial explosion. In
practical systems, recognition errors have to be taken into account and they should be identified as soon as possible.

Base chunks meeting the requirements of Proposition (2) are therefore often the ultimate structures for spoken language systems as input to higher analysis levels which have to assign semantic roles to the parts of chunked input.

To summarize, spoken, reliable syntactic structures other than chunks are often not available in speech processing.

1.4 Limitations and Problems of Chunking in English

Although on a first glance the idea of chunk parsing promises to push natural language understanding towards realistic applications, things are not as easy as they seem.

Named entities such as “John Smith” or “The United Kingdom” should be identified as soon as possible in language processing. As a consequence, named entity recognition has to be included in chunk parsing.

But merging subsequent nouns into the same chunk cannot be introduced as a general rule. In fact, the famous example (1) already contains a trap making it easy for chunk parsers to stumble over:

- “jumps” could be taken as a noun (plural of “jump”) and merged into the preceding noun chunk which is prohibitive for semantic analysis.
- Some more examples of this kind are:
  2 *The horse leaps with joy.*
  3 *This makes the horse leap with joy.*
  4 *Horse leaps are up to eight meters long.*
- Similar considerations hold for named entities comprising a sequence of nouns or terms such as
  5 *System under construction, matter of concern,…*
- Chunk Parsers usually do not wrap compound measurement expressions like
  6 *twenty meters and ten centimeters*

These examples reveal a fundamental problem of implementing rule (1) in Proposition (2): This rule expresses a semantic constraint — how can it be implemented consistently with the other rules? How should a chunk parser respect boundaries of structures which are to be built later based upon its own results?

Problems seizing even deeper show up on closer inspection of verb phrases as “jump over…” or “leap with…” in the examples (1) and (2)–(4), resp.

In example (1), it might not be helpful to merge the preposition “over” with the chunk (DOG) into a prepositional phrase. Here, the preposition qualifies
the verb, not the object. More precisely, “over” qualifies the object’s role: the
dog is is not a location, not a temporal unit, etc., but the passive part of the
predicate.
Again, whether or not a preposition following a word forms a phrasal expres-
sion with that verb and should be merged into a chunk is not decidable by
following the rules for chunks above.
This holds, of course, for verbal expressions spanning several words (e.g., “put
up with” or “add up to”), but also for expressions like “as far as I know”.
Missing or wrong chunking could in such cases lead later processing steps into
trap again.
At least, in English, prepositions or other material that considerably change the
reading of a verb directly follow that verb or are kept close to it.

1.5 Limitations of Chunking in German

The definition of chunks as subsequences of the original sentence provides a
serious coverage limitation for German with its relatively frequent discontinu-
ous structures.
The burden of ambiguities between verbs and nouns in German is somehow
eased by capitalization rules, which of course do not help in the case of speech
processing. In addition, German sentences may start with a verb, leaving sim-
ilar problems as described for English.
In German, it is not common to combine the words which make up geograph-
ical place names with dashes. So, instead of

7 Stratford-upon-Avon

there exist

8 Weil der Stadt

9 Neustadt an der Waldnaab

10 Neuhausen auf den Fildern

But even more horrible: The constraints easing the process of verb phrase
chunking as for English do not hold for German prefixed verbs which are sep-
arable in present tense. On the opposite, the space in between verbal stem and
split prefix does not only allow for constituents of arbitrary length, it even has
to include at least passive objects:

11 Ich hebe Äpfel, welche vom Baum gefallen sind, niemals auf.
I never pick up apples fallen down from the tree.

12 Ich hebe Äpfel niemals auf, welche vom Baum gefallen sind.

13 ?? Ich hebe niemals auf Äpfel, welche vom Baum gefallen sind.

The facts concerning German separable composite verbs in present tense com-
pared with phrasal verbs in English can be extended without exceptions to
auxiliary and modal constructions, including past, perfect and future. This,
amongst other linguistic phenomena, drastically limits the coverage of chunk
parsing for German. However, even for German dialogue systems, chunking
of at least noun phrases is necessary at the beginning of processing.
2 Using Corpora: From Chunking to Meaning

Our introduction suggests that a good start for chunk parsing would be to commence with regular rules. Unfortunately, pure rule-based approaches lack of sufficient coverage. Therefore, to amend the performance of chunk parsing, examples from corpora have to be comprised.

2.1 IOB Tagging

A general and approved approach to solve the problems as introduced in the preceding section is to incorporate annotated samples which are compared against text to be analysed. To make this approach applicable to base chunking, the task of chunk parsing is first transformed into a problem of tagging and prototyping as follows:

- Given a sequence of words \( w_1, w_2, \cdots, w_i, \cdots \)
  assign to it a corresponding sequence of tags \( t_1, t_2, \cdots, t_i, \cdots \) with
  \[
  t_i = \begin{cases} 
  B, & \text{if } w_i \text{ first word of a new chunk,} \\
  I, & \text{if } w_i \text{ inside chunk,} \\
  O, & \text{if } w_i \text{ outside chunk.}
  \end{cases}
  \]
- Optionally, \( B \)-tags are decorated with a syntactic classification: \( B-NP, B-VP \), etc.
- Alternatively, this subclassification is done separately.

