- **What is Natural language processing (NLP) ?**

  - **Natural language processing (NLP), as a subfield of Artificial Intelligence, applies computer techniques in linguistic research, thus implying usage of algorithms, data structures, formal models of representation and reasoning, as well as specific AI techniques (especially search and knowledge representation methods).**

  - **NLP ultimately creates a technology, namely an ensemble of processes, methods, techniques, that creates and implements ways of carrying out various tasks referring to natural language.**

- **Natural language engineering deals with the implementation of large-scale systems.**

➢ Applications of NLP

- **Aplications based on text:**
  - ✓ **document classification**
  - ✓ **information retrieval**
  - ✓ **information extraction**
  - ✓ **text understanding**
  - ✓ **knowledge acquisition**
  - ✓ **machine translation and machine aided translation**

- **Applications based on dialogue:**
  - ✓ **human-computer interaction**
  - ✓ **learning systems**
  - ✓ **question answering systems**

and others.

N.B. Note that:

- **Information retrieval has expanded and tends to become an autonomous field (that interacts with NLP).**

- **Speech processing also tends to represent an autonomous field.**

- **It is important to distinguish between natural language understanding and speech recognition. Speech recognition deals only with identifying spoken words that originate from a given signal. It does not deal with message UNDERSTANDING, which remains a task belonging to the field of NLP.**

LEVELS OF NATURAL LANGUAGE PROCESSING

- **The Morphological level**

  - **The word is studied from the grammatical perspective, namely with respect to the variation of its form (inflexion). A change in the form of a word (usually by adding a suffix) indicates a change in its grammatical function.**

  - **At morphological level we assign a part of speech (noun, verb, adjective etc.) to each word. This is done automatically by using a POS-tagger.**

  - **The process of assigning part of speech tags (noun, verb, adjective etc.) to tokens (words) is called POS-tagging.**

(See the corresponding slide referring to data preprocessing).

➢ **The Syntactic level (Syntactical analysis ≡ parsing)**

- **The analysis at this level is organized as taking place between the word – as minimal entity – and its possible combinations: phrases, sentences – as maximal entities.**

  N.B. Here "phrase" is not the Romanian "fraza", but can signify any group of words. Definition: a phrase is an expression consisting of one or more words forming a grammatical constituent of a sentence.

- **The syntactic level of analysis determines the structural role of each word within a sentence.**

- **The process of automatically determining the structure of a sentence is called parsing. The corresponding computer program is a parser.**

- **The type of parsing (syntactical analysis) that we perform depends on the type of grammar that is used.**

- **Example: A generative grammar (Noam Chomsky 1957, 1984) leads to parsing based on constituents; a dependency grammar (Hudson 1984) leads to dependency parsing. (See the lectures on syntactical analysis – coming up!)**

- **N.B. Here a grammar is considered an instrument for knowledge representation concerning a natural language.**

- **To examine how the syntactic structure of a sentence can be computed, you must consider two things: the grammar, which is a formal specification of the structures allowable in the language, and the parsing technique, which is the method of analyzing a sentence to determine its structure according to the given grammar.**

- **The Semantic level (Computational semantics)**

- **The semantic level refers to:**

  - **the meaning of words and sentences;**
  - **the way in which word meanings combine in order to form the meaning of an entire sentence.**

- **Computational semantics performs a context-independent study of meaning. Namely, we are interested in the meaning of a sentence independently of the context in which it occurs.**

- **One problem that arises when performing semantic analysis is that of *word ambiguity* – requiring word sense disambiguation (WSD).**

➢ The Pragmatic Level

- **Pragmatics deals with usage of language in context.**

- **In NLP the pragmatic level studies:**
  - usage of sentences in various contexts;
  - the way in which context influences the interpretation of a sentence.

- **Here the context plays an essential role.**

- **Pragmatic analysis becomes very difficult in the presence of various phenomena that are typical to discourse in natural language. See, for instance, *the reference relation*, as in:**

  John *sustains* he *doesn't believe this.*
  (here reference occurs within the same sentence)

  and

  John *thinks he found* his *hat.*
  (here reference can involve different sentences; "*his*" could refer to John, occurring in the same sentence, or to someone else, who has been mentioned in a different sentence).

  - **This introductory course will guide you to work at the morphological, syntactic and semantic levels of NLP.**

Representing the structure of a sentence

- **The most common way of representing how a sentence is broken into its major subparts, and how those subparts are broken up in turn, is a *tree*.**

- **In order to represent such trees, we introduce the concept of *syntactic phrase* (*grup sintactic*).**

Syntactic phrase

- **A syntactic phrase is a component of the structure of a sentence organized around a head that can be a verb, a noun, adjective, adverb or preposition. The syntactical-semantic cohesion of the group of words representing the syntactic phrase is ensured by grammatical constraints (of type case, topic, agreement) and by the semantic roles imposed by the head on the determinants.**

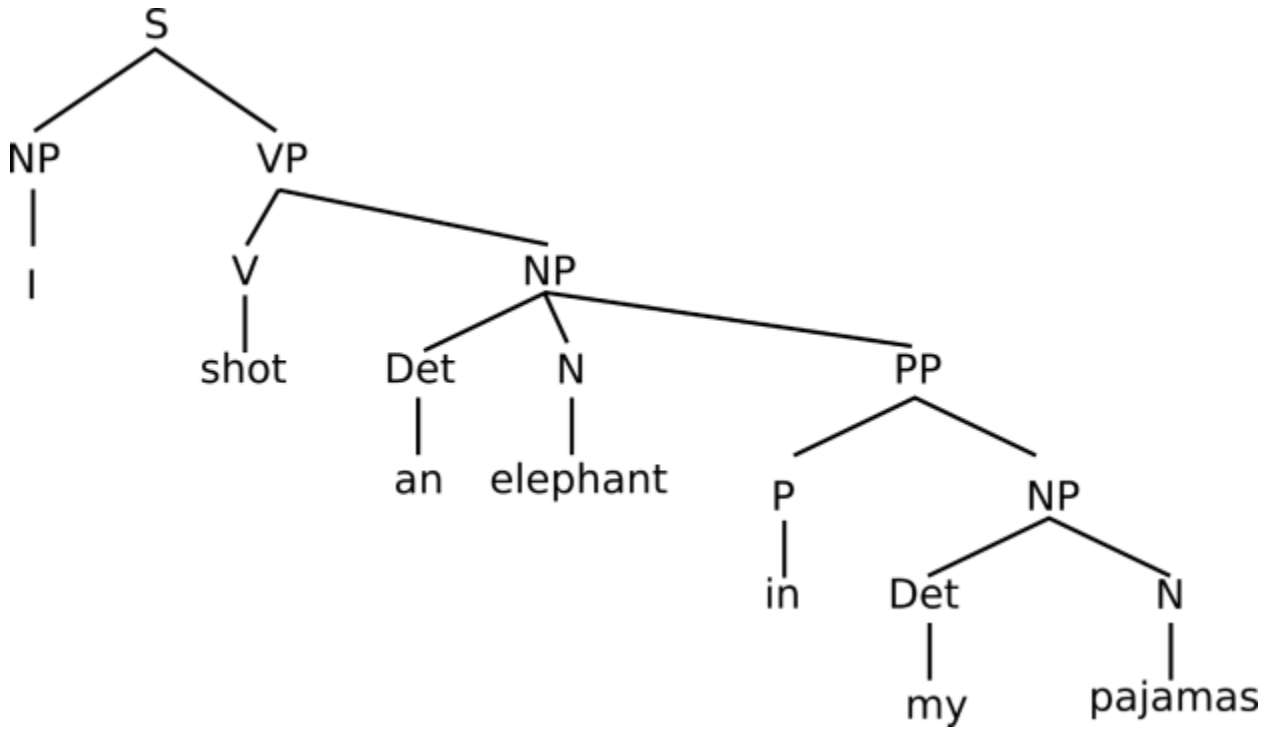
- **According to the morphological class of the head, we distinguish:**
  - **verb phrase – notation VP (the head is a verb)**
  - **noun phrase – notation NP (the head is a noun)**
  - **adjectival phrase – notation ADJP (the head is an adjective)**
  - **adverbial phrase – notation ADVP (the head is an adverb)**
  - **prepositional phrase – notation PP (the head is a preposition)**

- **S is the syntactic phrase that stands for the entire sentence.**

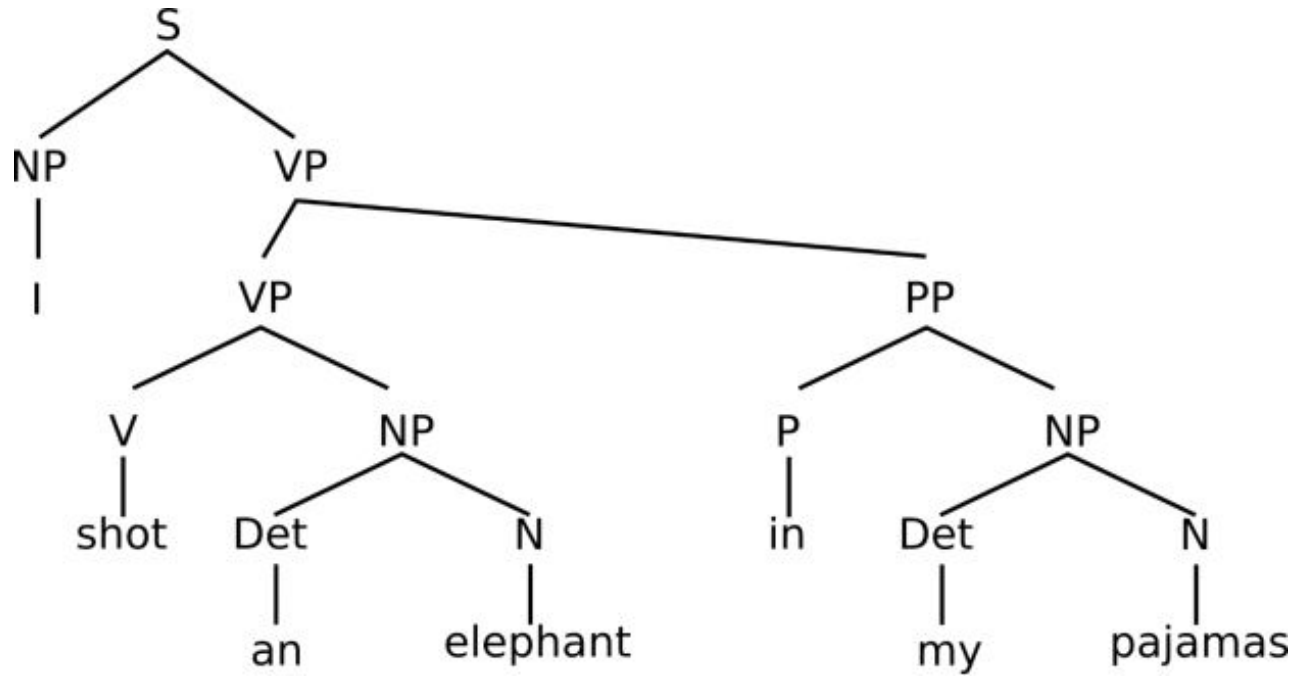
- **Syntactic phrases can be used to represent the syntactic structure of a sentence, as in the following example:**

"*While hunting in Africa, I shot an elephant in my pajamas. How an elephant got into my pajamas I'll never know*".

<u>Parse tree no. 1</u>



Parse tree no. 2



Parse tree no.1 corresponds to the correct sense, which can only be disclosed within the context. Here the elephant is inside the pajamas.

This is an example of ambiguity. The existence of two parse trees denotes structural ambiguity that leads to semantic ambiguity as well.

Ambiguity in natural language processing

- **Ambiguity is a major problem in natural language processing.**

- **There are various kinds of ambiguity an NLP system has to deal with:**

✓ **structural ambiguity - occurs when a parser detects more than one possible structure corresponding to an input sentence (as in the previous example)**

✓ **lexical ambiguity – occurs when a word can have multiple parts of speech as in:**

Rice flies like sand.

✓ **semantic ambiguity – occurs in the case of polysemous words (words having multiple senses) as in the following Romanian sentences:**

Kilogramul de rosii face 10 lei.

Ion face arhitectura.

Andrei face probleme.

- **Note that one type of ambiguity can lead to another. (For instance, lexical ambiguity can lead to structural ambiguity that can lead to semantic ambiguity etc.)**

- **Before conducting any kind of text analysis, there is a <u>data preprocessing</u> step (preprocesarea datelor) that must be performed!**

Data preprocessing

- **Tokenization - splitting the text into tokens (minimal meaningful units); we tokenize a *sentence* into *words* or an *article* into *sentences***
- **POS - tagging – assigning a part of speech to each word (noun, adjective etc.)**
- **Stemming - the process of reducing a word to its stem (ex.: *walking* is reduced to *walk*)**
- **Lemmatization - similar to stemming except it operates also by including the knowledge about the word's context (ex.: *good* and *better*)**
- **NER - Named Entity Recognition – labels a sequence of words that represents the names of things (ex.: a company, a street, a person's name)**
- **Parsing - a parser analyzes the grammar of a text in order to extract or determine its structure**

The difference between stemming and lemmatization

The aim of both processes is the same: reducing the inflectional forms of each word into a common base or root. However, these two methods are not exactly the same.

- **Stemming algorithms work by cutting off the end or the beginning of the word, taking into account a list of common prefixes and suffixes that can be found in an inflected word. This indiscriminate cutting can be successful in some occasions, but not always. Example: form – studies (the verb "study"); suffix – es; stem – studi**

- **Lemmatization takes into consideration the morphological analysis of the words. To do so, it is necessary to have detailed dictionaries which the algorithm can look through in order to link the form back to its lemma. Example: form - studies; morphological information - third person, singular number, present tense of the verb *study*; lemma – study**

Note:

1. **A lemma is the base form of all its inflectional forms (toate "formele flexionare" ale cuvantului), whereas a stem isn't.**
2. **Regular dictionaries are lists of lemmas, not stems. (Lema este forma din dictionar).**

NLP tools and libraries

- **Core NLP (Stanford CoreNLP): contains tools for tagging, parsing, analyzing text; is written in JAVA; supports other languages besides English (Arabic, French, Chinese, German, Spanish etc.)**

- **spaCy: created for industrial strength NLP; provides most mentioned functionalities (POS-tagging, tokenization etc.); supports various languages (German, Spanish, Portuguese, French, Italian, Dutch etc.); the 2017 version has 92,6% accuracy**

- **gensim: for semantic analysis (text mining); the author is a Czech researcher**

- **NLTK – Natural Language Toolkit – "the mother of all NLP libraries": developed since 2001; well maintained and modular; comes with big amount of data, corpora and trained modules; book:**

nltk.org/book

NLTK

Tokenization

- The process of breaking up text into smaller pieces (tokens)
- Token: can be a word or a sentence
- `import nltk` and see demo file

Stop words

- they are removed from the text
- usually they are language specific
- they are words like “then”, “before”, “between”, the” etc.
- these are words that are filtered out before processing natural text
- removing them is part of data processing (a stop word can be dominant in a text and must be removed from the text because it doesn’t import anything to what the text is about)
- with NLTK you import stop words corresponding to the language your text is written in (NLTK provides lists of stop words)
- punctuation must also be removed
- see demo file

POS-tagging

- **the process of assigning part of speech tags (noun, verb, adjective etc.) to tokens (words)**

```
>>> import nltk
>>> from nltk import pos_tag
…
>>> text = nltk.word_tokenize("And now for something
completely different")
>>>nltk.pos_tag(text)
[('And', 'CC'), ('now', 'RB'), ('for', 'IN'),
('something', 'NN'), ('completely', 'RB'), ('different',
'JJ')]
```

where:
CC designates a coordinating conjunction
RB designates an adverb
IN designates a preposition
NN designates a noun
JJ designates an adjective

- **NLTK provides documentation for each tag, which can be queried using the tag**

```
nltk.help.upenn_tagset('RB')
```

Named Entity Recognition

- **NER is used to recognize entities in a text - like people, organisations, places etc.**
- **One must import named entity chunk and also a sentence tokenizer:**

```
from nltk import ne_chunk, sent_tokenize,
word_tokenize
```

where:

sent_tokenize(text) - returns a sentence-tokenized copy of text, using NLTK's recommended sentence tokenizer

word_tokenize(text) - returns a tokenized copy of text, using NLTK's recommended word tokenizer

ne_chunk(tagged_tokens) - Use NLTK's currently recommended named entity chunker to chunk the given list of tagged tokens

- **See demo file**

TF - IDF

TF: term frequency

IDF: Inverse Document Frequency

TF = (number of times the term appears in a document) / (total number of terms in the document)

IDF = log(total number of documents)/(number of documents with the word in it)

TFIDF = TF * IDF

Example: We look at the Brown corpus, coming with NLTK, and we extract the key words

- **See the program TF-IDF.py on the slide "Example TFIDF.docx" included in the folder "Demo file"**
- **The code there uses Porter stemmer from NLTK**
- **For the code to run smoothly you need to install the Python library called `numpy`**
- **The program initializes a list of stop words (for English, taken from NLTK)**

POS-Tagging and Grammars

POS-tagging forms the basis of any further syntactic analyses.

Exploring the In-Built Tagger

```
import nltk
#  Create a variable called simpleSentence and assign a
# string to it
simpleSentence = "Bangalore is the capital of Karnataka."
# Use the nltk built-in tokenizer function called
# "word_tokenize"
wordsInSentence = nltk.word_tokenize(simpleSentence)
# It breaks a given sentence into words and returns a Python
# list data type; we assign it to the variable wordsInSentence
#Display the given data structure (a Python list)
print(wordsInSentence)
['Bangalore', 'is', 'the', 'capital', 'of', 'Karnataka', '.']
# Invoke the nltk built-in tagger, called pos_tag, which takes
# the list of words and identifies the part of speech of each
# word in wordsInSentence
partsOfSpeechTags = nltk.pos_tag(wordsInSentence)
# We see a list of tuples where each tuple has the tokenized
# word and the POS identifier
partsOfSpeechTags = nltk.pos_tag(wordsInSentence)
# invoke the print function
print(partsOfSpeechTags)
```

[('Bangalore', 'NNP'), ('is', 'VBZ'), ('the', 'DT'), ('capital', 'NN'), ('of', 'IN'), ('Karnataka', 'NNP'), ('.', '.')]

Learning to Write Your Own Grammar

```
# import the string module into your own program
import string
# import the generate function from nltk parse.generate
# module
from nltk.parse.generate import generate
# the grammar can contain some production rules; the
# starting symbol is ROOT
productions = [
                "ROOT -> WORD",
                "WORD -> NUMBER LETTER",
                "WORD -> LETTER NUMBER",
                ]
# all production rules are converted to a string
grammarString = "\n".join(productions)
# a new grammar object
# we use the CFG.fromstring method which takes the
# grammarString variable that was just created
grammar=nltk.CFG.fromstring (grammarString)
# print the grammar
print(grammar)
```

Writing a Recursive CFG

(CFG = Context-Free Grammar)

```
productions = [
                "ROOT->WORD"
                "WORD->'  '"
                ]
# Retrieve the list of decimal digits as a list in the alphabets
# variable
alphabets = list(string.digits)
# Using the digits from 0 to 9 we add more productions to
# our list
for alphabet in alphabets:
productions.append("WORD->'{w}'WORD'{w}'".
.format(w = alphabet))
# concatenate all the rules
grammarString = "\n".join(productions)
# create a new grammar object
grammar =  nltk.CFG.fromstring(grammarString)
print(grammar)
```

Syntactic Analysis based on Constituents
(with CFG grammars)

- **CFG Grammars (Context-free grammars):** By convention, the lefthand side of the first production is the start-symbol of the grammar, typically S. All well-formed trees must have this symbol as their root label.

- In NLTK context-free grammars are defined in the nltk.grammar module.

- Example of CFG:

```
grammar1 = nltk.parse_cfg("""
           S-> NP VP
           VP-> V NP| V NP PP
           PP -> P NP
           V -> "saw" | "ate" |"walked"
           NP -> "John" |"Mary" | "Bob" | Det N | Det N PP
           Det -> "a" | "an" | "the" | "my"
           N -> "man" | "dog" | "cat" |"telescope" | "park"
           P -> "in" |"on" | "by" | "with"
           """)
```

- Note that a production like

```
VP -> V NP | V NP PP
```

  has a disjunction on the righthand side, shown by the |, and is an abbreviation for the two productions `VP -> V NP` and `VP -> V NP PP`.

- A CFG grammar is formed of productions (representing the phrase-structure rules) and the lexicon (identical to the vocabulary).

- Parsing a sentence admitted by the grammar (using the recursive descent parser):

```
>>> sent = "Mary saw Bob".split()
>>>rd_parser = nltk.RecursiveDescentParser(grammar1)
>>>for tree in rd_parser.nbest_parse(sent):
...  print tree
(S (NP Mary) (VP (V saw) (NP Bob)))
```

- A grammar is said to be recursive if a category occurring on the lefthand side of a production also appears on the righthand side of the production.

Parsing with Context-Free Grammars

- **A parser processes input sentences according to the productions of a grammar, and builds one or more constituent structures that conform to the grammar.**

- **A grammar is a declarative specification of well-formedness, while a parser is a procedural interpretation of the grammar.**

- **The parser searches through the space of trees licensed by the grammar in order to find one that has the required sentence along its fringe (frontiera).**

- **Classical parsing algorithms:**
  - **the top-down method called "recursive descent parsing"**
  - **the bottom-up method called "shift-reduce parsing"**

- **More evolved parsing techniques:**
  - **a top-down method with bottom-up filtering called "left-corner parsing"**
  - **a dynamic programming technique called "chart parsing"**

Recursive descent parsing

(top-down)

- **Each production rule gets a procedural interpretation:**

```
S -> NP VP
```

can be interpreted as: "in order to analyze (isolate) an S, first analyze (isolate) an NP, then a VP".

```
N -> mother
```

can be interpreted as: "in order to analyze (isolate) an N (noun), accept the word *mother* from the input string".

