The Computational Complexity of Rule-Based Part-of-Speech Tagging

Karel Oliva[1], Pavel Květoň[2] and Roman Ondruška[2]

[1] Austrian Research Institute for Artificial Intelligence (OeFAI),
Schottengasse 3, A-1010 Wien, Austria
karel@oefai.at
[2] Institute of Formal and Applied Linguistics,
Faculty of Mathematics and Physics, Charles University,
Malostranské nám. 25, CZ-118 00 Praha 1 — Malá Strana, Czech Republic
kveton@ufal.ms.mff.cuni.cz

Abstract. The paper deals with the computational complexity of Part-of-Speech tagging (aka morphological disambiguation) by means of rules derived from *loosened negative n-grams.* Loosened negative n-grams [2] were originally developed as a tool for the task of pure verification of results of Part-of-Speech tagging (corpus quality checking). It is shown that while the verification is just a polynomial problem, the time consumed by the tagging (disambiguation) task cannot be bounded by a polynom in the general case. The results presented in the paper are relevant above all for disambiguation performed by means of Constraint-based Grammars [1] and similar frameworks, which are in fact only notational variants of the rules derived via loosened negative n-grams. Throughout the paper some familiarity with finite-state automata (FSA) and the class of NP problems is assumed.

1 Introduction

The linguistic core of the rule-based approach to Part-of-Speech (PoS) tagging can be seen in the simple idea of *negative bigrams.* A negative bigram is a pair of PoS tags (morphological analyses) of adjacent words which constitute an incorrect configuration in a text of a particular language. For example, the pair ARTICLE+FIN_VERB constitutes a negative bigram in English, since in no (correct) English sentence an article can be immediately followed by a finite verb. Originally, the idea of negative bigrams was developed for the purpose of detection of tagging errors in corpora [2], but it is obvious that after a small modification it can be used also for the purpose of disambiguation (tagging). In particular, in a situation where one of the words has a unique (i.e. safe) morphological reading while the other member of the pair is ambiguous (i.e. taken alone, it offers a number of different possible morphological interpretations), some of the readings of the ambiguous word can be discarded on the basis of the negativity of the bigrams which occur as combination of the readings of the two words. In the above English example, such an approach can be applied for deciding on

the noun/verb ambiguity, e.g., in the pairs *the advices, the smells, the use, ...* where obviously the words *advices, smells, use, ...* cannot be finite verbs (since otherwise the negative bigram ARTICLE+FIN_VERB would occur).

The idea of disambiguation by negative bigrams can be easily generalized in two directions. The first one is the usage of negative trigrams, tetragrams, pentagrams, ..., i.e. of configurations of length 3,4,5, ... which cannot occur as (correct) morphological analysis of sequences of adjacent words. Thus, for example, the configuration FIN_VERB+FIN_VERB+FIN_VERB is a negative trigram in English, since sentences of the type *The mouse the cat the dog chased caught survived* are considered unacceptable (the configuration FIN_VERB+FIN_VERB does not constitute a negative bigram, however — cf. the correct (acceptable, grammatical) sentence *The man Mary loves came late*).

The second possible generalization consists in the idea that the "negativity" of the negative bigram (or trigram, tetragram, etc.) can be in certain cases kept also if its parts are separated by some other lexical material which, on the one hand, has to meet certain conditions but, on the other hand, on the amount of which no limits are posed. For example, the configuration ARTICLE+FIN_VERB remains incorrect even after any number of adverbs and/or personal pronouns and/or prepositions and/or conjunctions is placed in-between the two constituting members of the negative bigram (and, hence, indeed the limitation is only qualitative, but not a quantitative one). Such configurations are called *loosened n-grams* (e.g., the configuration

$$\text{ARTICLE}+\{\text{ADV,PERS\_PRON,PREP,CONJ}\}^*+\text{FIN\_VERB}$$

is an example of a loosened bigram). Such a loosening of configurations of fixed length into configurations of arbitrary length provides for an extremely powerful disambiguation tool.