2.2 Solutions by Means of Statistical Inference

One class of solutions of the IOB-Tagging problem is to introduce numerical functions based on numerical features of the \( w(i) \):

- For each word in \( w_1, w_2, \cdots, w_i, \cdots \)
  assign a vector \( \overline{v}_i = (v_{i,1}, v_{i,2}, \cdots, v_{i,100}, \cdots) \) of features, i.e. numerical functions.
- Examples of definitions for \( v_{i,j} \):
  - \( v_{i,1} = 1 \), if \( w_i \) is a noun, 0 otherwise
  - \( v_{i,2} = 1 \), if \( w_i \) is a verb, 0 otherwise
  - \( \cdots \)
  - \( v_{i,30} = 1 \), if \( w_i \) = ‘Company’, 0 otherwise
  - POS-tag of \( w_i \)
  - Prefix or suffix
- With the exception of the beginning and end of a sentence or a sequence, the features \( v_{i,j} \) are independent from \( i \)
- Each feature is a function of a window \( w_{i-l}, w_{i-l+1}, \cdots, w_{i+l}, \cdots \) around \( w_i \)
• $t_i$ is expressed as a function of the feature vector assigned $w_i$, parametrized by a parameter vector $\pi$ independent of $i$:

$$t_i = F_{\pi}(v_i)$$

The parameters $\pi$ are calculated as to generate optimum result given a corpus equipped with IOB-tags which in turn are assumed to be correct:

**Proposition 3**

- Given a tagged corpus $R = (w_0, t_0), (w_1, t_1), \ldots, (w_N, t_N)$, called the training corpus,
- find $\pi$ to minimize $\sum \| F_{\pi}(v_i) - t_i \|; i = 0, \ldots, N$

The task of selecting the function $F_{\pi}(.)$ is far from being trivial, and so is the minimization task for the parameters; there are several approaches to find a satisfactory result. In general, the problem is given a geometric interpretation: The vectors of evaluated features are taken as points in a multidimensional space, each equipped with a tag, namely the assigned IOB-tag. The task then is to:

- Identify clusters of points carrying identical tag;
- Express membership to or distance from clusters by appropriate functions.
- Example: The Support Vector Machine SVM separates areas of different tags with hyperplanes of maximum coverage.

### 2.3 Preparing an Annotated Corpus

Calculating is the true proficiency of computers, so as soon as appropriate features $F_{\pi}(.)$ are selected, chunk parsing can be done efficiently as required. But how can the laborious work of preparing a training corpus $R$ be facilitated? There are two main options:

1. An automatically annotated corpus is corrected manually. For the beginning, a chunk parser, based on a few simple rules incorporating basic features, e.g. POS-tags, is used (cf. Proposition (2)). This initial parser is called the “baseline”.

2. Chunk structures are derived from a corpus already equipped with higher level analyses, for example constituency trees (treebank). For an example, cf. Sang and Buchholz [8].

At this place, it is worthwhile to point out that there is no formal and verifiable definition of correct chunking. Tagging a corpus to train a chunker also means to define the chunking task itself.
2.4 Transformation-Based Training

In view of option (2), another approach to the tagging task in general, including IOB-tagging, opens up. The method outlined in the following is called Transformation-based Learning, cf. [7].

1. Start with a baseline tagger.
   Example: Use POS-tags of words alone to define the mapping into IOB-space.

2. Identify the set of words in the input \( w_1, w_2, \ldots \) which have been mistagged.

3. Add one or more rules to correct as many errors as possible.

4. Retag the corpus and restart at step 2 until no or not enough errors remain.

5. Given a sequence of words \( u_1, u_2, \ldots \) outside the corpus to be tagged, do baseline tagging, then apply the rules found in the steps above.

Example: “Adjectives . . . that are currently tagged I but that are followed by words tagged O have their tags changed to O.” (Ramshaw [7], 91)

3 Assessment of Results

3.1 Measuring the Performance of Chunk Parsers

A manually tagged corpus (see \( \mathcal{R} \) in Proposition (3)) is passed to a chunker for automatic tagging.

The performance is measured in terms of Precision and Recall. Informally, precision is the number of segments correctly labeled by the chunker, divided by the total number of segments found by the chunker; i.e., the sum of true positives and false positives, which are items incorrectly labeled as belonging to the class. Recall is defined in this context as the number of true positives divided by the total number of elements that actually belong to the class; (i.e., the sum of true positives and false negatives, which are items which were not labeled as belonging to that class, but should have been.