- **The resulting process is a *recursive descent* (each production rule is turned into a procedure and the resulting procedures call each other).**

- **The top-down strategy starts with the S symbol and tries to rewrite the symbols until it reaches the input sentence (if this sentence is accepted by the given grammar).**

- **The parser processes the input string by accepting its words on a one-by-one basis.**

Analyzing a constituent of type C

If C represents the type of constituent that the parser is looking for at a given time of the analysis (S, NP, V etc.), then the algorithm for analyzing a constituent of type C is the following:

Algorithm 1

- **If C represents an individual word, it is looked up in the lexical rules of the grammar and it is accepted from the input string.**
- **Otherwise, the phrase-structure rules of the grammar are browsed, in order to rewrite C as a list of constituents. These constituents are then analyzed on a one-by-one basis.**

The State of the Analysis

- **The state of the analysis is a pair consisting of the symbol list (or list of symbols) and a number indicating the current position within the input string.**

- **The symbol list indicates the result of the processing operations having taken place until the current moment.**

  Example: The parser starts in the state (S). After applying the phrase-structure rule `S -> NP VP`, the symbol list becomes `(NP, VP)`. If immediately after the rule `NP -> ART N` is applied, then the symbol list becomes (ART N VP).

- **Example of the state of the analysis**

Input string:

$_1$ *Colentina* $_2$ *angajează* $_3$ *surori* $_4$

State of the analysis :

((N VP)1)

Significance of this state of the analysis :

The parser must find an N followed by a VP starting from position 1.

Top-down Parser

- **The parser must systematically explore all possible new states of the analysis (obtained by applying all phrase-structure rules and lexical rules of the grammar).**

- **All possible new states will be generated at each step. One of them will be chosen as representing the next state. The others will be saved as backup states.**

- **If the current state can not lead to a solution, a new current state is chosen from the list of backup states.**

- **The algorithm works with a list of possible states – the list of possibilities. The first element of this list is the current state, representing the state of the analysis. The other elements of the list represent the backup states.**

- **This type of syntactic analysis based on constituents can be viewed as a search problem. In the case of a depth-first strategy, the list of possibilities is organized as a stack. In the case of a breadth-first strategy, the list of possibilities is organized as a queue.**

EXAMPLE:

Consider the following context-free grammar (CFG):

$S \rightarrow NP\ VP$

$NP \rightarrow ART\ N$

$NP \rightarrow ART\ ADJ\ N$

$VP \rightarrow V$

$VP \rightarrow V\ NP$

together with the lexicon

latră: V

câine: N

un: ART

and the input string

$_1 Un\ _2 câine\ _3 latră\ _4$

We perform top-down parsing in this case, implementing a search strategy of type depth-first. The steps of the analysis are the following :

$_1$ *Un* $_2$ *câine* $_3$ *latră* $_4$

Step	Current state	Backup states	Comments
1.	((S)1)	-	Initial position
2.	((NP VP)1)	-	Rewrite S with rule 1
3.	((ART N VP)1)	((ART ADJ N VP)1)	Rewrite NP with rules 2 and 3
4.	((N VP)2)	((ART ADJ N VP)1)	ART is reduced with "Un"
5.	((VP)3)	((ART ADJ N VP)1)	N is reduced with "câine"
6.	((V)3)	((V NP)3) ((ART ADJ N VP)1)	Rewrite VP with rules 4 and 5
7.	()4)	((V NP)3) ((ART ADJ N VP)1)	V is reduced with "latră"
8.	STOP	-	Correct sentence

Algorithm 2 (top-down parsing) - Romanian

Se pleacă din starea iniţială ((S)1), fără stări backup. Paşii algoritmului sunt:

1. *Selectează* starea curentă: se scoate prima stare din lista de posibilităţi; fie ea starea C. Dacă lista de posibilităţi este vidă, atunci algoritmul eşuează (i.e. nu se poate face cu succes analiza sintactică, în sensul că secvenţa de intrare este respinsă ca nefiind conformă cu gramatica limbajului) şi STOP.

2. Dacă C are o listă de simboluri vidă, iar contorul indică poziţia de la sfârşitul propoziţiei, atunci algoritmul are succes (i.e. propoziţia este acceptată ca fiind corectă) şi STOP.

3. Altfel, *generează* următoarele stări posibile. (Adăugarea acestor stări în lista posibilităţilor se va face în funcţie de modul de organizare al listei, adică de modul în care se efectuează căutarea).

3.1. Dacă primul simbol din lista de simboluri a lui C este unul lexical şi următorul cuvânt al propoziţiei aparţine acelei categorii lexicale, atunci se creează o nouă stare prin înlăturarea primului simbol din lista de simboluri şi repoziţionarea contorului de poziţie. Starea nou creată se adaugă listei posibilităţilor.

3.2. Altfel (i.e. primul simbol din lista de simboluri a lui C este un neterminal), se generează o nouă stare corespunzător fiecărei reguli a gramaticii care poate rescrie acest neterminal. Se adaugă listei posibilităţilor noile stări generate.

Algorithm 2 (top-down parsing) - English

We start from the initial state ((S)1), with no backup states. The steps of the algorithm are the following:

1. ***Select*** **the current state: extract the first state from the list of possibilities; let it be state C. If the list of possibilities is empty, then the algorithm fails (the syntactic analysis can not be successfully performed in the sense that the input string is rejected as not being created according to the given grammar) and STOP.**
2. **If C has an empty list of symbols, and the current position is at the end of the input string, then the algorithm is successful (i.e. the input string is accepted as being correctly created) and STOP.**
3. **Otherwise,** ***generate*** **the next possible states. (The way in which these states are added to the list of possibilities depends on how this list is organized, namely on the type of search that is being performed).**

3.1. If the first symbol belonging to the symbol list of C is a lexical one, and the next word in the input string belongs to that lexical category, then create a new state by removing the first symbol from the list of symbols and by moving to the next position in the input string. The newly created state is added to the list of possibilities.

3.2. Otherwise (i.e. the first symbol of C's symbol list is a nonterminal), generate a new state corresponding to each rule of the grammar that can rewrite this nonterminal. Add the newly generated states to the list of possibilities.

The Need for More Parsing Techniques

Problem: When performed from left to right, top-down parsing encounters trouble with rules that exhibit *left recursion*.

Example:

`VP -> VP NP` **(rule for a complex verb phrase)**

This rule is left recursive in that the first symbol on the right hand side (RHS) is the same as the left hand side (LHS).

With left-recursive rules in a grammar, a left-to-right, top-down parser will get into an infinite loop if it makes the wrong choices. Additionally, if we are asking for all possible parses, it will have to try all possible choices – and will in the end get into an infinite loop.

BOTTOM-UP PARSING (SHIFT-REDUCE PARSER)

(Parser cu deplasare-reducere)

- **Bottom-up parsers try to find (within the input string) sequences of words and phrases that correspond to the righthand side of a grammar production and replace them with the lefthand side, until the whole sentence is reduced to an S.**

- **A simple kind of bottom-up parser is the shift-reduce parser.**

- **Action: the shift-reduce parser repeatedly pushes the next input word onto a stack – this is the *shift* operation. If the top *n* items on the stack match the *n* items on the righthand side of some production (phrase-structure rule of the grammar), then they are all popped off the stack, and the item on the lefthand side of the production is pushed onto the stack. This replacement of the top *n* items with a single item is the *reduce* operation.**

- **The *reduce* operation may be applied only to the top of the stack; reducing items lower in the stack must be done before later items are pushed onto the stack.**

- **End of parse: the parser finishes when all the input is consumed and there is only one item remaining on the stack, a parse tree with an S node as its root or just the S node.**

Example of shift-reduce parsing

Un câine latră.

Pas	Acţiune	Stivă	Şir de intrare
	START		*un câine latră*
1	**Shift**	*un*	*câine latră*
2	**Reduce**	*ART*	*câine latră*
3	**Shift**	*ART câine*	*latră*
4	**Reduce**	*ART N*	*latră*
5	**Reduce**	*NP*	*latră*
6	**Shift**	*NP latră*	-
7	**Reduce**	*NP V*	-
8	**Reduce**	*NP VP*	-
9	**Reduce**	*S*	-

- **There are two kinds of choices to be made by the parser:**
- **which reduction to do when more than one is possible**
- **whether to shift or to reduce when either action is possible**

- **A shift-reduce parser can be extended to implement policies for resolving such conflicts. For example:**
- **Shift-reduce conflicts - shifts only when no reductions are possible.**
- **Reduce-reduce conflicts - favors the reduction operation that removes the most items from the stack.**

<u>Advantages</u> of shift-reduce parsers over recursive descent parsers:

- **they only build structure that corresponds to the words in the input;**
- **they only build each substructure once.**

Advantages and Disadvantages

- **Shift-Reduce parsing can deal with productions displaying left recursion, unlike Recursive descent parsing. This is so because its actions are determined solely by words actually occurring in the input sentence.**

- **Shift-Reduce parsing suffers from another limitation: it is not able to deal with productions of type**

$$A \rightarrow \Phi$$

since it has no way of reacting when faced with a null constituent (a constituent that is missing).

Example for Romanian:

NP → Det N
Det → ϕ
corresponding to
"*un baiat*" and "*baiatul*", respectively.

- **A *better solution*:**

the left-corner parser – representing a top-down parser with bottom-up filtering (coming up in the next lecture!)

Shift-Reduce Parsing with NLTK

- **NLTK provides**

  `ShiftReduceParser()`

  which is a simple implementation of a shift-reduce parser.

- **This parser does not implement any backtracking, so it is not guaranteed to find a parse for a text, even if one exists. Furthermore, it will only find at most one parse, even if more parses exist.**

- **Recommended programming language for shift-reduce parsing: PROLOG – which incorporates the backtracking process.**

Left-corner parsing - a top-down method with bottom-up filtering

(Analiza sintactica din coltul stang)

- **Problems:**
  - **recursive descent parser – goes into an infinite loop when it encounters a left-recursive production (because it applies the grammar productions blindly, without considering the actual input sentence).**
  - **shift-reduce parser – does not know how to treat null (missing) constituents**

- **Solution: a hybrid between these top-down and bottom-up approaches**

- **The idea behind left-corner parsing: accept a word from the input string, determine the type of constituent that starts with the corresponding category and parse the rest of that constituent in a top-down manner.**

- **Remark: the parse tree is "revealed" starting from the bottom left corner.**

Algorithm (analyzing a constituent of type C)

1. **Accept a word from the input string and determine its category. Let this category be W.**

2. **Fill in C such that: if W = C, then STOP; otherwise:**
   - **using the grammar phrase-structure rules, find a constituent having an expansion (right-hand side) that starts with W; let P be this constituent;**
   - **recursively analyze, using the top-down strategy, all remaining elements of P's expansion;**
   - **substitute W by P and go to the beginning of step 2 (in order to continue the completion of C).**

Example:

For instance, if the category W is *Det*, and the grammar includes the phrase-structure rule *NP->Det N*, then we use this rule, which specifies Det N as representing a possible expansion of NP, and we determine P as being a noun phrase (NP).

Links

The left-corner parser can only partially deal with phrase-structure rules of type

$$A \rightarrow \phi$$

There are no problems when the parser encounters the null constituent while parsing top-down, namely in the situation

$$A \rightarrow B\ C$$

$$C \rightarrow \phi$$

because constituent C will be analized (parsed) top-down. It is only necessary to specify to the parser (by means of phrase-structure rules) that, when looking for a C, it can go on without analyzing anything.

- **This does not hold for languages like Romanian, where a phrase-structure rule of type**

$$NP \rightarrow Det\ N$$

can be accompanied by the rule

$$Det \rightarrow \phi$$

"*o fata citeste*", "*fata citeste*", namely when the null constituent is a null determinant occurring at the beginning of a noun phrase, a constituent that will be analyzed bottom-up.

The group of rules

$$S \rightarrow NP\ VP$$

$$NP \rightarrow ART\ N$$

$$ART \rightarrow \phi$$

tells the parser to accept the null determinant as representing the first step in the parsing of a NP or of an S. In this case, the left-corner parser will get into a loop.

- **<u>The problem</u>: the described left-corner parser is allowed (just like the shift-reduce one) to accept null constituents anytime it wishes to do so.**

- **Solution to the problem: placing constraints on the parser by adding a link table that specifies what *types* of constituents may occur on initial positions. The link table is also called a table of left corners.**

- **By adding the table of left corners**
  - **the issue of accepting null constituents is solved (the parser can't accept them anytime);**
  - **the parser becomes more efficient.**

- **Implementation:**

  Before starting its work, a left-corner parser preprocesses the context-free grammar to build a table where each row contains two cells, the first holding a non-terminal, and the second holding the collection of possible left corners of that non-terminal.

  The table

Category	Left corners (pre-terminals)
S	NP
VP	V

corresponds to the phrase-structure rules

$$S \rightarrow NP\ VP$$

$$VP \rightarrow V$$

$$VP \rightarrow V\ NP$$

Storing the Intermediate Results: Chart Parsing

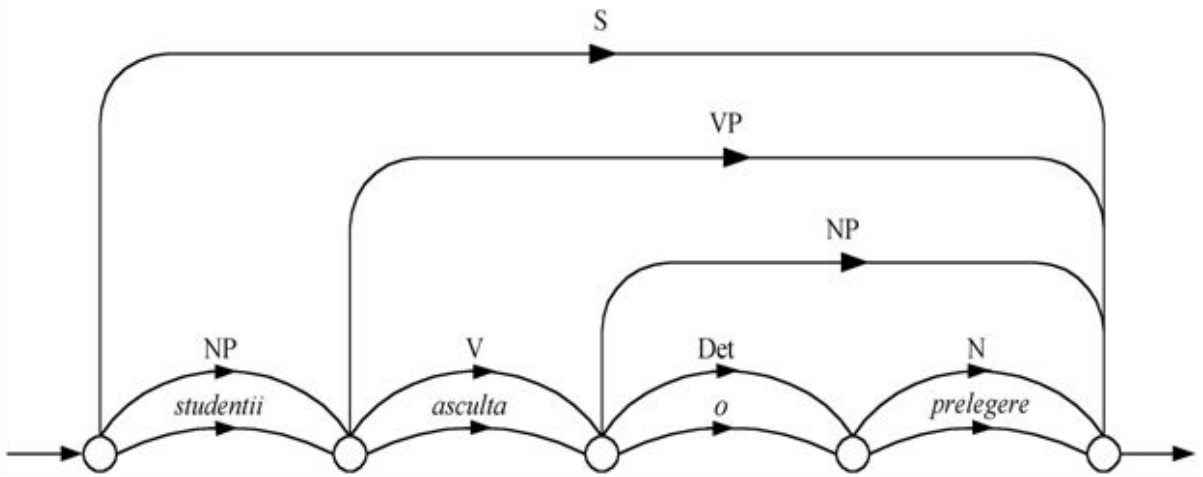
- **Storing intermediate results: a well-formed substring table (WFST) is a mechanism enabling a parser to keep a record of structures it has already found, so that it can avoid looking for them again.**

- **The extension represented by charts enables a parser, in addition, to record information about goals it has adopted. Recorded goals may have been unsuccessful or may still be under exploration. In either case it would be inefficient for the parser to start pursuing them again from scratch.**

- **With NLTK we apply the algorithm design technique of dynamic programming to the parsing problem. (Dynamming programming stores intermediate results and reuses them when appropriate, achieving significant efficiency gains). This approach to parsing is known as chart parsing.**

Well-formed Substring Tables (WFST)

- **Redundancy (repetition of the same situation) exists in the search space of a parsing algorithm. An efficient parser must have *memory*, namely it should be able to record the already found constituents and their structure.**



- **Consider the beginning and the end of the string to be numbered 0 and *n*, respectively, and the gaps between words to be numbered from 1 to *n-1* from left to right. A WFST tells us, for each pair of points *i*, *j* ($0 \leq i < j \leq n$), what categories can span the substring of words found between *i* and *j*.**
- **We think of a WFST as of a directed, acyclic graph with unique first and last nodes, a graph whose nodes are labelled from 0 (the first node) to *n* (the last), where *n* is the number of words in the string and whose arcs are labelled with syntactic categories and words.**

- **A WFST can be represented as a set of arcs, where an arc is a structure with the following attributes:**

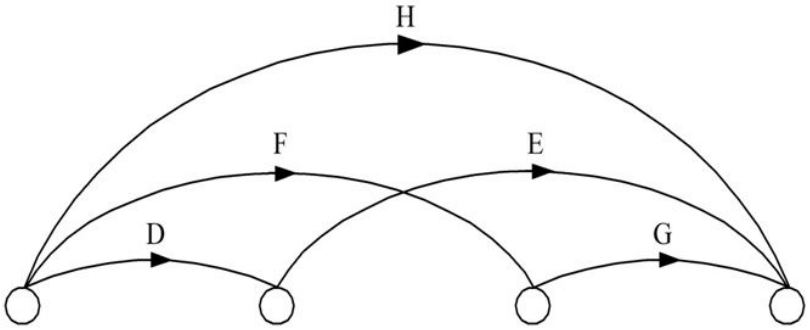
  <START> = ... an integer ...

  <FINAL> = ... an integer ...

  <LABEL> = ... a category ...

Remarks:

- **A phrase structure tree is also a directed acyclic graph, but one in which the nodes are labelled with categories and the arcs are unlabelled.**
- **A WFST encodes the order of phrases directly. (This is in contrast to trees).**



- **Trees encode immediate dominance relations, indicating which phrases are parts of which other phrases while WFSTs do not. Simply by looking at the WFST, you can not be sure of the rule (or rules) that legitimates a particular arc. Example: Is the above covering H warranted in virtue of H dominating D and E or in virtue of H dominating F and G?**

- **This is a neutral data structure: they permit us to use any of bottom up, left to right etc. parsing techniques, in any combination and any degree.**

- **The parser refrains from rediscovering things that it has already found out. The parser always tests for the presence of a certain category in the WSTF before trying to form it.**

- **Using a WFST:**

  A top-down parser, for instance, imposes the requirement that the already discovered syntactic groups are registered as entries in various locations of the WFST. When searching for a specific type of syntactic group, at a specific location in the input string, the table will be consulted. If the table contains an entry of the desired type, starting at the required position, then the table entries are used as possible analyses without the parser duplicating this work.

The Active Chart (Harta activa)

- **The use of a WFST will save work because it means that a successfully found syntactic group (for instance an NP) only needs to be found once. But it will not save the parser any time reinvestigating hypotheses that have previously failed.**

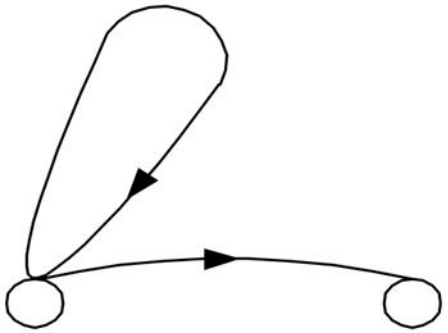
- **The WFST is a good way of representing facts about structure only. It does not help concerning structural hypotheses.**

- **The only way we can avoid duplicating previous attempts at parsing is to have an explicit representation about the different goals and hypotheses that the parser has at any one time.**

- **Charts will enable a parser, in addition, to record information about the goals it has adopted.**

We perform <u>two changes</u> to the data structure of a WFST:

- **Instead of requiring the directed graph to be strictly acyclic, we will relax things to permit <u>simple arcs</u> (called "*empty arcs*") that cycle back to the node they start out from. This will be the only type of cycle that will be permitted.**



- **The label on arcs changes from a simple category to a rule of the grammar. This rule will be called a "*dotted rule*", having the following significance:**

- **If $S \rightarrow NP\ VP$ is a rule of the grammar, then the following objects ("dotted rules") are well-formed arc labels:**

$$S \rightarrow .NP\ VP$$

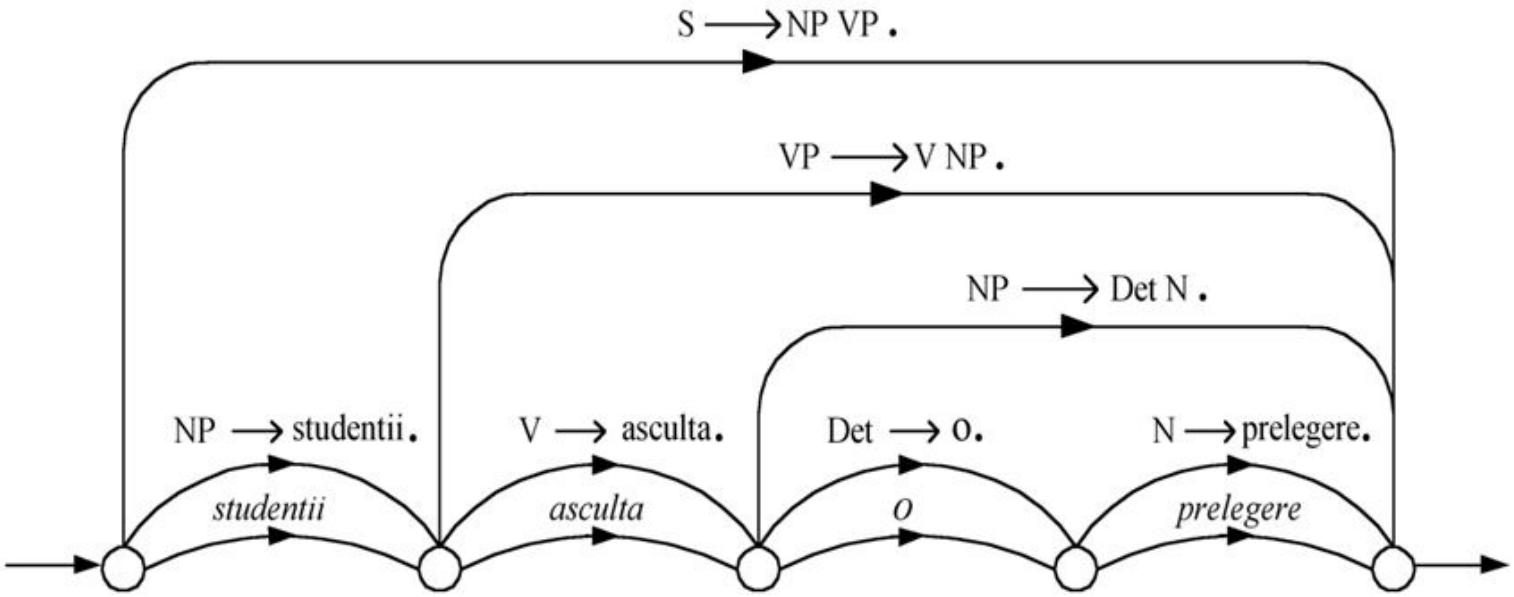
$$S \rightarrow NP.VP$$

$$S \rightarrow NP\ VP.$$

<u>Significance</u>:

The 'dot' in one of these labels indicates to what extent the hypothesis that this rule is applicable has been verified by the parser.