In such a way, the original idea behind the negative n-grams could be developed into a tool for a partial disambiguation of a corpus, i.e. for removing at least some of the tags within an ambiguous corpus (typically, a corpus which underwent a morphological analysis, i.e. a corpus each word of which has been assigned all its morphological readings). Such a partially disambiguated corpus can further serve as an input for a stochastic tagger.

The rest of the paper is organized as follows:

- in Section 2, we present the loosened negative n-grams in more detail and prove that the verification of a non-ambiguous sentence is a polynomial task;
- in Section 3, we define a very transparent but time-exponential algorithm for total conservative partial disambiguation using loosened negative n-grams.
- finally, in Section 4, we define several formal languages representing diverse types of partial disambiguation and we show that all these languages are NP-hard to recognize.

2 Disambiguating with negative n-grams

Throughout the paper it is assumed that T is a *tagset*, i.e. a set of (morphological) tags of a particular natural language.

Definition 1. *A* sentence *(of length k) is a sequence $W_1, W_2, \ldots, W_k$, where $W_i \subseteq T$ holds for all $1 \leq i \leq k$.*

The sets W_i denote the words *of the particular sentence. A word W_i is* non-ambiguous *if the set W_i contains at most one tag.*

A sentence is non-ambiguous *if every its word is non-ambiguous. Otherwise the sentence is* ambiguous.

Definition 2. *A sentence $S' = W'_1, \ldots, W'_m$ is a* partial disambiguation *of sentence $S = W_1, \ldots, W_k$ whenever $m = k$ and $W'_i \subseteq W_i$ for all $1 \leq i \leq k$.*

The disambiguation algorithms discussed below expect a sentence on their input and their result is a partial disambiguation of this input sentence — in other words, their task is to remove some tags from each word in the sentence.

Definition 3. *By a* reading *of a sentence S we denote any non-ambiguous partial disambiguation of S.*

It can be immediately seen that an ambiguous sentence contains generally an exponential number of readings (in the length of sentence).

Definition 4. *The tag sequence $t_1, t_2, \ldots, t_n$ ($n \in N$, $t_i \in T$ for all $1 \leq i \leq n$) is called a* negative n-gram *if the sequence of words $\{t_1\}, \{t_2\}, \ldots, \{t_n\}$ cannot be found as a sub-sequence of any well-formed non-ambiguous sentence.*

Definition 5. *The sequence $b = t_1, C_1, t_2, C_2, \ldots, t_{n-1}, C_{n-1}, t_n$ ($n \in N$, $t_i \in T$ (for $1 \leq i \leq n$) and tag sets C_i (for $1 \leq i < n$)) is called a* loosened negative n-gram *if any non-ambiguous sentence S containing a subsequence*

$$\{t_1\}W_1^1 W_1^2 \ldots W_1^{n_1}\{t_2\}W_2^1 \ldots W_2^{n_2}\{t_3\} \ldots \{t_{n-1}\}W_{n-1}^1 \ldots W_{n-1}^{n_{n-1}}\{t_n\} \quad (1)$$

such that $W_i^j \subseteq C_i$ holds for all $1 \leq i < n$, $1 \leq j \leq n_i$ is not *a well-formed sentence (of the particular language).*

We denote $|b|$ to be the length of the n-gram b (i.e. $|b|=n$, in this case).

If a sentence S contains the sub-sequence (1), we say that S matches *the loosened negative n-gram b.*

In other words, if the tags of the (original) negative n-gram $t_1, t_2, \ldots, t_n$ are found in a non-ambiguous sentence in the correct order, but some words in-between them disturb the direct sub-sequentiality of t_i's, then *loosening contexts* C_i are taken into consideration — these contexts are defined in such a way that if all positions (in any amount) in-between t_i and t_{i+1} contain only the tags from C_i, the configuration remains impossible.