The formal definitions of precision and recall are as follows:

- **True Positives (TP)**: Segments that are identified and correctly labeled by the chunker;
- **False Positives (FP)**: Segments that are labeled by the chunker, but not in \( \mathcal{R} \);
- **False Negatives (FN)**: Segments that are labeled in \( \mathcal{R} \), but not by the chunker;
- **True Negatives (TN)**: Segments that are labeled neither in \( \mathcal{R} \), nor by the chunker;

**Precision**: Percentage of selected items that were correctly labeled:

\[
\text{Precision} = \frac{TP}{TP + FP}
\]
• **Recall:** Percentage of segments that were detected by the chunker

\[
\text{Recall} = \frac{TP}{TP + FN}
\]

The results of the “CoNLL-2000 Shared Task in Chunking” [8] are still representative for the state of the art: Precision and Recall below 94% for pure IOB-tagging have been achieved. Bashyam and Taira 2007 [2] report on lower results when training a special domain chunker for anatomical phrases.

### 3.2 How far can Statistical Chunking Reach?

At the Conference on Natural Language Learning in 2004, the CoNLL-2004 Shared Task of Semantic Role labeling (SRL) had been introduced. SRL can be understood as the minimum requirement for automated *semantic* analysis of free input text (cf. [5]):

*Who is the agent addressed by the verb of a sentence,*  
*who or what is the patient or instrument,*  
*what adjuncts specify location, manner or cause belonging to the verb?*

The particular challenge was to restrict machinery involved in solving the task to levels below chunk parsing (i.e., words and POS-tags), and more basic chunk parsing applications: Pure segmentation, i.e., IOB-tagging without further labeling of segments, or named entity recognition, i.e., identification of noun chunks representing names of persons, organizations etc. together with a label indicating the type of the entity: person, organization, location, and other.

The results achieved for language-dependent named entity recognition are:

<table>
<thead>
<tr>
<th>Language</th>
<th>Precision</th>
<th>Recall</th>
</tr>
</thead>
<tbody>
<tr>
<td>German</td>
<td>below 84%</td>
<td>below 65%</td>
</tr>
<tr>
<td>English</td>
<td>below 90%</td>
<td>below 90%</td>
</tr>
</tbody>
</table>

As a consequence of the mentioned constraints, the results of the CoNLL-2004 competition can be taken as a realistic orientation mark as far as the applicability of chunk parsing is concerned in the sense of a realistic use of automatic language analysis for whatever specific application.

So, the overall precision of participants in SRL hardly jumps over 75%; for correct identification of the agent of a sentence, precision reaches 94%.

### 4 Conclusion

The authors of the CoNLL-2004 Shared Task [5] conclude:

“...state-of-the-art systems working with full syntax still perform substantially better, although far from a desired behavior for real-task applications. Two questions remain open: Which syntactic structures are needed as input for the task, and what other sources of information are required to obtain a real-world, accurate performance?”
Appendix: Some Publicly Available Chunk Parsers

There are several chunk parsers which can be downloaded for free from the World Wide Web. We present a small selection of those we tried with test data.

One the one hand, there are natural language processing toolkits and platforms such as NLTK (Natural Language ToolKit)\(^1\), or GATE (General Architecture for Text Engineering)\(^2\) which contain part-of-speech taggers and chunk parsers. In particular, NLTK offers building blocks for ambitious readers, who want to develop chunkers or amend existing ones on their own (in Python).

SCP is a Simple rule-based Chunk Parser by Philip Brooks [4], which is part of ProNTo, a collection of Prolog Natural language TOols. A POS tagger and a chunker for English with special features for parsing a “huge collection of documents”\(^3\) have been developed by Tsuruoka and Tsujii [9]. A state-of-the-art pair of a tagger and a chunker, along with parameter files trained for several languages, has been developed at the University of Stuttgart.\(^4\) The “STuttgart Tag Set” STTS has become quite popular in recent years.

Finally, chart parsers running in bottom-up mode and equipped with an appropriate chunk grammar, can be used for chunk parsing as well. This technique has been used in our dialogue system CONALD [6].

Acknowledgement. The authors are indebted to Martin Hacker for critical remarks on an earlier draft of this paper.

References


\(^1\)http://nltk.sourceforge.net/index.php/Main_Page, accessed 31-10-2008
\(^2\)http://gate.ac.uk/, accessed 31-10-2008
\(^3\)available for download from http://www-tsujii.is.s.u-tokyo.ac.jp/ tsuruoka/chunkparser/, accessed 31-10-2008.
\(^4\)available for download from http://www.ims.uni-stuttgart.de/projekte/corplex/TreeTagger/, accessed 31-10-2008.