Example of a chart for the same input string:



- **A WFST that has been modified in this way is known as an active chart (*harta activa)* and will be referred to simply as *chart*.**

- **The family of parsers that exploit the chart as a data structure are known as *chart parsers*.**

- **A node in a chart is referred to as a *vertex* and the arcs as *edges*.**

- **Arcs that represent unconfirmed hypotheses are known as *active edges* and those that represent confirmed hypotheses are known as *inactive edges*. Namely: an inactive edge represents a result, while an active edge represents a structure hypothesis (a hypothesis concerning structure).**

- **The chart is also completely neutral with respect to the chosen parsing strategy.**

CHART REPRESENTATION

- **We can represent a chart as a set of structures, each of which has the following attributes**

  < START > = < an integer >

  < FINISH > = < an integer >

  < LABEL > = < a category >

  < FOUND > = < a sequence of categories >

  < TOFIND > = < a sequence of categories >

  where:

  - **<LABEL> is the left hand side (LHS) of the appropriate dotted rule;**
  - **<FOUND> is the sequence of right hand side (RHS) categories to the left of the dot;**
  - **<TOFIND> is the sequence of RHS categories to the right of the dot.**

  ✓ **In this representation, an edge whose value for TOFIND is the empty sequence will be an inactive edge, and all other edges will be active.**

- **The alternative representation**

  <0, 2, S → NP.VP>

  represents the following active and inactive edges:

  <START> = 0

  <FINISH> = 2

  <LABEL> = S

  <FOUND> = <NP>

  <TOFIND> = <VP>

The Fundamental Rule of Chart Parsing

Fundamental rule: If the chart contains edges $< i, j, A \rightarrow W1.B\ W2 >$ and $< j, k, B \rightarrow W3.>$, *where A and B are categories and W1, W2 and W3* are (possibly empty) sequences of categories or words, then add edge $< i, k, A \rightarrow W1\ B.W2 >$ to the chart.

Note that the essence of chart parsing is the interaction between an active edge and an inactive edge of the desired category.

The result is either a new inactive edge or a new active edge representing an extension of the original active edge.

Effect: Put a new edge into the chart that spans both the active and inactive edges.

Note that the rule performs only additions of edges to the chart. This rule does not remove from the chart the active rules that were successful. This is useful for finding all possible parses of the input sentence.

Initialization (of the chart)

- **The task of ensuring that there are some inactive edges to be found is the job of initialization.**

In order to start parsing, namely to apply the fundamental rule, the chart must contain at least one active edge and one inactive edge. By initializing the chart we provide active edges as follows:

Assuming the following lexicon is given

Word studentii:

<cat> = NP.

Word asculta:

<cat> = V.

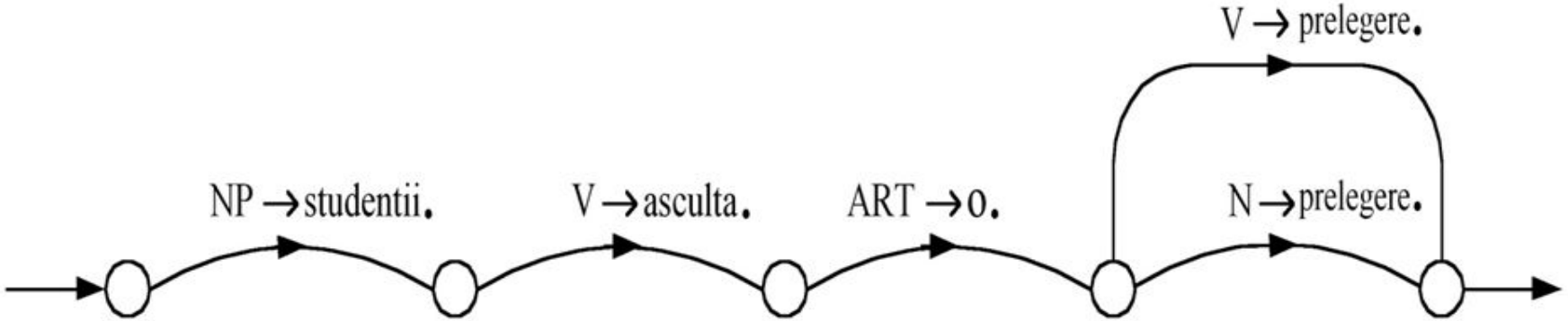
Word o:

<cat> = ART.

Word prelegere:

<cat> = N, V.

the corresponding initialized chart is the following:


V → prelegere.
NP → studentii.
V → asculta.
ART → o.
N → prelegere.

Rule Invocation

We still need a way of creating new active edges for the application of the fundamental rule!

There are two strategies for invoking rules: bottom-up and top-down

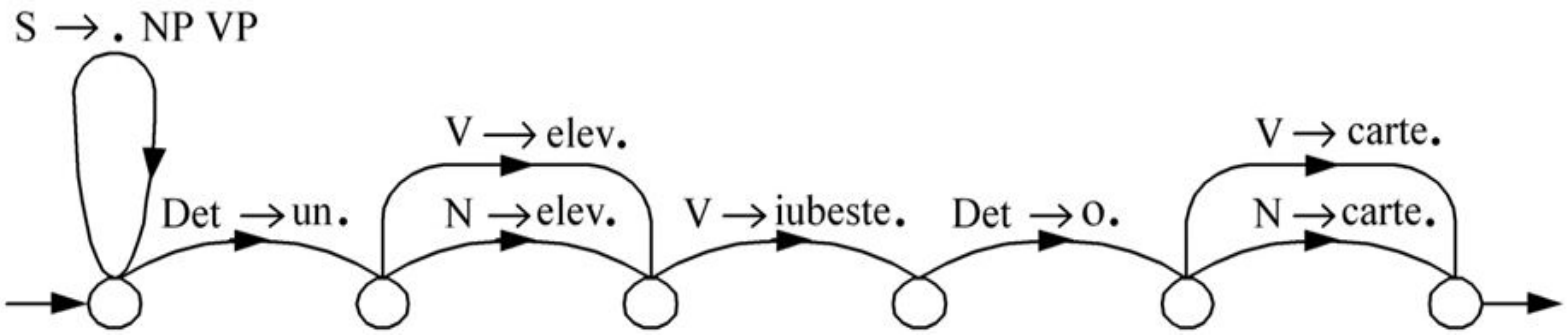
Bottom-up rule

If you are adding edge $< i, j, C \rightarrow W1. >$ to the chart, then for every rule in the grammar of the form $B \rightarrow C\ W2$, add an edge $< i, i, B \rightarrow .C\ W2 >$ to the chart.

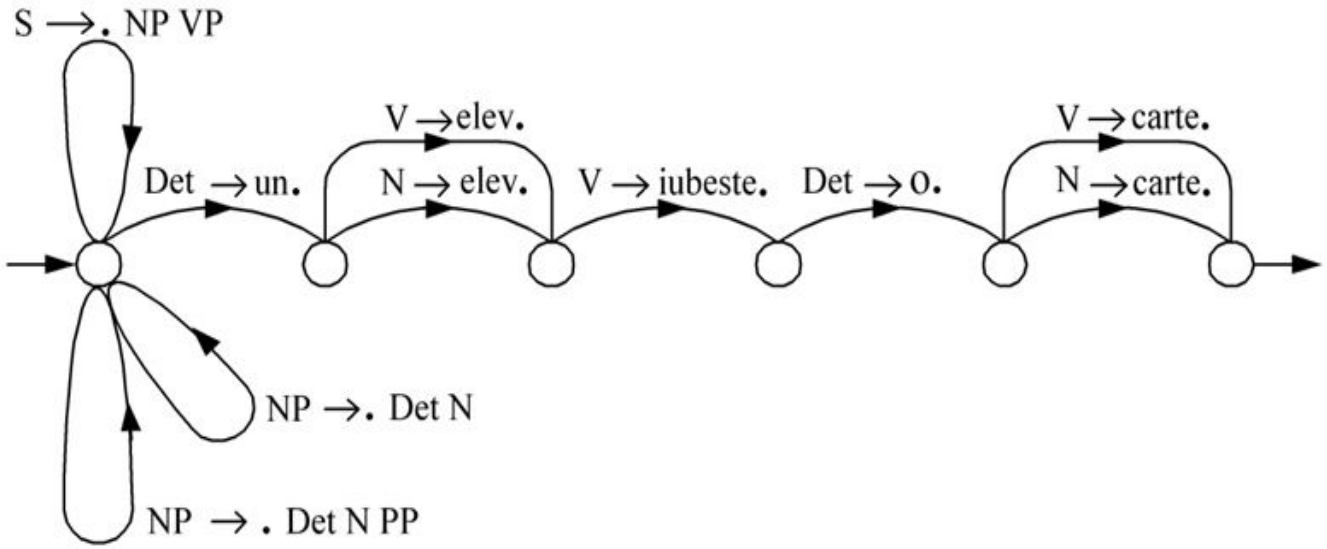
Top-down strategy

1) **At initialization, for each rule $A \rightarrow W$, where A is a category that can span a chart (typically S), add $< 0, 0, A \rightarrow .W >$ to the chart.**
2) **If you are adding edge $< i, j, C \rightarrow W1.BW2 >$ to the chart, then for each rule $B \rightarrow W$, add $< j, j, B \rightarrow .W >$ to the chart.**

Example for the top-down strategy; after the first clause:



After applying the second clause the chart becomes:



No matter what strategy we apply, we must take into account the fact that each active edge represents a hypothesis that must be explored. If this hypothesis is at least partially successful, then it can generate new hypotheses (both active and inactive edges), due to the action of the fundamental rule and the strategy for rule invocation that is applied.

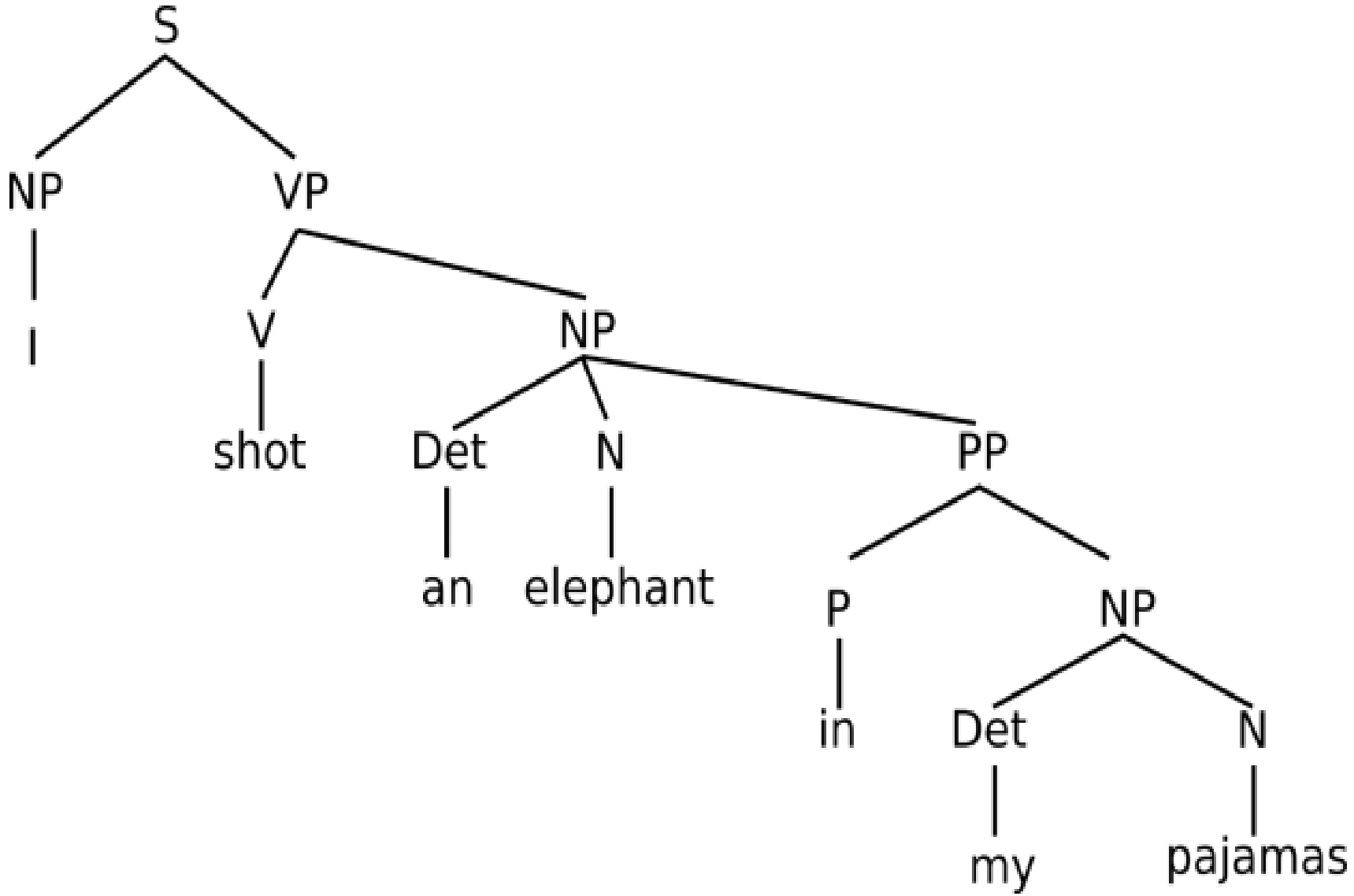
PARSING

- **The context-free grammars used for parsing so far find the constituent-based structure of the input sentence. This is useful for detecting the phenomenon of "omnipresent ambiguity".**

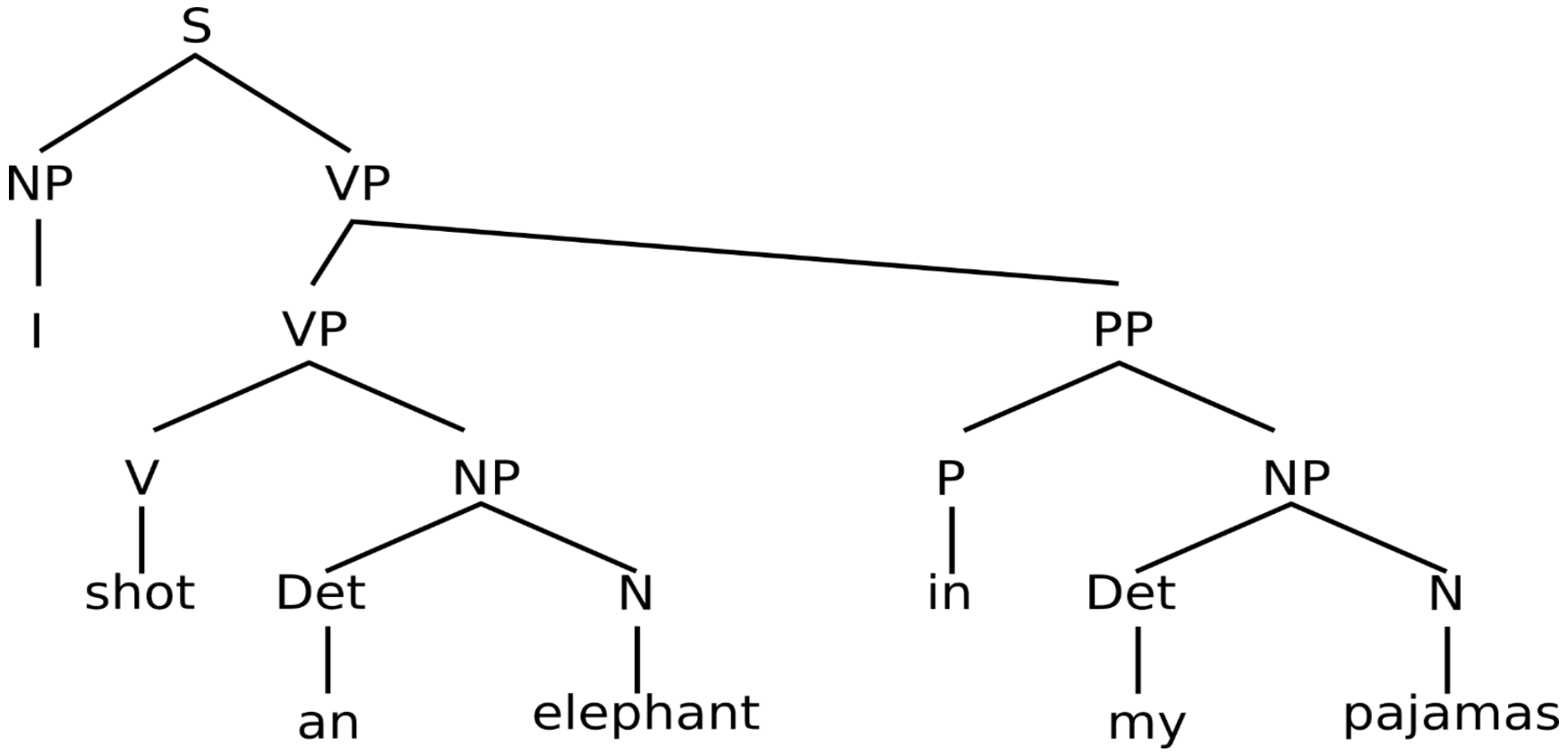
  Example: "*While hunting in Africa, I shot an elephant in my pajamas. How an elephant got into my pajamas I'll never know*".

- **The ambiguity is reflected by the two resulting parse trees. This is structural ambiguity that reflects the existing semantic ambiguity.**

Parse tree no. 1



Parse tree no. 2



Parse tree no.1 corresponds to the correct sense, which can only be disclosed within the context. Here the elephant is inside the pajamas.

Dependencies and Dependency Grammars

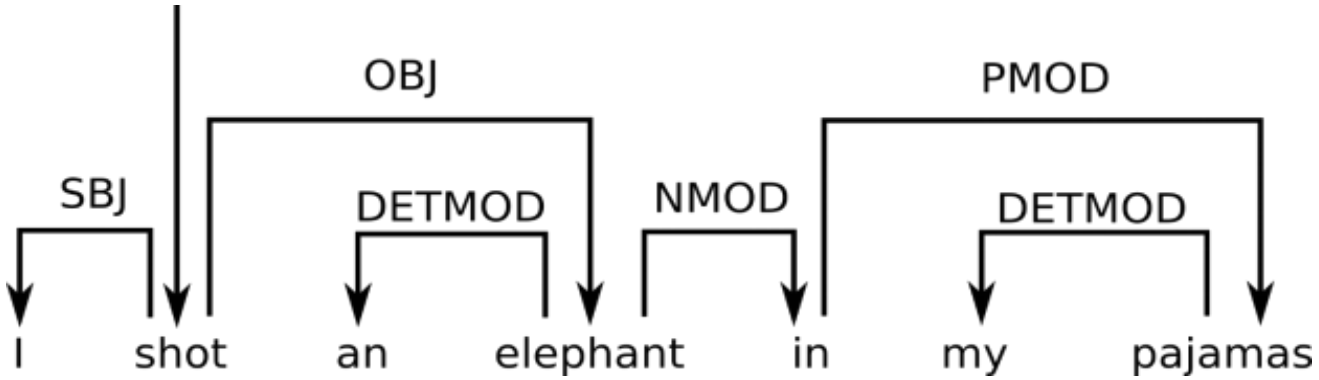
- **Context-free grammars (phrase-structure grammars) illustrate how words and sequences of words *combine* to form constituents.**

- **The dependency grammar approach focuses instead on how words *relate* to other words.**

- ***Dependency* is a binary asymmetric relation that holds between a head and its dependents.**

Example

Here is the dependency structure (parse) of the same sentence:



- **The head of a sentence is usually taken to be the tensed verb.**

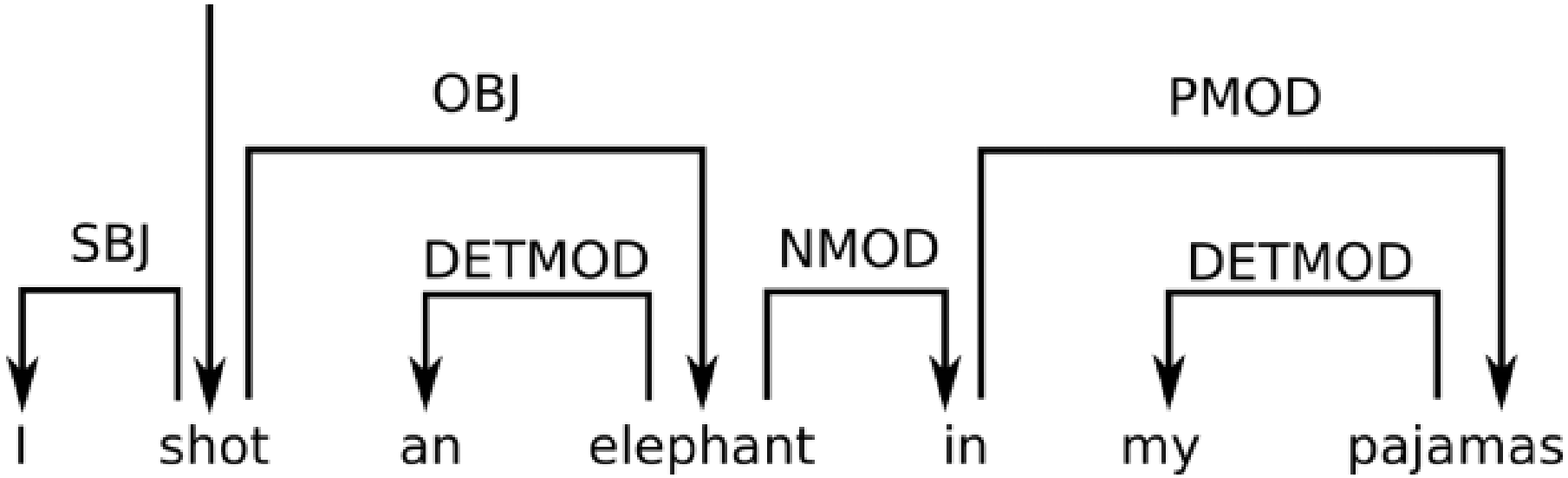
- **Every other word is either dependent on the sentence head or connects to it through a path of dependencies.**

- **Dependency grammar (DG) is a class of modern grammatical theories that are all based on the dependency relation (as opposed to the relation of phrase structure) and that can be traced back primarily to the work of Lucien Tesnière.**

- **Dependency is the notion that linguistic units, e.g. words, are connected to each other by directed links. The tensed verb is taken to be the structural center of clause structure. All other syntactic units (words) are either directly or indirectly connected to the verb in terms of the directed links, which are called *dependencies*.**

- **A dependency representation of a sentence is a labeled directed graph, where the nodes are the lexical items and the labeled arcs represent dependency relations from heads to dependents.**

Let's look again at the dependency structure of the same sentence (below):



Dependency structure:

- **The arcs are labeled with the *grammatical relations* between a dependent and its head. For instance:**
  - **the verb "shot" is the head of the entire sentence;**
  - **the pronoun "I" is in the grammatical relation SBJ (is the subject of) with the verb "shot";**
  - **"in" is in the grammatical relation NMOD (noun modifier) with "elephant".**

  These grammatical functions (given by relations) represent the *dependency type*. (Functiile gramaticale sunt exprimate prin tipurile dependentelor).

- **A dependency graph is projective (*projective analysis*) if, when all the words are written in linear order, the edges can be drawn above the words without crossing. This is equivalent to saying that a word and all its descendants (dependents and dependents of its dependents, etc.) form a contiguous sequence of words within the sentence.**

- **Dependency parsing is non-projective when the edges are allowed to cross.**

- **The dependency grammar theory has initially considered only the dependency syntactic analysis of projective type.**

Main ideas:

- **Dependency grammars are not based on the concept of constituent, but on the direct relations existing among words, viewed as dependency relations.**

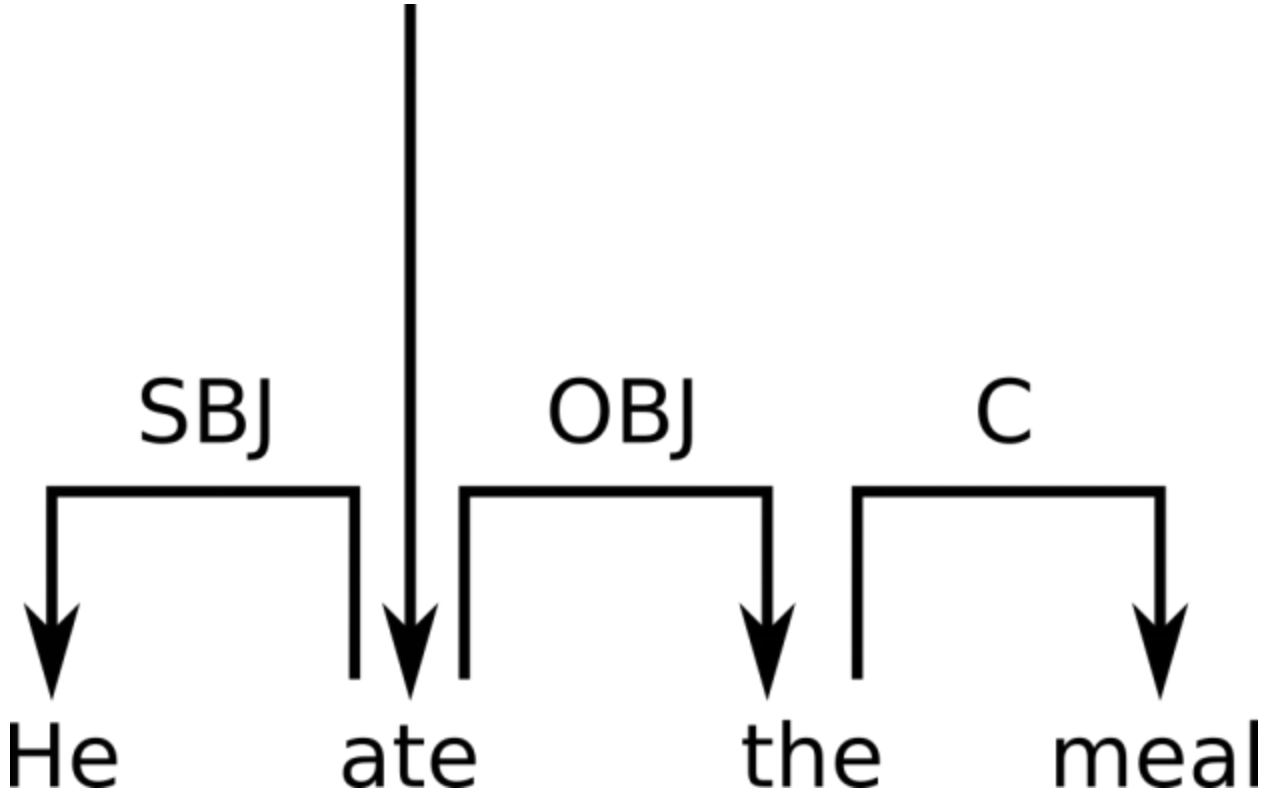
- **At the heart of dependency grammars we have the relation existing between the head word and the dependent word.**

- **In this framework, the syntactic analysis of a sentence (*dependency parse*) consists of describing *all* dependency relations existing among *all* words of a sentence.**

- **The dependency graph that represents the dependency parse of a sentence can be projective or non-projective. Initially, only the projective type of analysis was considered. However, in languages with more flexible word order than English, non-projective dependencies are more frequent.**

- **A major characteristic of this type of syntactic analysis is that it views syntactic phrases (noun phrase, verb phrase etc.) as representing byproducts of the existing dependencies! (byproduct = produs secundar)**

Comparison:



COMPARISON:

1. **Syntactic analysis based on constituents**

- **the syntactic phrase "the meal" represents a realization of the phrase-structure rule**

$$NP \rightarrow Det\ N$$

i.e. **it is a noun phrase organized around the noun "meal" – having the head role within the syntactic group**

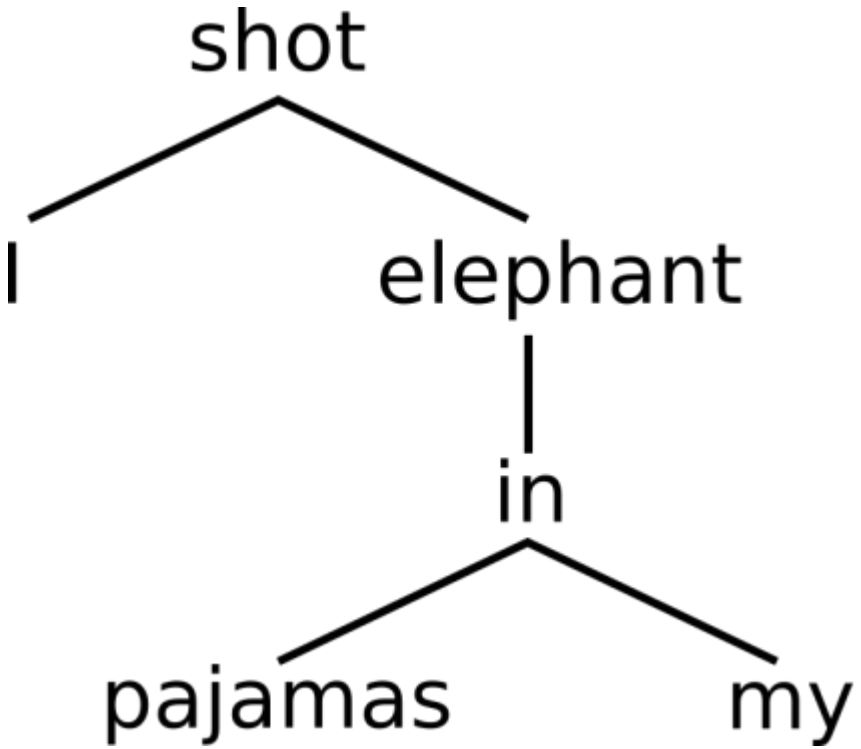
2. **Syntactic analysis of dependency type**

- **"the" represents the head and "meal" is the dependent (because "the" brings in, inserts "meal" into the sentence).**

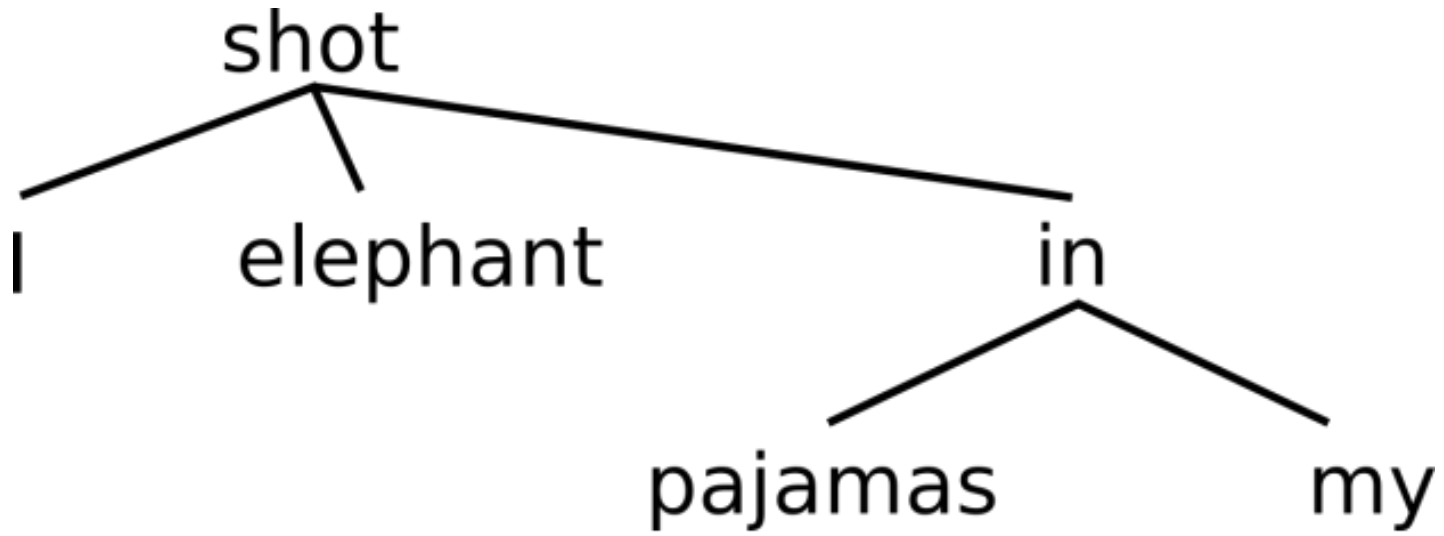
- **Note that: A great variety of dependency relations may exist among the words of a sentence. The role of dependency grammars is that of specifying the restrictions that the dependency relations must meet in order for the structure they define to be correct from the linguistic point of view.**

- **The dependency structure of a sentence can also be represented as a tree, one in which the dependents present themselves as sons of the corresponding head words.**

- **The ambiguity displayed by our previous example leads to two different dependency structures having the following corresponding parse trees:**



Here the elephant is inside the pajamas. Note that "in" depends on "elephant". The analysis is a <u>projective</u> one.



Here I was dressed in my pajamas when I shot the elephant. Note that "in" depends on "shot". The dependency type is different. The analysis is again a projective one.

CRITERIA for deciding what is the head H and what is the dependent D in a construction C (most important ones):

1. **H determines the distribution class of C; or alternatively, the external syntactic properties if C are due to H.**

2. **H determines the semantic type of C.**

3. **H is obligatory while D may be optional.**

4. **H selects D and determines whether it is obligatory or optional.**

5. **The morphological form of D is determined by H (e.g., agreement or case government).**

Main TOOL for dependency parsing:

STANFORD DEPENDENCY PARSER

Stanford dependencies are triplets consisting of:

- **name of the relation**
- **governor**
- **dependent**

See dependency parsing with NLTK at the lab !

REPRESENTING WORDS FOR NLP

There are currently two main ways to represent words in NLP:

- **Word embeddings -**
  - **a distributed representation**
- **Semantic networks**

✓ **Word embeddings: Word embeddings are a class of techniques where individual words are represented as real-valued vectors in a predefined vector space. Each word is mapped to one vector and the vector values are learned in a way that resembles a neural network (see Word2vec with gensim at the lab).**

✓ **Semantic networks: a knowledge base that represents semantic relations between concepts in a network. The semantic network represents an instrument for knowledge representation. It is a directed or undirected graph consisting of vertices, which represent concepts, and edges, which represent semantic relations between concepts (see the next lecture and lab for the semantic network WordNet).**

Word2vec

- **The word2vec algorithm uses a neural network model to learn word associations from a large corpus of text. Once trained, such a model can detect synonymous words or suggest additional words for a partial sentence.**

- **Word2vec is a group of related models that are used to produce word embeddings. These models are shallow (putin adanc), two-layer neural networks that are trained to reconstruct linguistic contexts of words. Word2vec takes as its input a large corpus of text and produces a vector space, typically of several hundred dimensions, with each unique word in the corpus being assigned a corresponding vector in the space. Word vectors are positioned in the vector space such that words that share common contexts in the corpus are located close to one another in the vector space. The vectors are chosen such that a simple mathematical function (the cosine similarity between the vectors) indicates the level of semantic similarity between the words represented by those vectors.**

- **In NLP, "word embedding" is a term used for the representation of words for text analysis, typically in the form of a real-valued vector that encodes the meaning of the word such that the words that are closer in the vector space are expected to be similar in meaning.**

- **Word2vec was created, patented, and published in 2013 by a team of researchers led by Tomas Mikolov at Google over these two papers:**

  - ✓ **Mikolov, Tomas; et al. (2013). "Efficient Estimation of Word Representations in Vector Space".**
  - ✓ **Mikolov, Tomas (2013). "Distributed representations of words and phrases and their compositionality". *Advances in Neural Information Processing Systems*.**

- **The general method ("A Neural Probabilistic Language Model", 2003):**

  "Associate with each word in the vocabulary a distributed word feature vector … The feature vector represents different aspects of the word: each word is associated with a point in a vector space. The number of features … is much smaller than the size of the vocabulary."

- **Word2vec can utilize either of two model architectures to produce a distributed representation of words: continuous bag-of-words (CBOW) or continuous skip-gram.**

- **In the continuous bag-of-words (CBOW) architecture, the model predicts the current word from a window of surrounding context words. The order of context words does not influence prediction (bag-of-words assumption)**

- **In the continuous skip-gram architecture, the model uses the current word to predict the surrounding window of context words. The skip-gram architecture weighs nearby context words more heavily than more distant context words.**

  - ✓ **Skip-gram: works well with a small amount of the training data, represents well even rare words or phrases.**
  - ✓ **CBOW: several times faster to train than the skip-gram, slightly better accuracy for the frequent words.**

  - ✓ **Note that results of word2vec training can be sensitive to parametrization (see the lab). The use of different corpus sizes and different model parameters (training epochs, window size and vocabulary size for the training algorithms) can greatly affect the quality of a word2vec model.**

- **The word embedding approach is able to capture multiple different degrees of similarity between words. Mikolov et al. (2013) found that semantic and syntactic patterns can be reproduced using vector arithmetic. Patterns such as "*Man is to Woman as Brother is to Sister*" can be generated through algebraic operations on the vector representations of these words such that the vector representation of *"Brother" - "Man" + "Woman"* produces a result which is closest to the vector representation of *"Sister"* in the model. Such relationships can be generated for a range of semantic relations (such as Country–Capital) as well as syntactic relations (e.g. present tense–past tense).**

Implementation – with Gensim

- **Gensim is a Python library for *topic modelling*, *document indexing* and *similarity retrieval* with large corpora.**

- **There are several ways to train word vectors in Gensim among which is Word2Vec.**

- **models.word2vec – Word2vec embedding**
  This module implements the word2vec family of algorithms. Tutorial:
  https://rare-technologies.com/word2vec-tutorial/

The Semantic Level of NLP – Introduction

- **The semantic level of natural language processing deals with the meaning of words and sentences, as well as with the way in which words' meanings *combine* in order to form the meaning of an entire sentence.**

- **At this processing level, computational semantics studies meaning in a context-free manner. Namely it studies the meaning of a sentence *regardless of the context in which that sentence occurs*. (Here the context is given by an entire paragraph, a page etc.)**

- **Note that, when taking the context into account, we move to the pragmatic level of processing natural language.**

- **Word Sense Disambiguation (WSD) signifies determining the meaning of a polysemous word occurring in a specific sentence.**

- **WSD has multiple applications in artificial intelligence and in information systems (man-machine communication, message understanding, machine translation, speech processing, text processing, information retrieval etc.)**

WSD

- **There are three classes of WSD methods and corresponding algorithms: supervised, unsupervised and knowledge-based.**
  - **Hybrid methods and approaches also exist (for instance, performing WSD at the border between unsupervised and knowledge-based techniques).**
- **Determining the meaning of a unique polysemous word in context (the sentence where it occurs) represents *local disambiguation*. We will be using it in order to determine the meaning of an entire text, which represents *global disambiguation*.**

Supervised WSD

- **Supervised WSD is based on machine learning.**

- **The (supervised) problem: given a word and a set of labels (etichete), namely a set of senses that can be associated with that word, one must establish which of these labels (senses) corresponds to that word, in the specific context in which it occurs.**

- **From a technical point of view, this is a classification problem.**

- **The elements which define this classification problem are: *the sense inventory* (inventarul de sensuri), namely the set of *all* senses that can be associated to *each* word that must be disambiguated and *the evidence sources* (sursele de evidenta), namely the information that can be extracted from the context and on which the classification process can rely.**

- **This WSD process uses a *training corpus* (corpus de antrenare) that contains the words which are to be disambiguated already labeled with their correct senses according to the respective context. This annotated corpus is used in order to train a classifier that can label the words occurring within a new and unlabeled text.**

- **The task is that of building a classifier that classifies correctly the new cases, based on the context in which they are used.**

- **Note that, because it requires the existence of an already annotated training corpus, this type of WSD can not be used in practice – at a large scale. Such a training corpus does not always exist and not for all words of a language – because it must be manually built by humans.**

Unsupervised WSD

- **Unsupervised WSD is also corpus-based, but uses an unannotated corpus, which makes it helpful in practice.**

- **Unsupervised WSD does not accomplish straight-forward sense disambiguation, because it does not assign sense tags to words. Rather it discriminates among the different meanings of a word.**

  - **Unsupervised WSD discriminates among word meanings based on information found in unannotated corpora.**

- **Note that unannotated corpora is always available for all languages (see, for instance, the texts on the web).**

  {singular – corpus
  plural – corpora}

- **Conclusion: These methods are data-driven and language-independent, and rely on the distributional characteristics of unannotated corpora.**

Unsupervised WSD

- **Distributional approaches do not assign meanings to words, but rather allow us to *discriminate* among the meanings of a word by identifying *clusters* of similar contexts, where each cluster shows that word being used in a particular meaning.**

- **Distributional approaches are based on the assumption that words that occur in similar contexts will have similar meanings.**

- **A classical clustering technique that can be used in order to obtain clusters of similar contexts is given by the Naïve Bayes model. (We will see this in a future lecture).**

- **Establishing the clustering technique to be used (classical versus state of the art) is of the essence.**

- **These methods are knowledge-lean (lipsite de cunostinte) because they do not rely on external knowledge sources (such as machine readable dictionaries, concept hierarchies, or sense-tagged text).**

Unsupervised WSD

We will focus on unsupervised corpus-based methods of *word sense discrimination* that are knowledge-lean, and do not rely on external knowledge sources such as machine readable dictionaries, concept hierarchies, or sense-tagged text.

Knowledge-based WSD

- **Knowledge-based WSD methods use a preexisting *sense inventory* in order to assign a specific meaning to a specific word in a given context.**

- **The sense inventory can be a classical dictionary in electronic format, a bilingual dictionary, namely machine readable dictionaries, concept hierarchies, semantic networks and lexical databases (such as WordNet) etc.**

- **The choice of sense inventory is of the essence for knowledge-based WSD accuracy. WordNet probably represents the most widely used sense inventory nowadays.**

WSD (in general)

- **WSD systems are based either on preexisting knowledge (thus requiring an external knowledge base) or on machine learning (thus requiring corpora – annotated when referring to supervised WSD, and unannotated when referring to unsupervised WSD).**

- **WSD can be performed at various levels of granularity. For instance, usage of the knowledge base WordNet leads to fine-grained WSD - because WordNet makes fine-grained distinctions between senses and subsenses. For example, most knowledge bases will determine the legal meaning of ("*suit*" - *speta*) without differentiating between *civil suit* (*speta civila*) and *criminal suit* (*speta penala*) as does WordNet.**

WordNet (WN)

- **WordNet (WN) is a lexical database of English created by Prof. George Miller at Princeton University in the nineties and being constantly updated.**

- **WN comprises all content words in English (nouns, adjectives, verbs and adverbs). Nouns and verbs are organized into hierarchies, adjectives and adverbs into clusters.**

- **The building brick of WN is the *synset* or synonym set (a set of synonyms).**

- **Synonymy in WN is not the classical one that linguists refer to. In WN, two words are considered synonyms if they refer to the same *concept*. For instance, the words "chair" and "table" could be considered synonyms because they both refer to the concept of *furniture*.**

WordNet (WN)

- **A polysemous word (a word with multiple senses) refers to a different concept through each of its senses. Therefore, a polysemous word belongs to multiple WN synsets. It belongs to a different sysnset through each of its senses. For example, the polysemous Romanian word "masa" could belong to each of the synsets referring to the concepts of *mancare, mobila, multime, masa din fizica etc.***
  - ✓ **Each WN synset refers to a *concept*.**

- **A WN synset contains all synonyms that lexicalize the concept it refers to, and that have the same part of speech (only nouns, only verbs etc.), together with a string of text named *gloss*, which resembles a classical dictionary definition. The gloss may contain an example of usage.**

- **Note that, in WN, the syntactic category (noun, adjective, verb, adverb) is used as parameter.**

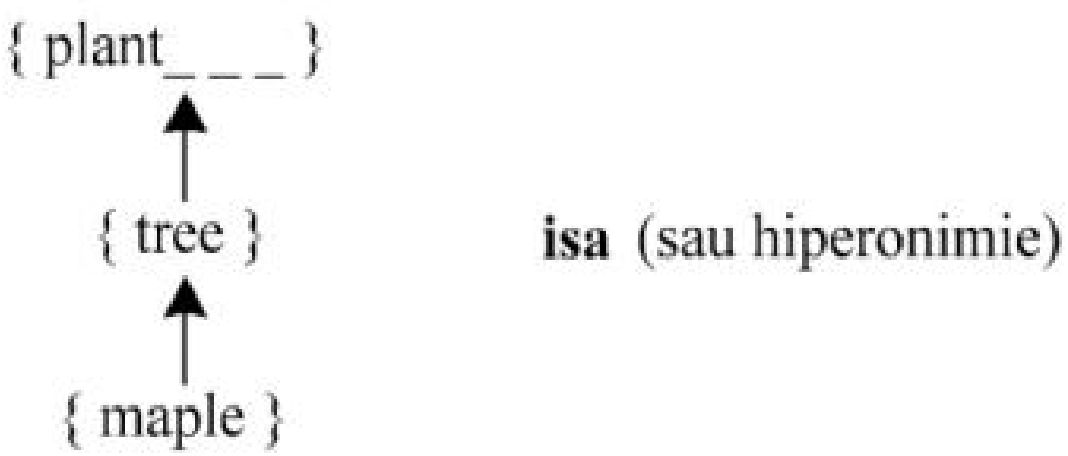
WordNet (WN)

- **WN synsets are linked together through semantic relations (existing between concepts and which are language independent).**

- **NOUN SYNSETS**
  The noun synsets are linked through semantic relations such as *hypernymy (hiperonimie)*, the inverse relation *hyponymy (hiponimie)*, *meronymy (meronimie)*, the inverse relation *holonymy (holonimie)* etc. Hypernymy denotes the father concept within the father-son relation. Its inverse, hyponymy denotes the son concept within the same semantic relation. Meronymy is the *part-of* relation, while holonymy is its inverse. For example, take *car* and *tire*, with the tire being *part of* the car (both are nouns). Then *car* is a holonym of *tire* and *tire* is a meronym of *car*.

- **The hypernymy relation organizes the WN noun synsets into hierarchies, as in the following example:**

(The hypernymy relation is the general *isa* relation from artificial intelligence in the case of the NLP field. The *isa* relation is used in AI to create hierarchies.)



Wordnet (WN)

Usage of semantic relations between synsets turns WN into a semantic network.

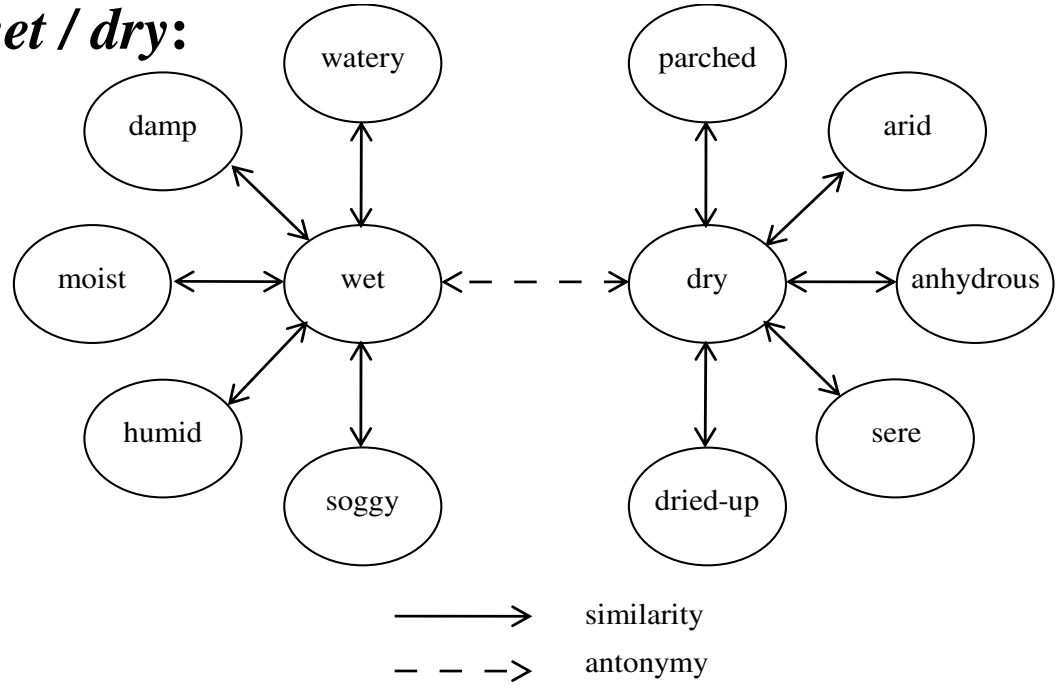
Through the hypernymy relation concepts inherit all properties of their superconcepts (parent concepts). In the above example, the maple has all the properties of a tree. Moreover, it has all the properties of a plant, in general. Going through the hierarchy, we will know that a maple has all the properties of a plant. The inheritance of properties turns WN into a knowledge base as well.

WN is a lexical database of English, a semantic network and a knowledge base. The fact that WN is a knowledge base makes it useful for a great variety of AI applications.

ADJECTIVE SYNSETS

- **Other, different semantic relations hold between adjective sysnsets. The main semantic relation is considered to be *antonymy*.**

- **Using *the antonymy relation* adjective synsets are organized into *clusters*. The following is an example of cluster formed around the antonyms *wet / dry*:**



This structure holds for the majority of descriptive adjectives (those that assign to a noun a value of an attribute).

Adjective synsets

- **The *similarity relation* holds between adjectives only. It brings in *indirect antonyms*. In the example, *wet* and *dry* are *direct antonyms*. *Wet* and *arid* are *indirect antonyms*. *Moist* does not have a direct antonym, but its indirect antonym can be found following the path *moist->wet->dry*.**

- **Other semantic relations important for adjectives are *also-see, pertaining-to* and *attribute*.**

- **The antonymy relation was proven to bring in important information for WSD.**

VERB SYNSETS AND ADVERB SYNSETS

- **Verb synsets are also organized into hierarchies, the same as noun synsets, only according to the *entailment* relation (to entail = a atrage dupa sine)**

- **Another important semantic relation for verbs is the *causal* relation.**

- **Adverb synsets are organized into clusters just like adjective synsets.**

SEE THE WN IMPLEMENTATION IN NLTK AT THE LAB!

Representing words in NLP

- **Remember?**

  There are two main ways to represent words in NLP:

  - **Word embeddings -**
    - **a distributed representation**
  - **Semantic networks**

  ✓ **Word embeddings: Word embeddings are in fact a class of techniques where individual words are represented as real-valued vectors in a predefined vector space. Each word is mapped to one vector and the vector values are learned in a way that resembles a neural network (see Word2vec with gensim at the lab).**

  ✓ **Semantic networks: a knowledge base that represents semantic relations between concepts in a network. This represents an instrument for knowledge representation. It is a directed or undirected graph consisting of vertices, which represent concepts, and edges, which represent semantic relations between concepts. The widely used semantic network for NLP is WordNet, which exists for various languages. (See also the lab for the semantic network WordNet).**

SIMILARITY AND RELATEDNESS
(Similaritate si inrudire)

- **Measures of semantic similarity or relatedness are used in various NLP applications such as: WSD, text summarization, information retrieval and information extraction etc.**

- **Semantic relatedness (inrudire) is a more general concept than semantic similarity (similaritate). Semantic relatedness has as inverse the semantic distance.**

- **Semantic similarity is a particular case of relatedness. Similarity is more specialized than relatedness. For instance, *doctors* and *hospitals* can be related, but are not similar. *Car* and *tire* are not similar, but are certainly related.**

- **Various measures for semantic similarity and/or relatedness exist in the literature.**

Relatedness and similarity

- **It has turned out that, for the task of WSD, relatedness is more important and relevant than similarity. (Example: *car* and *tire* are related but not similar. However, the occurrence of *car* in the same sentence as *tire* will help disambiguate *tire* with its noun sense – as part of a car – versus the verb sense of being tired).**

- **The knowledge-based WSD algorithm that we will study during the next lecture will strongly rely on relatedness between concepts.**

Knowledge-based WSD
The Extended Lesk Algorithm
(Banerjee & Pedersen, 2003)

- **Banerjee and Pedersen (2003) present a new measure of semantic relatedness between concepts that is based on the number of shared words (overlaps) in their definitions (glosses).**

- **This measure takes as input two concepts (represented by two WordNet synsets) and outputs a numeric value that quantifies their degree of semantic relatedness. This numeric value quantifying their degree of semantic relatedness is further used for performing WSD.**

- **The measure and corresponding WSD algorithm represent a variant of the classical Lesk algorithm (that is based on gloss overlaps), a variant which extends the glosses of the concepts under consideration.**

- **This measure extends the glosses of the concepts under consideration to include the glosses of other concepts to which they are related according to a given concept hierarchy (in this case WordNet).**

The Lesk Algorithm

- **Gloss overlaps were introduced by [Lesk, 1986] to perform WSD.**

- **The Lesk algorithm assigns a sense to a target word ("cuvant tinta" sau cuvant de dezambiguizat), in a given context, by comparing the glosses of its various senses with those of the other words in the context. That sense of the target word whose gloss has the most words in common with the glosses of the neighboring words is chosen as its most appropriate sense.**

- **Example: consider the following glosses of *car* and *tire***
  car*: *four wheel motor vehicle usually propelled by an internal combustion engine
  tire*: *hoop that covers a wheel, usually made of rubber and filled with compressed air
  - ✓ **the glosses of these concepts share the content word *wheel***

- **Note that the original Lesk algorithm only considers overlaps among the glosses of the target word and those words that surround it in the given context.**

- **Limitation: dictionary glosses are short and do not provide sufficient vocabulary (content words).**

- **The extended gloss overlap measure introduced by Banerjee and Pedersen (2003) expands the glosses of the words being compared to include glosses of concepts that are known to be related to the concepts being compared.**

  - ✓ **Concepts that are related through explicit WordNet relations are being taken into consideration.**

The Extended Gloss Overlap Measure

- **When measuring the relatedness between two input synsets, we not only look for overlaps between the glosses of those synsets, but also between the glosses of the hypernym, hyponym, meronym, holonym and troponym synsets of the input synsets, as well as between synsets related to the input synsets through such relations as attribute, similar-to, also-see etc.**

- **NOT all of these relations are equally helpful!**

- **The optimum choice of relations to use for comparisons is possibly dependent on the application in which the overlaps-measure is being employed.**

- **Here we will be applying this measure of relatedness to the task of WSD.**

  - ✓ **Example: In the case of noun disambiguation, empirical testing has proven it is useful to compare the glosses of hyponyms and meronyms of the input synsets. Note the recommendation to use hyponym synsets, not hypernym ones, or both of them. In other words, hyponymy provides more information than hypernymy, although both form the IS-A relation.**

Why it is important to have a measure of relatedness:

WordNet provides *explicit* semantic relations between synsets. However, such links do not cover all possible relations between synsets. For example, WN encodes no direct link between the synsets *car* and *tire*, although they are clearly related. We observe however that the glosses of these two synsets have words in common. Such overlaps provide evidence that there is an *implicit* relation between those synsets. Given such a relation, we further conclude that synsets explicitely related to *car* are thereby also related to synsets explicitly related to *tire*. Therefore:

- **This measure combines the advantages of gloss overlaps with the structure of a concept hierarchy (such as WN) to create an extended view of relatedness between synsets.**

Scoring mechanism

- **The original Lesk algorithm compares the glosses of a pair of concepts and computes a score by counting the number of words that are shared between them.**

- **Note that this scoring mechanism does not differentiate between single word and phrasal overlaps. For example, it assigns a score of 3 to the concepts *drawing paper* and *decal* having the following glosses:**

  drawing paper: paper that is specially preparedfor use in drafting

  decal: the art of transferring designs from specially prepared paper to a wood or glass or metal surface.

  Here there are three words that overlap, *paper* and the two-word phrase *specially prepared*.

- **Banerjee and Pedersen (2003) assign an *n* word overlap the score of n^2 (because a phrasal *n*-word overlap is a much rarer occurrence than a single word overlap). For the above gloss pair they would assign the score of 5 (instead of 3).**

The Banerjee and Pedersen overlap detection and scoring mechanism can be formally defined as follows:

- **When comparing two glosses, we define an overlap between them to be the longest sequence of one or more consecutive words that occurs in both glosses such that neither the first nor the last word is a function word, that is a pronoun, preposition, article or conjunction.**

- **If two or more such overlaps have the same longest length, then the overlap that occurs earliest in the first string being compared is reported.**

- **Given two strings, the longest overlap between them is detected, removed and in its place a unique marker is placed in each of the two input strings. The two strings thus obtained are then again checked for overlaps, and this process continues until there are no longer any overlaps between them.**

- **The sizes of the overlaps thus found are squared and added together to arrive at the score for the given pair of glosses.**

COMPUTING RELATEDNESS

- **The extended gloss overlap measure computes the relatedness between two input synsets *A* and *B* by comparing the glosses of synsets that are related to *A* and *B* through explicit relations provided in WordNet.**

1. We define *Rels* as a non-empty set of WN relations:

$$\text{RELS} \subset \{r \mid r \text{ is a relation defined in WordNet}\}.$$

For example, if $r \in$ RELS represents the hypernymy relation, then $r(A)$ returns the gloss of a hypernym synset of *A*.

- ✓ **Note that, if more than one synset is related to the input synset by means of the same semantic relation, then the respective glosses are concatenated and returned as a unique string. If no synset is related to the input one by the given relation, then the null string is returned.**

2. Next, form a non-empty set of *pairs* of relations from the set of relations RELS. The only constraint in forming such pairs is that if the pair (r_1, r_2) is chosen, ($r_1, r_2 \in$ RELS), then the pair (r_2, r_1) must also be chosen so that the relatedness measure is <u>reflexive</u>. That is,

$$relatedness\ (A, B) = relatedness\ (B, A).$$

Thus, we definethe set RELPAIRS as follows:

$$\text{RELPAIRS} = \{(R_1, R_2) \mid R_1, R_2 \in \text{RELS; if } (R_1, R_2) \in \text{RELPAIRS, then } (R_2, R_1) \in \text{RELPAIRS}\}$$

COMPUTING RELATEDNESS

3. **Assume that *score( )* is a function that accepts as input two glosses, finds the phrases that overlap between them and returns a score as previously described.**

- **Given all of the above, the relatedness score between the input synsets *A* and *B* is computed as follows:**
  $$relatedness\ (A, B) = \sum_{\forall (R_1, R_2) \in \text{RELPAIRS}} score\ (R_1(A), R_2(B)).$$

- **Example:**

Assume that our set of relations is RELS = {gloss, hype, hypo}
(where *hype* and *hypo* are contractions of hypernym and hyponym respectively). Further assume that our set of relation pairs is
RELPAIRS = {(gloss, gloss), (hype, hype), (hypo, hypo), (hype, gloss), (gloss, hype)}.
Then the relatedness between synsets *A* and *B* is computed as follows:
$$relatedness\ (A,B) = score\ (gloss(A), gloss(B)) + score\ (hype(A), hype(B)) + score\ (hypo(A), hypo(B)) + score(hype(A), gloss(B)) + + score\ (gloss(A), hype(B)).$$

Application to WSD

(Usage of the defined relatedness score in WSD)

Terminology and notations:

- **word to be disambiguated = target word**
- **given the input sentence, we look at the target word within a context window or window of context (fereastra de context)**
- **within the context window we look only at the content words (cuvinte cu continut: substantive, adjective, verbe, adverbe)**
- **A window of size *n* will denote taking into consideration *n* content words to the left and *n* content words to the right of the target word, whenever possible. The total number of words taken into consideration for disambiguation will therefore be *2n+1*. When not enough content words are available, the entire sentence in which the target word occurs will represent the context window.**
- **Assume that the context window consists of *2n+1* words denoted by w_i, $-n \le i \le +n$, where the target word is denoted w_0. Further let $|w_i|$ denote the number of candidate senses of word w_i, and let these senses be denoted by $s_{i,j}$, $1 \le j \le |w_i|$.**
- **Next we assign to each possible sense *k* of the target word a $SenseScore_k$ computed by adding together the relatedness scores obtained by comparing the sense of the target word in question with every sense of every non-target word in the context window. The SenseScore for sense $s_{0,k}$ is computed as follows :**

$$SenseScore_k = \sum_{i=-n}^{n} \sum_{j=1}^{|w_i|} relatedness(s_{0,k}, s_{i,j}), \quad i \neq 0$$

Result:

That sense with the highest SenseScore is judged to be the most appropriate sense for the target word.

<u>Complexity</u> – linear:

If there are on average α senses per word and the window of context is N words long, there are $\alpha^2 \times (N - 1)$ pairs of sets of synsets to be compared, which increases linearly with N.

Implementation remarks:

- **If several senses have the same SenseScore then the first sense – according to WN – is chosen for the target word.**

- **Increasing the size of the context window does not improve disambiguation results. (The usual context window sizes are 3, 5, 7).**

- **If a word in the window is used as part of a compound (ex.: *child-doctor*), then the senses associated with that compound are the candidates (i.e. not the senses of "child", not the senses of "doctor", but the senses of "child-doctor").**

- **The choice of semantic relations depends on the part of speech of the target word. For nouns the most informative relations are *hyponymy* and *meronymy*; for adjectives *also-see* and *attribute*; for verbs *hyponymy*.**

- **It is useful for disambiguation to use the example which is included in some of the WN glosses. (Use the gloss entirely, example included).**

Knowledge-rich WSD based on WordNet++

(Dezambiguizare BOGATA in cunostinte)

The idea:

- **Simple, basic disambiguation algorithms can have higher performance than sophisticated ones when being fed enough knowledge (cunostinte). Here knowledge will be provided by means of semantic relations.**

- **When provided with a vast amount of high-quality semantic relations, simple knowledge-lean disambiguation algorithms compete with state-of-the-art supervised WSD systems.**

(knowledge-lean = lipsit de cunostinte)

The method:

- **Paolo Ponzetto and Navigli (2010) have conceived a methodology for automatically extending WordNet with a great number of new semantic relations, provided by an encyclopedic resource – Wikipedia.**

- **The resulting resource is called WordNet++**

HOW:

Wikipedia pages are automatically associated with WordNet senses and the associative semantic relations from Wikipedia are transferred to WordNet. The result is a much richer lexical resource.

Wikipedia was launched on January 15, 2001 by Jimmy Wales and Larry Sanger.

The Structure of Wikipedia

Wikipedia text is very structured:

- **the pages corresponding to articles are formatted into sections and paragraphs**
- **various relations among pages exist**

The relations among pages include:

1. **Redirect pages (redirecting relations) – used in order to redirect the query towards the actual article page that contains information concerning the entity designated by the query. These pages are used in order to designate alternate expressions for an entity. (EX.: the pages CAR and SICKNESS redirect towards pages AUTOMOBILE and DISEASE respectively). These pages model synonymy.**

2. **Disambiguation pages – they collect links for a number of possible entities towards which the initial query might point. They model homonymy.**

3. **Internal links – articles mentioning other entries in the encyclopedia point towards them through the internal hyperlinks. They model article cross-reference.**

   EX.: The page about Wikipedia contains hyperlinks pointing towards the two inventors.

In summary:

A Wikipedia page (henceforth, Wikipage) presents the knowledge about a specific concept (e.g. SODA (SOFT DRINK)) or named entity (e.g. FOOD STANDARDS AGENCY). The page typically contains hypertext linked to other relevant Wikipages. For instance, SODA (SOFT DRINK) is linked to COLA, FLAVORED WATER, LEMONADE, and many others. The title of a Wikipage (e.g. SODA (SOFT DRINK)) is composed of the lemma of the concept defined (e.g. soda) plus an optional label in parentheses which specifies its meaning in case the lemma is ambiguous (e.g. SOFT DRINK vs. SODIUM CARBONATE). Finally, some Wikipages are redirections to other pages, e.g. SODA (SODIUM CARBONATE) redirects to SODIUM CARBONATE.

KNOWLEDGE-RICH WSD

- **Introduced in the paper:**

  Knowledge-rich Word Sense Disambiguation Rivaling Supervised Systems. Simone Paolo Ponzetto, Roberto Navigli, Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics, 2010, p. 1522 – 1531.

- **The underlying idea: the ideas of Bunescu and Pasca (2006) and of Mihalcea (2007), according to which the Wikipages can be considered as representing word senses.**

- **The technique that will be used: the knowledge from Wikipedia will be injected into a WSD system by means of a mapping to WordNet.**

Extending WordNet
(Mapping Wikipedia to WordNet)

PHASE I: automatic mapping between Wikipages and WordNet senses

PHASE II: the relations connecting Wikipedia pages are transferred to WordNet; the result is WordNet++

Notations:

- ✓ **The set of all Wikipedia pages** – $Senses_{Wiki}$
- ✓ **The set of all WN senses** – $Senses_{WN}$
- ✓ **A Wikipage** – w

Let

$\mu : Senses_{Wiki} \rightarrow Senses_{WN}$

be the mapping we want to obtain.

Let $w \in Senses_{Wiki}$

$$\mu(w) = \begin{cases} s \in Senses_{WN}(w), \text{ if a connection can be established} \\ \varepsilon, \text{ otherwise} \end{cases}$$

where $Senses_{WN}(w)$ **is the set of senses of *w*'s *lemma* in WordNet.**

- ✓ **<u>Reminder</u>: The title of a Wikipage (e.g. SODA (SOFT DRINK)) is composed of the lemma of the concept defined (e.g. soda) plus an optional label in parentheses which specifies its meaning in case the lemma is ambiguous (e.g. SOFT DRINK vs. SODIUM CARBONATE).**

<u>Example:</u>

$\mu(\text{SODA}(\text{SOFT DRINK})) = \text{soda}_n^2,$

where soda_n^2 **is the corresponding WordNet sense.**

Establishing the mapping

In order to establish a mapping between the two resources, we first perform two operations:

- **we identify various types of disambiguation contexts for Wikipages**
- **we identify various types of disambiguation contexts for WordNet senses**

These two contexts will be intersected in order to obtain the mapping.

1. Disambiguation context for a Wikipage

Given a target Wikipage *w*, that we want to map to a WordNet sense of *w*, we use the following information as disambiguation context:

- ✓ **Sense labels: e.g. given the page SODA (SOFT DRINK), the words soft and drink are added to the disambiguation context.**

- ✓ **Links: the titles' lemmas of the pages linked from the Wikipage *w* (outgoing links). For instance, the links in the Wikipage SODA (SOFT DRINK) include soda, lemonade, sugar etc.**

- ✓ **Categories: Wikipages are classified according to one or more categories, which represent meta-information used to categorize them. For instance, theWikipage SODA (SOFT DRINK) is categorized as SOFT DRINKS. Since many categories are very specific and do not appear in WordNet (e.g., SWEDISH WRITERS or SCIENTISTS WHO COMMITTED SUICIDE), we use the lemmas of their syntactic heads as disambiguation context (i.e. writer and scientist). To this end, we use the category heads provided by Ponzetto and Navigli (2009) in:**

Simone Paolo Ponzetto and Roberto Navigli. 2009. Large-scale taxonomy mapping for restructuring and integrating Wikipedia. In: Proceedings of IJCAI-09, pages 2083–2088.

The disambiguation context of a Wikipage is formed as follows:

Given a Wikipage *w*, we define its disambiguation context, *Ctx(w)*, as the set of words obtained from some or all of the three previously mentioned sources.

2. The disambiguation context of a WordNet sense

Given a WordNet sense *s* and its synset *S*, we use the following information as disambiguation context to provide evidence for a potential link in our mapping μ:

- **Synonymy: all synonyms of *s* in synset *S*.**
- **Hypernymy/Hyponymy: all synonyms in the synsets H such that H is either a hypernym (i.e., a generalization) or a hyponym(i.e., a specialization) of S.**
- **Sisterhood: words from the sisters of *S*. A sister synset *S'* is such that *S* and *S'* have a common direct hypernym.**
- **Gloss: the set of lemmas of the content words occurring within the gloss of *s*.**

Given a WordNet sense *s*, we define its disambiguation context, *Ctx(s)*, as the set of words obtained from some or all of these four sources.

The Mapping

Contexts *Ctx(w)* and *Ctx(s)* are intersected in order to obtain the mapping.

Mapping Algorithm

- **We want to link each Wikipage to a WordNet sense. Let w be such a Wikipage.**

The steps of the algorithm:

- **Initialization: our mapping μ is empty, i.e. it links each Wikipage w to ε (lines 1-2).**

- **For each Wikipage w whose lemma is monosemous both in Wikipedia and in WordNet i.e.**

$$|SensesWiki(w)| = |SensesWN(w)| = 1$$

we map w to its only WordNet sense w^1_n (lines 3-5).

- **For each remaining Wikipage w for which no mapping was previously found (i.e., $\mu(w)=\varepsilon$, line 7), we do the following:**

  - ✓ **lines 8-10: for each Wikipage d which is a redirection to w, for which a mapping was previously found (i.e. $\mu(d)\neq\varepsilon$, that is, d is monosemous in both Wikipedia and WordNet) and such that it maps to a sense $\mu(d)$ in a synset S that also contains a sense of w, we map w to the corresponding sense in S.**

  - ✓ **lines 11-14: if a Wikipage w has not been linked yet, we assign the most likely sense to w based on the maximization of the conditional probabilities $p(s|w)$ over the senses**

$$s \in Senses_{WN}(w)$$

(no mapping is established if a tie occurs, line 13).

- **As a result of the execution of the algorithm, the mapping μ is returned (line15).**

- **At the heart of the mapping algorithm lies the calculation of the conditional probability $p(s|w)$ of selecting the WordNet sense s given the Wikipage w. The sense s which maximizes this probability can be obtained as follows:**

$$\mu(w) = \underset{s \in Senses_{WN}(w)}{argmax}\ p(s/w) = \underset{s}{argmax}\ \frac{p(s,w)}{p(w)} =$$

$$= \underset{s}{argmax}\ p(s,w)$$

The latter formula is obtained by observing that $p(w)$ **does not influence our maximization, as it is a constant independent of** s**. As a result, the most appropriate sense** s **is determined by maximizing the joint probability** $p(s,w)$ **of sense** s **and page** w**.**

We estimate $p(s,w)$ as:

$$p(s,w) = \frac{scor(s,w)}{\sum_{\substack{s' \in Senses_{WN}(w) \\ w' \in Senses_{Wiki}(w)}} scor(s',w')}$$

where

$$scor(s,w) = \left| Ctx(s) \cap Ctx(w) \right| + 1$$

(we add 1 as a smoothing factor).

Formula interpretation:

We determine the best senses by computing the intersection of the disambiguation contexts of s and w, and normalizing by the scores summed over all senses of w in Wikipedia and WordNet.

The algorithm:

Input: $Senses_{Wiki}$, $Senses_{WN}$

Output: The mapping $\mu : Senses_{Wiki} \rightarrow Senses_{WN}$

1: **for each** $w \in Senses_{Wiki}$

2: $\mu(w) := \varepsilon$

3: **for each** $w \in Senses_{Wiki}$

4: **if** $|Senses_{Wiki}(w)| = |Senses_{WN}(w)| = 1$ **then**

5: $\mu(w) := w^{1}_{n}$

6: **for each** $w \in Senses_{Wiki}$

7: **if** $\mu(w) = \varepsilon$ **then**

8: **for each** $d \in Senses_{Wiki}$ **such that** d **redirects to** w

9: **if** $\mu(d) \neq \varepsilon$ **and** $\mu(d)$ **is in a synset of** w **then**

10: $\mu(w) :=$ **sense of** w **in synset of** $\mu(d)$**;break**

11: **for each** $w \in Senses_{Wiki}$

12: **if** $\mu(w) = \varepsilon$ **then**

13: **if no tie occurs then**

14: $\mu(w) := \underset{s \in Senses_{WN}(w)}{argmax}\ p(s/w)$

15: **return** μ

Transferring the semantic relations

- **The output of the presented algorithm is a mapping between Wikipages and WordNet senses (that is, implicitly, synsets). We use this alignment to enable the transfer of semantic relations from Wikipedia to WordNet.**

- **Given a Wikipage w we can collect all Wikipedia links occurring in that page. For any such link from w to w', if the two Wikipages are mapped to WordNet senses (i.e., $\mu(w) \neq \varepsilon$ and $\mu(w') \neq \varepsilon$), we can transfer the corresponding edge $(\mu(w), \mu(w'))$ to WordNet.**

- **Note that $\mu(w)$ and $\mu(w')$ are noun senses, as Wikipages describe nominal concepts or named entities.**

➢ **The obtained extended resource is called WordNet++**

The obtained resource, WordNet++, can be used, instead of WordNet, in various knowledge-based disambiguation algorithms (for instance the extended Lesk algorithm).

Conclusion:

"Knowledge-rich disambiguation" is a competitive alternative to supervised systems, even when relying on a simple algorithm.

The next step: BabelNet (for several languages)

BabelNet ≡ "the largest multilingual encyclopedic dictionary"

Supervised WSD

The Naïve Bayes Model for Supervised WSD

- **The classical approach to WSD that relies on an underlying Naïve Bayes model represents an important theoretical approach in statistical language processing: Bayesian classification (Gale et al. 1992).**

- **The idea:**

  - ✓ **The idea of the Bayes classifier (in the context of WSD) is that it looks at the words around an ambiguous word in a large context window. Each content word contributes potentially useful information about which sense of the ambiguous word is likely to be used with it. The classifier does no feature selection. Instead it combines the evidence from all features.**

- **NaïveBayes is widely used due to its efficiency and its ability to combine evidence from a large number of features. It is applicable if the state of the world that we base our classification on is described as a series of attributes. In our case, we describe the context of the ambiguous word in terms of the words that occur in the context.**

- **The Naïve Bayes assumption is that the attributes used for description are all conditionally independent, an assumption having two main consequences:**
  - ✓ **all the structure and linear ordering of words within the context are ignored, leading to a so-called "bag of words model";**
  - ✓ **the presence of one word in the bag is independent of another (which is clearly not true in the case of natural language).**

- **Note that:**
  - ✓ **In spite of these simplifying assumptions, this model has been proven to be quite effective when put into practice.**
  - ✓ **The contextual features (caracteristici contextuale) used for sense disambiguation here are the content words of the context window themselves.**
  - ✓ **The supervised training of the classifier assumes that we have a corpus where each use of ambiguous words is labeled with its correct sense.**

- **A *Bayes classifier* applies the *Bayes decision rule* when choosing a class, the rule that minimizes the probability of error. The Bayes decision rule is:**

  Decide s' **if** $P(s'/c) > P(s_k/c)$ **for** $s_k \neq s'$

- **Notations:**

  w **- an ambiguous word**

  $s_1, \ldots, s_k, \ldots, s_K$ **- the senses of the ambiguous word** w

  $c_1, \ldots, c_i, \ldots, c_I$ **- contexts of** w **in a corpus**

  $v_1, \ldots, v_j, \ldots, v_J$ **- words used as features for disambiguation**

- **With these notations, the Bayes decision rule is:**

  Decide s' **if** $P(s'/c) > P(s_k/c)$ **for** $s_k \neq s'$ **(1)**

- **The Bayes decision rule is optimal because it minimizes the probability of error. This is true because for each individual case it chooses the class (or sense) with the highest conditional probability and hence the smallest error rate. The error rate for a sequence of decisions (for example, disambiguating all instances of *w* in a multi-page text) will therefore also be as small as possible.**

- **In formula (1) we usually don't know the value of the probability $P(s_k/c)$, but we can compute it using <u>Bayes' rule:</u>**

$$P(s_k/c) = \frac{P(c/s_k)}{P(c)} \cdot P(s_k)$$

- **Here:**

  $P(s_k)$ is the *prior probability* of sense s_k, namely the probability that we have an instance of s_k if we do not know anything about the context. $P(s_k)$ is updated with the factor

$$\frac{P(c/s_k)}{P(c)}$$

  which incorporates the evidence (probele) which we have about the context. The result is the *posterior probability* $P(s_k/c)$.

- **If all we want to do is choose the correct class, we can simplify the classification task by eliminating $P(c)$ (which is a constant for all senses and hence does not influence what the maximum is). We can also use logs of probabilities to make the computation simpler. Then, we want to assign w to the sense s' where:**

$$s' = \arg\max_{s_k} P(s_k/c) = \arg\max_{s_k} \frac{P(c/s_k)}{P(c)} \cdot P(s_k) = \arg\max_{s_k} P(c/s_k) \cdot P(s_k) =$$
$$= \arg\max_{s_k} [\log P(c/s_k) + \log P(s_k)]$$

- **<u>Remark:</u>**

  This classifier of Gale et al. is an instance of a particular kind of Bayes classifier, the Naïve Bayes classifier. Naïve Bayes is widely used in machine learning due to its efficiency and its ability <u>to combine evidence from a large number of features</u>. It is applicable if the state of the world that we base our classification on is described as <u>a series of attributes</u>. In our case, we describe the context of w in terms of the words v_j that occur in the context.

- **The Naïve Bayes assumption is that the attributes used for description are all conditionally independent:**

$$P(c/s_k) = P\left(\{v_j / v_j \text{ in } c\}/s_k\right) = \prod_{v_j \text{ in } c} P(v_j/s_k) \qquad (3)$$

- **Remark: In our case, the Naive Bayes assumption has two consequences:**
  - ✓ **all the structure and linear ordering of words within the context is ignored; this is referred to as a bag of words model;**
  - ✓ **the presence of one word in the bag is independent of another (which is clearly not true...)**

- **With the Naive Bayes assumption, we get the following modified decision rule for classification:**

Modified decision rule for classification:

Decide s' if [(3) in (2)]

$$s' = \arg\max_{s_k} \left[\log P(s_k) + \sum_{v_j \text{ in } c} \log \mathrm{P}(v_j / s_k) \right] \quad \textbf{(4)}$$

In (4) we compute $P(v_j/s_k)$ and $P(s_k)$ as follows:

Modified decision rule for classification:

Decide s' if [(3) in (2)]

(4)

In (4) we compute $P(v_j/s_k)$ and $P(s_k)$ via Maximum-Likelihood estimation, perhaps with appropriate smoothing (netezire), from the labeled training corpus, and using the following notations:

$C(v_j, s_k)$ = the number of occurrences of v_j in a context of sense s_k in the training corpus;

$C(s_k)$ = the number of occurrences of sense s_k in the training corpus;

$C(w)$ = the total number of occurrences of the ambiguous word w.

- **Computing $P(v_j/s_k)$ and $P(s_k)$ via the Maximum-Likelihood estimation:**

$$P\left(v_j / s_k\right) = \frac{C\left(v_j, s_k\right)}{\sum_t C\left(v_t, s_k\right)} \quad \textbf{(5)}$$

$$P\left(s_k\right) = \frac{C\left(s_k\right)}{C\left(w\right)} \quad \textbf{(6)}$$

- **$P(v_j/s_k)$ and $P(s_k)$ are <u>the model parameters</u>.**
- **(5) and (6) are used in (4).**

After estimation, based on the training corpus, of parameters $P(v_j/s_k)$ and $P(s_k)$, the model has been trained and can be used as a classifier for WSD – by determining the most probable sense of an ambiguous word given the context in which it occurs (see the algorithm that follows).

The algorithm for supervised Bayesian disambiguation

1. **comment: training**
2. **for all senses** s_k **of** w **do:**
3. **for all words** v_j **in the vocabulary do**
4. $$P(v_j / s_k) = \frac{C(v_j, s_k)}{\sum_t C(v_t, s_k)}$$
5. **end**
6. **end**
7. **for all senses** s_k **of** w **do:**
8. $$P(s_k) = \frac{C(s_k)}{C(w)}$$
9. **end**
10. **comment: Disambiguation**
11. **for all senses** s_k **of** w **do:**
12. scor $(s_k) = \log P(s_k)$
13. **for all words** v_j **in the context window** c **do:**
14. scor (s_k) = scor $(s_k) + \log P(v_j/s_k)$
15. **end**
16. **end**
17. **choose** $s' = \arg\max_{s_k} scor(s_k)$

- **Conclusion: the Bayes classifier uses information coming from all words in the context window in order to make the disambiguation decision - at the cost of a somewhat unrealistic independence assumption.**

- **Test results: Gale, Church and Yarowski (1992) report this disambiguation algorithm working correctly on 90% of the occurrences of 6 ambiguous nouns (in the Hansard corpus). The tested nouns are: duty, drug, land, language, position and sentence.**

Supervised and unsupervised WSD based on a Naïve Bayes Model

- **In order to formalize the described model, we shall present the probability structure of the corpus *C* .**

- **Notations:**

  w **– target word (word to be disambiguated)**
  $s_1, \ldots, s_k$ **- possible senses for** w
  $c_1, \ldots, c_I$ **- contexts of** w **in a corpus** C
  $v_1, \ldots, v_J$ **- words used as contextual features for the disambiguation of** w

- **The probability structure of the corpus is based on one main assumption: *the contexts {ci, i} in the corpus C are independent*. Hence, the likelihood of *C* is given by the product**

$$P(C) = \prod_{i=1}^{I} P(c_i)$$

On considering the possible senses of each context, one gets

$$P(C) = \prod_{i=1}^{I} \sum_{k=1}^{K} P(s_k) \cdot P(c_i \mid s_k)$$

- **A model with independent features (usually known as the Naïve Bayes model) assumes that the contextual features are conditionally independent. That is,**

$$P(c_i \mid s_k) = \prod_{v_j \, in \, c_i} P(v_j \mid s_k) = \prod_{j=1}^{J} (P(v_j \mid s_k))^{|v_j \ in \ c_i|},$$

where by $|v_j \text{ in } c_i|$ **we denote the number of occurrences of feature** v_j **in context** c_i**. Then, the likelihood of the corpus** C **is**

$$P(C) = \prod_{i=1}^{I} \sum_{k=1}^{K} P(s_k) \prod_{j=1}^{J} (P(v_j \mid s_k))^{|v_j \ in \ c_i|}.$$

- **<u>The parameters</u> of the probability model with independent features are**

$$\{P(s_k),\ k = 1,..., K \text{ and } P(v_j \mid s_k),\ j = 1,..., J,\ k = 1,..., K\}.$$

- **<u>Notation:</u>**

$P(s_k) = \alpha_k,\ k = 1,..., K,\ \alpha_k \geq 0$ **for all** k, $\sum_{k=1}^{K} \alpha_k = 1$

$P(v_j \mid s_k) = \theta_{kj},\ k = 1,..., K,\ j = 1,..., J,\ \theta_{kj} \geq 0$ **for all** k **and** j**,**

$\sum_{j=1}^{J} \theta_{kj} = 1$ **for all** $k = 1, ..., K$

With this notation, the likelihood of the corpus C can be written as

$$P(C) = \prod_{i=1}^{I} \sum_{k=1}^{K} \alpha_k \prod_{j=1}^{J} (\theta_{kj})^{|v_j \text{ in } c_i|}.$$

The well known Bayes classifier involves the a posteriori probabilities of the senses, calculated by the Bayes formula for a specified context c,

$$P(s_k \mid c) = \frac{P(s_k) \cdot P(c \mid s_k)}{\sum_{k=1}^{K} P(s_k) \cdot P(c \mid s_k)} = \frac{P(s_k) \cdot P(c \mid s_k)}{P(c)}$$

with the denominator independent of senses. The Bayes classifier chooses the sense s' for which the a posteriori probability is maximal (sometimes called the Maximum A Posteriori classifier)

$$s' = \arg\max_{k=1,\ldots,K} P(s_k \mid c).$$

Taking into account the previous Bayes formula, one can define the Bayes classifier by the equivalent formula

$$s' = \arg\max_{k=1,\ldots,K} (\log P(s_k) + \log P(c \mid s_k)).$$

Of course, when implementing a Bayes classifier, one has to estimate the parameters first.

Parameter estimation

Parameter estimation is performed by the Maximum Likelihood method, for the available corpus *C*. That is, one has to solve the optimization problem

$$\max\left(\log P(C) \,|\, \{P(s_k),\ k=1,...,K \ \text{ and } \ P(v_j \,|\, s_k),\ \ j=1,...,J,\ k=1,...,K\}\right).$$

For the Naïve Bayes model, the problem can be written as

$$\max\left(\sum_{i=1}^{I}\log\left(\sum_{k=1}^{K}\alpha_k\prod_{j=1}^{J}(\theta_{kj})^{|v_j \text{ in } c_i|}\right)\right) \qquad (1)$$

with the constraints

$$\sum_{k=1}^{K}\alpha_k = 1,\ \sum_{j=1}^{J}\theta_{kj} = 1 \text{ for all } k = 1, ..., K$$

For *supervised disambiguation*, where an annotated training corpus is available, the parameters are simply estimated by the corresponding frequencies (as in last week's lecture):

$$\hat{\theta}_{kj} = \frac{|\text{occurrences of } v_j \text{ in a context of sense } s_k|}{\sum_{j=1}^{J}|\text{occurrences of } v_j \text{ in a context of sense } s_k|},\ k = 1, ..., K;\ j = 1, ..., J$$

$$\hat{\alpha}_k = \frac{|\text{occurrences of sense } s_k \text{ in } C|}{|\text{occurrences of } w \text{ in } C|},\ k = 1, ..., K$$

For *unsupervised disambiguation*, where no annotated training corpus is available, the maximum likelihood estimates of the parameters are constructed by means of the Expectation-Maximization (EM) algorithm.

- **For the unsupervised case, the optimization problem (1) can be solved only by iterative methods. The Expectation Maximization algorithm is a very successful iterative method, known as very well fitted for models with missing data. (Here the missing data are the senses of the ambiguous word).**

- **Each iteration of the algorithm involves two steps:**
  - ✓ **estimation of the missing data by the conditional expectation method (E-step)**
  - ✓ **estimation of the parameters by maximization of the likelihood function for complete data (M-step)**

  The E-step calculates the conditional expectations given the current parameter values, and the M-step produces new, more precise parameter values. The two steps alternate until the parameter estimates in iteration $r + 1$ and r differ by less than a threshold ε.

- **The EM algorithm is guaranteed to increase the likelihood log $P(C)$ in each step. Therefore, two stopping criteria for the algorithm could be considered:**
  - **Stop when the likelihood log $P(C)$ is no longer increasing significantly;**
  - **Stop when parameter estimates in two consecutive iterations no longer differ significantly.**

- **The available data, called *incomplete data*, are given by the corpus C. The *missing data* are the senses of the ambiguous words, hence they must be modeled by some random variables. The *complete data* consist of incomplete and missing data.**

Unsupervised WSD with the Naïve Bayes model – Feature selection

- **The classical clustering technique is represented by the Naïve Bayes model trained with the EM algorithm.**
- **When the Naïve Bayes model is applied to supervised disambiguation, the actual words occurring in the context window are usually used as features. This type of framework generates a great number of features and, implicitly, a great number of parameters. This can dramatically decrease the model performance since the available data is usually insufficient for the estimation of the great number of resulting parameters. This situation becomes even more drastic in the case of unsupervised disambiguation, where parameters must be estimated in the presence of missing data (the sense labels).**
- **In order to overcome this problem, the various existing unsupervised approaches to WSD implicitly or explicitly perform a feature selection.**

Semantic WN-based feature selection

- **This approach is based on a set of features formed by the actual words occurring near the target word (within the context window) and reduces the size of this feature set by performing knowledge-based feature selection that relies entirely on WN.**

- **The WN semantic network provides the words considered as relevant for the set of senses taken into consideration corresponding to the target.**

- **The value of a feature is given by the number of occurrences of the corresponding word in the given context window.**

Semantic WN - based Feature Selection

- **We start by choosing words occurring in the same WN synsets as the target word (WN synonyms), corresponding to all senses of the target. Additionally, the words occurring in the synsets related (through explicit relations provided in WN) to those containing the target word are also considered as part of the so-called "disambiguation vocabulary". Synsets and relations are restricted to those associated with the part of speech of the target. The content words of the glosses and the example strings of all types of synsets participating in the disambiguation process are equally considered. Usage of the example string as well is in accordance with previous studies performed for knowledge-based WSD.**

Syntactic Dependency-based Feature Selection

- **This type of features augments the role of linguistic knowledge.**

- **Syntactic features are provided by dependency relations as defined by the classical Dependency Grammar formalism.**

- **The semantic space that is proposed to the Naïve Bayes model for unsupervised WSD is entirely based on syntactic knowledge, more precisely on dependency relations, extracted from natural language texts via a syntactic parser. The parser extracts dependency relations that will indicate the disambiguation vocabulary required by Naïve Bayes. This is how the disamb. voc. is formed:**

- **It is various combinations of dependencies, pointed out by the obtained dependency structure, that form the context over which the semantic space for WSD is constructed. The number of features (coming from all resulted dependencies) is decreased by taking into account only specific dependency relations. The disambiguation vocabulary is formed by considering all words that participate in the considered dependencies.**

Syntactic Dependency-based Feature Selection

In order to inform the construction of the semantic space for WSD with syntactic knowledge of this type, several elements must be taken into consideration. At the first stage of the experiments, we make no qualitative distinction between the different relations, by not taking into account the type of the involved dependencies. At the second stage of the experiment, we can inform the construction of the semantic space in a more linguistically rich manner, by considering the dependency type. Therefore we make use only of specific dependency relations, which are considered more informative than others relatively to the studied part of speech.

Syntactic Dependency-based Feature Selection

- **In defining the syntactic context of the target word, we can first take into consideration direct relationships (first order dependencies) between the target and other words, where the target can be either the head or a dependent, and which correspond to paths of length 1 anchored at the target in the associated dependency graph. At the next stage we can consider indirect relationships between the target and other words, by taking into account paths of length 2 (second order dependencies) in the same associated graph. The order of these dependencies (lengths of the paths in the associated graph) represents a parameter that can vary and which can be considered analogous to the classical "window size" parameter. Just as in the case of the window size, this parameter should have relatively small values, since it is a known fact that linguistically interesting paths are of limited length. This justifies the usual choice of limiting the investigation to second order dependencies.**

- **Separate experiments, which view the target word as head and as dependent, respectively, can be carried out both corresponding to first order and to second order dependencies.**

Syntactic Dependency-based Feature Selection

- **When using the Naïve Bayes model as clustering technique, it is observed that a small number of dependency types should be considered, if possible just one, in order to decrease the number of parameters that must be estimated by the EM algorithm.**

  This conclusion would not hold in the case of a clustering technique requiring a high number of features (such as spectral clustering, for instance). In such a case, several types of dependencies would probably have to be considered, which would create a more linguistically informed semantic space for WSD – that could be even more beneficial.

Syntactic Dependency-based Feature Selection

While the classical dependency-based linguistic theory does not allow the arches denoting the dependency relations to intersect (thus leading to an oriented graph which has no cycles), the dependency analysis performed by the Stanford parser, for instance, can be either projective (disallowing crossing dependencies) or non-projective (permitting crossing dependencies). The number of resulting features is then decreased by taking into account only dependency relations of specific types. Both types of analyses (non-projective and projective) have been performed and discussed in the literature, relatively to the presented case studies, with the projective syntactical analysis leading to the best results so far.

Algoritmul coloniei de furnici pentru dezambiguizarea nesupervizata a textelor (*Ant colony algorithm*)

Bibliografie

- **Ant Colony Algorithm for the Unsupervised Word Sense Disambiguation of Texts: Comparison and Evaluation** (D. Schwab, J. Goulian, A. Tchechmedjiev, H. Blanchon) in "Proceedings of COLING 2012", Mumbai, paginile 2389-2404

- Lucrarea se refera la *knowledge-based unsupervised word sense disambiguation* (este la granita dintre cele doua)

- Procesul de invatare este nesupervizat dar algoritmului i se dau cunostinte (foloseste Lesk extins, deci cunostinte date de WordNet)
- Masura Lesk locala este propagata intr-un intreg text. In general un algoritm global este o metoda care permite propagarea unei masuri locale in cadrul unui intreg text pentru a atribui un sens fiecarui cuvant al textului.

Analiza se face la nivelul intregului text; fereastra de context se inlocuieste cu tot textul (algoritm global)

Notatii:

- w_i = cuvant ambiguu
- m = numar de cuvinte in intregul text
- $w_{i,j}$= cel mai adecvat sens al cuvantului w_i fiind dat contextul.
- $d(w_{i,j})$= definitia unui sens j al cuvantului i

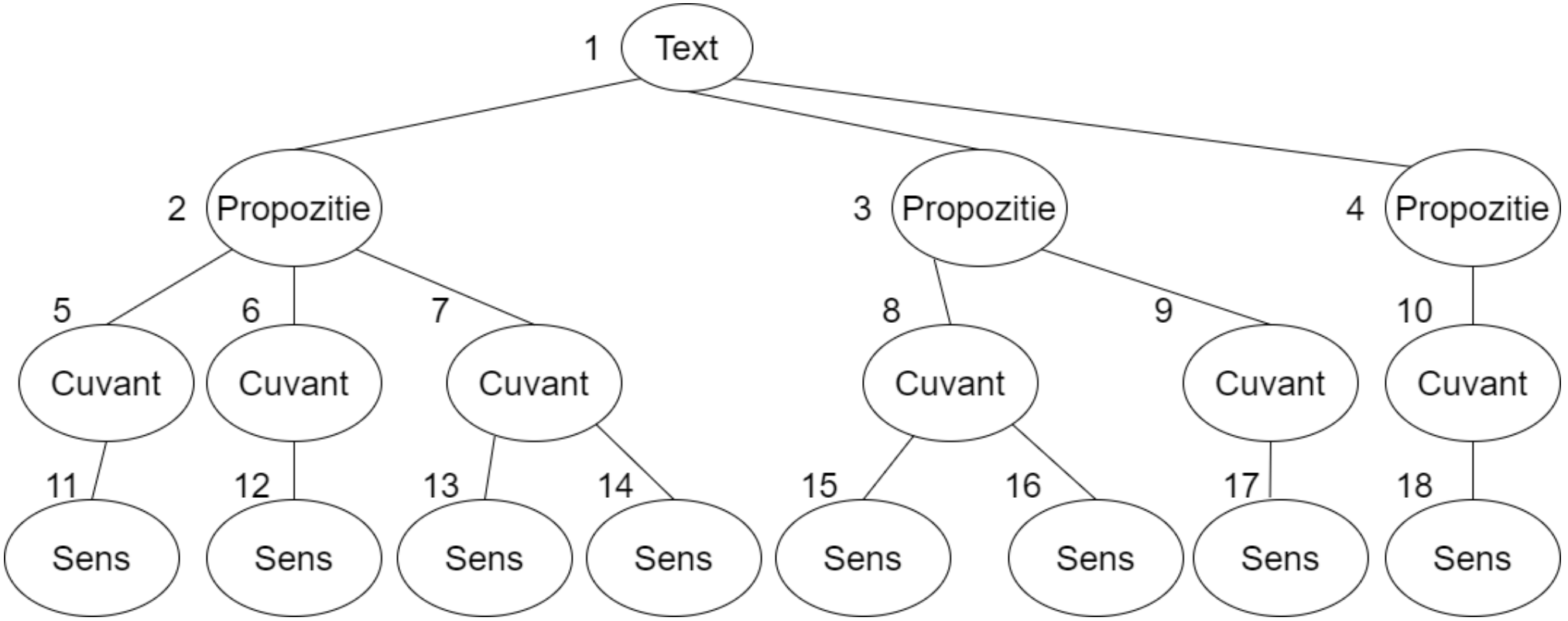
- Spatiul de cautare corespunde tuturor combinatiilor posibile ale tuturor sensurilor cuvintelor textului care se proceseaza.
- Cu C se noteaza o configuratie a problemei.
- Reprezentarea unei configuratii C a problemei consta intr-un vector de intregi astfel incat j = C[i] este sensul j selectat pentru cuvantul w_i

Furnici artificiale

- Au fost prima data folosite pentru a rezolva “Traveling Salesman Problem” (Dorigo si Gambardella, 1997)
- Mediul este, de obicei, reprezentat printr-un graf, in interiorul caruia furnici virtuale exploreaza dare de feromon (drumuri) depozitat de acele furnici
- Avantaje:
  - O buna alternativa pentru rezolvarea problemelor modelate prin grafuri
  - Adaptabilitate mare la schimbarile de mediu (medii cu grad mare de schimbare)

- Fiecare sens posibil al unui cuvant este asociat unui cuib (nod frunza)
- Cuiburile produc furnici care se misca in graf
- Furnicile cara un miros (matrice) care contine cuvintele din definitia sensului corespunzator cuibului mama al furnicii
- Tipuri de noduri pentru o furnica:
  - Cuibul mama (unde s-a nascut furnica)
  - Cuib dusman (corespunde unui alt sens al aceluiasi cuvant)
  - Potential cuib prieten (reprezentat de orice alt cuib)
  - Un nod simplu (orice nod care nu este cuib)
- Fiecarui nod simplu, ii este asociat un vector de miros, de lungime fixata, care initial este vid.

- Exemplu de graf pentru reprezentare:



Radacina arborelui corespunzatoare grafului e intreg textul.
De exemplu, cuvantul corespunzator nodului 7 este un cuvant polisemantic.

- Miscarea furnicilor depinde de:
  - Scorurile acordate de algoritmul local
  - Prezenta energiei
  - Trecerea altor furnici (atunci cand trec, furnicile lasa o dara de feromon, care se evapora cu timpul)
  - Vectorii de miros (*odour*) ale nodurilor (furnicile depoziteaza o parte din mirosul lor in nodurile prin care trec)
- Atunci cand o furnica ajunge in cuibul unui alt termen (i.e. un nod care corespunde unui sens) poate decide: sa continue explorarea sau sa construiasca un pod intre cele doua cuiburi si sa urmeze acest pod pentru a ajunge acasa.

- Podurile se comporta ca niste muchii obisnuite, numai ca, atunci cand concentratia de feromon corespunzatoare lor atinge valoarea 0, podul se prabuseste.
- Cu cat mai apropiate sunt sensurile corespunzatoare cuiburilor, cu atat mai mult un pod intre ele va contribui la intarirea lor reciproca si la partajarea de resurse intre ele
- Podurile dintre sensuri mai indepartate vor avea tendinta sa se prabuseasca
- In acest fel se creeaza drumuri de interpretare ("*interpretative paths*"), adica posibile interpretari ale textului, in urma unui comportament emergent. Tot in acest fel este eliminata necesitatea folosirii unui graf intreg (care include toate legaturile posibile intre sensuri)

- Executia algoritmului consta intr-un numar de cicluri potential infinit. Dupa fiecare ciclu se observa starea mediului ("*state of the environment*")
- Un ciclu se compune din urmatorii pasi:
  - Se elimina furnicile moarte si podurile care nu mai au feromon
  - Corespunzator fiecarui cuib se produce o furnica
  - Pentru fiecare furnica se determina modul in care se afla (cauta energie sau se intoarce); o facem sa se miste; se creeaza un pod de interpretare, daca este posibil ("*interpretative bridge*")
  - Se actualizeaza mediul – *update the environment* (nivelul de energie al nodurilor, feromon si vectori de miros – *odour vectors*)

Notatii principale pentru algoritm:

F_A – cuib care corespunde sensului A

f_A – furnica nascuta in cuibul F_A

V(X) – vector de miros asociat lui X (furnica sau nod)

E(X) – energia lui X (unde X poate fi furnica sau nod)

$Eval_f(N)$ – evaluarea unui nod N de catre o furnica f

$Eval_f(A)$ – evaluarea unei muchii A (cantitatea de feromon) de catre o furnica f

$\varphi_{(t/c)}(A)$ – cantitatea de feromon pe o muchie A la momentul t sau ciclul c

Parametri pentru algoritm:

Notatie	**Descriere**
E_a	Energia luata de o furnica atunci cand ajunge la un nod
E_{max}	Cantitatea maxima de energie pe care o poate cara o furnica
δ	Rata de evaporare a feromonului intre doua cicluri
E_0	Cantitatea initiala de energie din fiecare nod
ω	Durata de viata a furnicii
L_v	Lungimea vectorului de mirosuri
δ_v	Proportia de componente de miros depozitate de catre o furnica intr-un nod oarecare
c_{ac}	Numarul de cicluri al simularii

1. Nasterea furnicilor, moartea si modelul de energie

Initial se atribuie o cantitate fixa de energie, E_o, fiecarui nod al mediului. La inceputul fiecarui ciclu, fiecare nod cuib N are posibilitatea sa produca o furnica A, folosind o unitate de energie, cu probabilitatea $P(N_A)$. In literatura, aceasta probabilitate este definita ca fiind functia:

$$P(N_A)=\frac{\arctan(E(N))}{\pi}+0.5$$

unde E(N) este energia nodului N

Atunci cand este creata, o furnica are o durata a vietii alcatuita din **ω** cicluri. Atunci cand viata ei atinge valoarea 0, furnica este distrusa (stearsa) la inceputul urmatorului ciclu, iar energia ei este depozitata in nodul unde a murit. In felul asta se asigura pastrarea **echilibrului global de energie al sistemului**, ceea ce joaca un rol fundamental in convergenta catre o solutie.

2. Miscarile furnicilor

Miscarea furnicilor este aleatoare, dar influentata de mediu.

Atunci cand o furnica se afla intr-un nod, ea atribuie o probabilitate de tranzitie muchiilor care conduc spre toate nodurile vecine. Probabilitatea de a trece printr-o muchie A_j pentru a ajunge intr-un nod N_i este:

$$P(N_i, A_j)=\frac{Eval_f(N_i,A_j)}{\sum_{k=1,l=1}^{k=n,l=m} Eval_f(N_k,A_l)}$$

unde $Eval_f(N,A)=Eval_f(N)+Eval_f(A)$ este functia de evaluare a unui nod N atunci cand se ajunge in el venind de-a lungul muchiei A

O furnica nou-nascuta incepe sa caute mancare in felul urmator:

a) Este atrasa de nodurile care au cea mai mare cantitate de energie:

$Eval_f(N)=\frac{E(N)}{\sum_{i=0}^{m} E(N_i)}$

b) Evita sa mearga de-a lungul muchiilor cu mult feromon:

$Eval_f(A)=1-\varphi_t(A)$

unde $\varphi_t(A)$ este cantitatea de feromon a muchiei A la momentul t

pentru a favoriza o explorare mai ampla a spatiului de cautare

Furnica colecteaza atata energie (mancare) cat este posibil, pana cand decide sa o aduca inapoi acasa – si intra in “*return mode*” cu probabilitatea:

$$P(return)=\frac{E(f)}{E_{max}}$$

unde E(f) este energia pe care o are furnica f la momentul current

iar E_{max} este cantitatea maxima de energie pe care o poate transporta o furnica

Observatie: *Atunci cand o furnica f isi atinge capacitatea maxima de transport, probabilitatea ca ea sa se intoarca este 1.*

Cand decide sa se intoarca, se deplaseaza urmand statistic acele muchii care contin cel mai mult feromon:

$Eval_f(A)=\varphi_t(A)$

$Eval_f(A)$ reprezinta evaluarea muchiei A de catre furnica f (cantitate de feromon)

$\varphi_t(A)$ reprezinta cantitatea de feromon a muchiei A la momentul t

De asemenea, muchiile trebuie sa conduca spre noduri cu un miros (odour) apropiat de al ei:

$$Eval_f(N)=\frac{ExtLesk(V(N),V(f_A))}{\sum_{i=1}^{k} ExtLesk(V(N_i),V(f_A))}$$

V(N) – vector de miros asociat nodului N

$V(f_A)$ – vector de miros asociat furnicii nascute in nodul F_A

ExtLesk() – algoritmul Lesk extins

Observatie: Algoritmul Lesk extins se aplica asupra a doi vectori de miros.

3. Crearea si stergerea podurilor; tipuri de poduri

Atunci cand o furnica ajunge intr-un nod adiacent unui potential cuib prieten (care corespunde unui sens al cuvantului) trebuie sa decida intre a urma oricare dintre caile posibile sau a merge la acel nod cuib. Deci avem un caz particular al algoritmului de alegere a drumului de catre furnica, cazul cu:

$Eval_f(A)=0$ (i.e. feromonul muchiei este ignorat)

Singura diferenta este ca, daca furnica alege sa mearga la potentialul cuib prieten, se construieste un pod intre acest cuib si cuibul parinte al furnicii. Furnica parcurge acest pod pentru a se intoarce acasa.

Podurile se comporta ca si celelalte muchii, cu exceptia situatiei in care concentratia de feromon corespunzatoare lor atinge valoarea 0. In acest caz, podul se prabuseste si este inlaturat.

4. Modelul de feromon

Atunci cand se misca in graf, furnicile lasa dare de feromon de-a lungul muchiilor pe care le parcurg.

Furnicile au doua tipuri de comportament:

- cauta sa acumuleze energie
- vor sa se intoarca la cuibul mama

Miscarea furnicilor in graf este influentata de densitatea feromonului corespunzator fiecarei muchii: **ele prefera sa evite muchiile cu mult feromon atunci cand cauta energie** (hrana) si **le urmeaza** pe acestea atunci **cand vor sa transporte energia inapoi la cuibul mama**

Atunci cand se deplaseaza de-a lungul unei muchii A, furnicile lasa o dara depozitand o cantitate de feromon $\theta \in \mathbb{R}^+$ a.i.

$\varphi_{t+1}(A)=\varphi_t(A) + \theta$

In plus, , corespunzator fiecarui ciclu, exista o evaporare liniara a feromonului (care penalizeaza drumurile putin frecventate):

$\varphi_{t+1}(A)=\varphi_t(A) * (1 - \delta)$

unde δ reprezinta rata de evaporare a feromonului

5. Miros

Definitie: *Mirosul unui cuib este un vector de valori numerice asociat sensului respectiv (sensul pentru care s-a creat cuibul).* Acest vector numeric al sensului se determina in felul urmator:

- Se aplica Lesk extins (Banerjee &Pedersen, 2002), usor modifcat, astfel incat fiecare cuvant continut in oricare dintre definitii (glose) este indexat printr-un unic numar intreg. Modificarea consta in faptul ca suprapunerile de siruri de cuvinte nu primesc un scor mai mare ("la patrat"), cuvintele individuale fiind tratate in mod individual (suprapunere de tip "*bag of words*"). Aceasta se face pentru a micsora complexitatea de la O(m*n) la O(m), m>n, unde m si n sunt lungimile celor doua definitii care se compara. Prin aceasta indexare, intreaga definitie primeste un scor reprezentat printr-un vector.

Exemplu:

Definitia: “Some kind of evergreen tree”

S-a determinat ca:

“same” este indexat prin 123

“kind” este indexat prin 14

“evergreen” este indexat prin 34

“tree” este indexat prin 90

atunci reprezentarea indexata a intregii definitii este:

{14,34,90,123} (Observatie: vectorul este sortat)

Toate furnicile **nascute in acelasi cuib** au acelasi vector al mirosului. Atunci cand o furnica ajunge intr-un nod oarecare N, ea depune in acel nod unele dintre componentele vectorului sau de miros (urmand o distributie uniforma). Acestea vor fi adaugate la, sau vor inlocui componenta existenta a vectorului nodului, V(N).

Mirosul nodurilor cuib nu se modifica niciodata.

Acest mecanism permite furnicilor sa regaseasca drumul inapoi la nodul lor in cuib. Cu cat un nod este mai apropiat de un cuib dat, cu atat mai multe furnici ale acelui cuib au trecut prin acel nod si au depus componente de miros.

Prin urmare, mirosul unui nod va reflecta vecinatatea cuibului respectiv si va permite furnicilor sa gaseasca calea calculand scorul intre mirosul lor (cel al cuibului parinte) si mirosul nodurilor inconjuratoare. In urma acestui calcul, vor alege sa se deplaseze la nodul cu scorul cel mai mare.

Acest proces permite si existenta erorilor (de exemplu, o furnica ajunge in alt cuib decat cel propriu). Procesul este insa benefic pentru ca le determina pe furnici sa construiasca mai multe poduri.

Evaluare globala

- La sfarsitul fiecarui ciclu, se construieste configuratia curenta a problemei pe baza formei grafului: pentru fiecare cuvant, se alege sensul corespunzand cuibului avand cea mai mare cantitate de energie.
- In continuare, se calculeaza scorul global al configuratiei curente:
  - Scorul unui sens selectat al unui cuvant poate fi exprimat ca fiind suma scorurilor locale intre acel sens si sensurile selectate pentru toate celelate cuvinte ale unui context (o propozitie)
  - Pentru scorul global al configuratiei se aduna scorurile pentru toate sensurile selectate corespunzator cuvintelor textului:

$$Scor(C) = \sum_{i=1}^{m} \sum_{j=1}^{m} ExtLesk\left(w_{i,C[i]}, w_{j,C[j]}\right)$$

j=C[i] este sensul selectat j pentru cuvantul w_i

Observatie: Aici se vede ca nu se lucreaza cu o fereastra de context ci cu tot textul.

- Complexitate: $O(m^2)$, $\text{unde } m = \text{numarul}$ de cuvinte din text.

- In timpul executiei algoritmului se pastreaza configuratia care are scorul cel mai mare si care va fi folosita la sfarsit pentru a genera solutia.

Valorile uzuale ale parametrilor algoritmului (articolul original, 2012):

Notatie	Descriere	Valoare
E_a	Energia luata de o furnica atunci cand ajunge la un nod	1 – 30
E_{max}	Cantitatea maxima de energie pe care o poate cara o furnica	1 – 60
δ	Rata de evaporare a feromonului intre doua cicluri	0.0 – 1.0
E_0	Cantitatea initiala de energie din fiecare nod	5 – 60
ω	Durata de viata a furnicii	1 – 30 (cicluri)
L_v	Lungimea vectorului de mirosuri	20 – 200
δ_v	Proportia de componente de miros depozitate de catre o furnica intr-un nod oarecare	0 – 100%
c_{ac}	Numarul de cicluri al simularii	1 – 500

Valorile parametrilor nu pot fi determinate in mod analitic si au fost evaluate in mod experimental.

Experimentul care s-a facut a constat in adnotarea a 2269 cuvinte cu unul dintre sensurile lor posibile date de WordNet. Gradul mediu de polisemie intalnit a fost de 6.19 (S-a folosit corpusul de la SemEval 2007).

In urma simularilor si a testelor s-au gasit urmatoarele valori optime ale parametrilor:

$\omega = 25, E_a$=16, E_{max}=56, E_0=30, δ_v=0.9, δ=0.9, L_v=100

Testele din anul 2013 au condus la urmatoarele valori ale parametrilor:

- Pentru limba engleza

$\omega = 26, E_a$=14, E_{max}=3, E_0=34, δ_v=0.9775, δ=0.3577, L_v=25

- Pentru limba franceza

$\omega = 19, E_a$=9, E_{max}=3, E_0=32, δ_v=0.9775, δ=0.3577, L_v=25

Acuratete obtinuta (F1) pe datele SemEval-2007: **76.41%**

Algoritmul a fost superior celui mai bun "*state of the art*" sistem nesupervizat si celui mai slab sistem supervizat.

Caracteristicile algoritmului:

- Dezambiguizare globala (intregul text este context de dezambiguizare)
- Este de tip *knowledge based unsupervised* (se dau cunostinte din WordNet; procesul de invatare este de tip nesupervizat)

Algoritmul ShotgunWSD

Bibliografie

ShotgunWSD: An unsupervised algorithm for global word sense disambiguation inspired by DNA sequencing (Andrei M. Butnaru, Radu Tudor Ionescu, Florentina Hristea)

Cunostinte preliminare

- word embeddings
- sense embeddings
- inrudire semantica
- algoritmul Lesk extins (Banerjee & Pedersen)

Word embeddings

- se reprezinta fiecare cuvant sub forma unui vector de numere reale (de dimensiune mica), astfel incat cuvintele inrudite sa fie vecine(distanta dintre ele sa fie mica) in spatiul vectorilor generati
- se poate folosi toolkit-ul *word2vec* pentru a obtine word embeddings
- pentru acest algoritm s-a folosit word2vec pre-antrenat pe setul de date Google News folosind modelul Skip-gram
- modelul pre-antrenat contine vectori de dimensiune 300 pentru 3 milioane de cuvinte si fraze

Sense embeddings

- folosite pentru a masura **inrudirea intre doua synset-uri**

Calcularea scorului de inrudire:

- pentru fiecare cuvant din **vocabularul de dezambiguizare al unui sysnset** calculam vectorul de tip word embeddings, astfel se obtine un cluster de vectori pentru fiecare sysnset
- vectorii de tip sense embedding se obtin calculand **centroidul** fiecarui cluster (mediana pentru toti vectorii din cluster)
- inrudirea este data de masura cosinus de similaritate (**cosine similarity**) intre cei doi centroizi ai clusterelor celor doua sysnset-uri

Inrudire semantica – pentru determinarea vocabularului de dezambiguizare (1)

- **Utilizare pentru sense embeddings**
- Pentru o configuratie de n cuvinte (o fereastra de context) se calculeaza inrudirea semantica intre fiecare pereche de sensuri selectate
- in acest algoritm se calculeaza inrudirea semantica intre doua synset-uri
- pentru fiecare synset se construieste un **vocabular de dezambiguizare**

Inrudire semantica (2)

- **vocabularul de dezambiguizare** va contine:
  - toate sinonimele care formeaza synsetul respective, impreuna cu cuvintele (cu continut – *content words*) din glosa (definitie), inclusiv din exemplele de folosire a cuvantului
  - in cazul substantivelor, se adauga si hiponimele si meronimele (relatiile *hyponymy* si *meronimy*)
  - pentru adjective: se considera synset-urile inrudite prin relatiile: *similarity, antonymy, attributes, pertaining to* si *see also*
  - pentru adverbe: se considera synset-urile inrudite prin relatiile: *antonymy, pertaining to* si *topic*
  - pentru verbe: se considera synset-urile inrudite prin relatiile: *troponymy, hypernymy, entailment* si *outcome*

Inrudire semantica (3)

- vocabularul de dezambiguizare este procesat in continuare in felul urmator:
  - se elimina **stopwords**
  - pentru cuvintele care raman, se obtine *stem-ul* (radacina), aplicand **Porter stemmer**

- stem-urile vor fi folosite mai departe pentru a calcula inrudirea dintre synset-uri

Tehnica Shotgun de secventiere a ADN-ului

- secventierea ADN-ului se refera la determinarea succesiunii de nucleotide dintr-o molecula de ADN
- folosit in cercetare in domeniul geneticii pentru a obtine fragmente lungi de ADN
- tehnica a fost folosita inclusiv pentru secventierea genomului uman

Tehnica Shotgun – Pasi (1)

- inainte de a se putea citi o seventa lunga de ADN, este nevoie sa se creeze mai multe copii ale secventei respective
- ADN-ul e impartit in in mai multe segmente mici (de obicei continand intre 30 si 400 nucleotide).
- segmentele sunt apoi secventiate folosind tehnologia *Next-Generation Sequencing* (de exemplu cu ajutorul unei masini Illumina). Aceste fragmente se numesc *citiri* (reads).

Tehnica Shotgun – Pasi (2)

- se elimina fragmentele de calitate slaba
- intregul genom este reconstruit asambland fragmentele ramase. Doua sau mai multe fragmente sunt combinate (pentru a obtine segmente mai mari) atunci cand au subsecvente semnificative de nucleotide, care se suprapun. **Suprapunerea** se calculeaza obtinand "distanta" dintre ele (de exemplu, distanta Levenshtein, numita si distanta de editare)

Algoritmul ShotgunWSD – Caracteristici (1)

- are ca scop gasirea unei configuratii de sensuri G pentru intreg documentul D (**dezambiguizare globala**)
- o configuratie de sensuri se obtine asignand un sens fiecarui cuvant din text
- sensurile sunt selectate din WordNet
- pentru a folosi tehnica Shotgun se considera ca o configuratie de sensuri (pentru intreg documentul) corespunde ADN-ului care trebuie secventiat

Algoritmul ShotgunWSD – Caracteristici (2)

- configuratiile pentru ferestrele scurte de context (continand mai putin de 10 cuvinte) vor corespunde *citirilor* (reads) de ADN.
- o diferenta cruciala fata de tehnica Shotgun originala este ca de data asta cunoastem locatiile ferestrelor de context in cadrul documentului inca de la inceput, deci este mai usor sa fie asezate in reasamblarea finala.

Algoritmul ShotgunWSD – Pasi (1)

- in fiecare locatie posibila din document se selecteaza o fereastra de **n cuvinte**. (Valoarea lui n se alege experimental pentru a obtine cele mai bune rezultate)
- pentru fiecare fereastra de context se calculeaza toate posibilitatile de alocare a sensurilor
- se calculeaza un scor pentru fiecare configuratie de sensuri astfel obtinuta, folosind gradul de inrudire semantica intre sensurile cuvintelor

Algoritmul ShotgunWSD – Pasi (2)

- se folosesc doua metode de calculare a gradului de inrudire
  - masura Lesk extinsa
  - sense embeddings
- se pastreaza configuratiile cu cel mai mare scor pentru a fi asamblate. Se foloseste un parametru **c** pentru a determina **cate configuratii de sensuri sunt pastrate pentru o fereastra de context**

Algoritmul ShotgunWSD – Pasi (3)

- se gasesc apoi suprapunerile verificand daca prefixul unei configuratii coincide cu sufixul altei configuratii, pentru a fi asamblate in aceeasi secventa. **Lungimea l** a sufixului si prefixului trebuie, evident, sa fie mai mare decat 0 astfel incat macar un sens sa coincida in cele doua configuratii.
- in mod gradual se cauta sufixe si prefixe din ce in ce mai scurte, valoarea initiala a lui l fiind l=min{4, n-1}, unde n e lungimea ferestrei de context.

Algoritmul ShotgunWSD – Pasi (4)

- se continua asamblarea configuratiilor de sensuri (pentru a obtine secvente mai lungi) pana cand niciuna dintre secventele rezultate nu mai poate fi combinata cu vreo alta secventa
- in urma combinarii a doua secvente, trebuie recalculat scorul de inrudire (pentru secventa rezultata)
- **Observatie:** configuratiile mai lungi indica faptul ca a existat o potrivire intre sensurile alese, care s-a intins pe o bucata mai mare de text; prin urmare, configuratiile mai lungi au o probabilitate mai mare sa ofere sensuri corecte.

Algoritmul ShotgunWSD – Pasi (5)

- Dupa asamblare, se asigneaza sensurile cuvintelor din text. Pornind de la presupunerea ca secvetele mai lungi contin informatii mai bune (mai corecte) construim o lista ordonata (descrescator dupa lungimea secventelor) de configuratii pentru fiecare cuvant din document (se vor adauga in lista doar configuratiile care contin acel cuvant)
- Sensul final al fiecarui cuvant se obtine pe baza unui "vot" majoritar provenit de la **primele k configuratii** din lista ordonata, unde k e un parametru care este setat experimental pentru rezultate optime.

Algoritmul ShotgunWSD - Notatii

- $X=(x_1, x_2, ..., x_m)$ reprezinta un vector (multime ordonata de elemente)
- lungimea lui X e notata cu $|X| = m$
- consideram ca vectorii incep de la pozitia 1, deci $X[i] = x_i \; \forall \; i \in \{1, 2, ..., m\}$

ShotgunWSD - input

Input:

D = (w_1, w_2, ..., w_m) un document de m cuvinte notate cu w_i, unde i ∈ {1, 2, ..., m}

n = lungimea ferestrelor de context (1 < n < m)

k = numarul de configuratii de sensuri luate in considerare pentru votare (k > 0)

Parametri

Algoritmul are 3 parametri ale caror valori sunt determinate experimental pentru a obtine rezultate optime

- **c** – numarul de **configuratii de sensuri pastrate pentru o fereastra de context**
- **n** – dimensiunea **ferestrei de context**
- **k** – numarul de **configuratii folosite in procesul de votare** al celui mai probabil sens

Parametrul c

Parametrul c a fost setat la valoarea 20 pentru a avea un numar rezonabil de configuratii (din care putem alege) pentru pasii urmatori.

In felul acesta nu se foloseste foarte multa memorie sau timp de procesare

Parametrul n

- pentru setarea parametrilor n si k se folosesc sense embeddings pentru a calcula scorul de inrudire semantica
- acuratetea maxima pentru ShotgunWSD se poate obtine prin forta bruta, explorand toate configuratiile posibile de sensuri pentru cuvintele din document. Marind valoarea lui n, algoritmul se apropie din ce in ce mai mult de aceasta acuratete, insa timpul creste exponential in functie de valoarea lui n
- De exemplu, algoritmul ruleaza in 15 secunde pentru n=4 si in 1892 secunde pentru n=9 (deci ajunge sa fie de 120 de ori mai lent fata de n=4). S-a ales n=8, avand un timp rezonabil de rulare, de 187 secunde.

Parametrul k

- parametru k nu are o influenta semnificativa asupra timpului de rulare a algoritmului, astfel ca i-au fost alese valori pentru a obtine un scor F_1 cat mai bun
- Dintre valorile testate {1, 3, 5, 10, 15, 20} pentru SemEval 2007 s-a obtinut scorul cel mai mare (79.38%) pentru k=15

Datele pentru testare

Corpusurile:

- **SemEval 2007 (Coarse-grained english all-words)** care cuprinde 5 documente continand 2269 cuvinte ambigue (1108 substantive, 591 verbe, 362 adjective, 208 adverbe)
- **Senseval-2 (English all-words)** - cuprinde 3 documente cu 2473 cuvinte ambigue (1136 substantive, 581 verbe, 457 adjective, 299 adverbe)
- **Senseval-3 (English all-words)** - cuprinde 3 documente cu 2081 cuvinte ambigue (951 substantive, 751 verbe, 364 adjective, 15 adverbe)

Rezultate pentru SemEval 2007

Metoda	**Scorul F1**
Most Common Sense (MCS)	78.89%
Algoritmi genetici	74.53%
Colonie de furnici	79.03%
ShotgunWSD + Lesk extins	79.15%
ShotgunWSD + Sense Embeddings	79.38%

Observatie: Pe acest corpus s-au obtinut rezultatele cele mai bune.

S-au folosit ferestre de lungime n=8 cuvinte si votul s-a bazat pe primele k=15 configuratii

Rezultate pentru Senseval - 2

Metoda	**Scorul F1**
Most Common Sense (MCS)	60.10%
ShotgunWSD + Lesk extins	56.28%
ShotgunWSD + Sense Embeddings	56.42%

S-au folosit ferestre de lungime n=8 cuvinte si votul s-a bazat pe primele k=15 configuratii

Rezultate pentru Senseval - 3

Metoda	**Scorul F1**
Most Common Sense (MCS)	62.30%
ShotgunWSD + Lesk extins	58.84%
ShotgunWSD + Sense Embeddings	58.95%

S-au folosit ferestre de lungime n=8 cuvinte si votul s-a bazat pe primele k=15 configuratii

Anexa

Algoritmul ShotgunWSD - Notatii

- $X=(x_1, x_2, ..., x_m)$ reprezinta un vector (multime ordonata de elemente)
- lungimea lui X e notata cu $|X| = m$
- consideram ca vectorii incep de la pozitia 1, deci $X[i] = x_i \; \forall \; i \in \{1, 2, ..., m\}$

ShotgunWSD - pseudocod

Input:

D = (w_1, w_2, ..., w_m) un document de m cuvinte notate cu w_i, unde i ∈ {1, 2, ..., m}

n = lungimea ferestrelor de context (1 < n < m)

k = numarul de configuratii de sensuri luate in considerare pentru votare (k > 0)

ShotgunWSD - pseudocod

Initializare

c ⟵ 20;

for i ∈ {1, 2, ..., m} do

S_{w_i} ⟵ multimea synset-urilor din WordNet ale cuvantului w_i;

S ⟵ ∅;

G ⟵ (0, 0, ..., 0) astfel incat |G| = m;

Calculare

for i ∈ {1, 2, ..., m-n+1} do

$C_i \leftarrow \emptyset$;

while (nu s-au generat toate configuratiile de sensuri) do

C ⟵ o configuratie noua (s_{w_i}, $s_{w_{i+1}}$, ..., $s_{w_{i+n-1}}$), $s_{w_j} \in S_{w_j}$, ∀ j ∈ {i, ..., i+n-1}, astfel incat C ∉ C_i

r ⟵ 0;

for p ∈ {1, 2, ..., n-1} do

for q ∈ {p+1, 2, ..., n} do

r ⟵ r + inrudire (C[p], C[q]);

$C_i \leftarrow C_i \cup \{(C,i,n,r)\}$;

C_i ⟵ primele c configuratii obtinute sortand descrescator configuratiile din C_i dupa scorul de inrudire;

S ⟵ S ∪ C_i

for $l \in \{\min\{4, n-1\}, ..., 1\}$ do

for $p \in \{1, 2, ..., |S|\}$ do

$(C_p, i_p, n_p, r_p) \longleftarrow$ elementul de pe pozitia p din S;

for $q \in \{1, 2, ..., |S|\}$ do

$(C_q, i_q, n_q, r_q) \longleftarrow$ elementul de pe pozitia q din S;

if $i_q - i_p < n_p$ si $i_p \neq i_q$ then

$t \longleftarrow$ true;

for $x \in \{1, ..., l\}$ do

if $C_p[n_p - l + x] \neq C_q[x]$ then

$t \longleftarrow$ false;

if t = true then

$C_{p \oplus q} \longleftarrow (C_p[1], C_p[2], ..., C_p[n_p], C_q[l+1], C_q[l+2], ..., C_q[n_q])$;

$r_{p \oplus q} \longleftarrow r_p$;

for $i \in \{1, 2, ..., n_p + n_q - l\}$ do

for $j \in \{l+1, l+2, ..., n_q\}$ do

$r_{p \oplus q} \longleftarrow r_{p \oplus q}$ + inrudire($C_{p \oplus q}[i], C_q[j]$);

$S \longleftarrow S \cup \{(C_{p \oplus q}, i_p, n_p + n_q - l, r_{p \oplus q})\}$;

Alocarea sensurilor finale

for j ∈ {1, 2, ..., m} do

$Q_j \longleftarrow \{(C, i, d, r) \mid (C, i, d, r) \in S, j \in \{i, i + 1, ..., i + d - 1\}\}$;

$Q_j \longleftarrow$ primele k configuratii obtinute sortand configuratiile din Q_j descrescator dupa lungime;

$ps_{w_j} \longleftarrow$ sensul predominant obtinut folosind o schema de vot majoritar pe Q_j;

$G[j] \longleftarrow ps_{w_j}$

Output

$G = (ps_{w_1}, ps_{w_2}, ..., ps_{w_m})$, $ps_{w_i} \in s_{w_i}$, $\forall i \in \{1, 2, ..., m\}$ – reprezinta configuratia globala a sensurilor returnata de algoritm