Definition 6. *A reading R of a sentence S is* compatible *with a set B of loosened negative n-grams if R matches no element of B. Any reading R containing a word $\emptyset$ (word with no tag) is said to be* incompatible *with any set of negative n-grams.*

Proposition 1. *Let B be a set of loosened negative n-grams and S a non-ambiguous sentence. The compatibility of S and B can be verified on a deterministic Turing machine (DTM) in polynomial time (in the size of S and B).*

Proof. The basic idea is to convert (in polynomial time) the input set B to non-deterministic FSA A that will try to find the occurence of an n-gram from B in S and then simulate A on S by DTM in polynomial time.

The automaton $A = (Q, X, \delta, I, F)$ is defined as follows. The set of states is $Q = \{q, g\} \cup \bigcup_{b \in B} Q_b$, where $Q_b = \{q_b^i | 1 \leq i \leq |b|\}$ and q, g are states different from all other states. The set of input symbols is $X = \{\{t\} | t \in T\} \cup \{\emptyset\}$. The set of initial states is $I = \{q\}$. The set of final (accepting) states is $F = \{q_b^{|b|} | b \in B\} \cup \{g\}$. The state g is the garbage state — any undefined transition leads to this state (accepting).

The transition function δ is defined separately for q and for the states arising from the n-grams. For the state q, the transitions over every $\{t\}$ for $t \in T$ loops to q. Moreover, if t is the first tag of an n-gram b, there is also a transition from q to q_b^1. From a state $q_b^k \in Q$, for $k < |b|$ and $b = t_1 C_1 \ldots C_{|b|-1} t_{|b|}$, there is a transition to q_b^{k+1} over $\{t_{k+1}\}$ as well as loops to q_b^k over every $\{t\}$ such that $t \in C_k$. The transitions over every $\{t\}$ such that $t \notin C_k$ and $t \neq t_{k+1}$ lead back to q. From $q_b^{|b|}$, there is just a loop over every $\{t\}$ such that $t \in T$.

The automaton A first looks for possible start of an n-gram $b \in B$ (in state q). Once A decides the b has started, it tries to match it on the sentence through the states q_b. If b matches, it waits in $q_b^{|b|}$ for the end of input (or at most switches to g over $\emptyset$). If b does not match, A returns back to q and tries the next match.

If the automaton finishes in $q_b^{|b|}$, the sentence is accepted (contains an n-gram from B). If the automaton finishes in g, the sentence is somehow corrupted (and accepted), thus the sentence is accepted by A if and only if the sentence is not compatible with B.

It can be easily seen that the construction of A has a polynomial time complexity (in the sizes of B and T).

Now, the work of the non-deterministic FSA $A = (Q, X, \delta, I, F)$ on the input non-ambiguous sentence $S = W_1, \ldots, W_k$ can be simulated by the following algorithm:

1. *Start:* Initialize a new set $M = I$;
2. for $c = 1, \ldots, k$ do the following steps:
   (a) initialize a new set $M' = \emptyset$;
   (b) for each $q \in M$ let $M' = M' \cup \delta(q, W_c)$;
   (c) let $M = M'$;
3. if $M \cap F = \emptyset$ then the algorithm accepts the input sentence (it neither matches any n-gram from B nor contains an empty word), or refuses it otherwise.

The algorithm consumes on DTM the time of at most $k \times |Q|^3$ and accepts the input if and only if the sentence S is compatible with B. □

Definition 7. *A partial disambiguation S' of the sentence S is* conservative *w.r.t. a set B of loosened negative n-grams if every reading R of S such that R is compatible with B is also present in S'.*[3]

Definition 8. *A partial disambiguation S' of the sentence S is* total *w.r.t. a set B of loosened negative n-grams if every tag t from S' is present in some reading of S which is compatible with B.*[4]

Trivially, any algorithm that does nothing (returns always the input sentence as its output) produces always a conservative disambiguation, while any algorithm that deletes everything (all tags of all words) produces always a total disambiguation. The core of the problem is, however, how to produce a disambiguation which is both conservative and total.

It is worth mentioning on this spot that with respect to a total conservative tagging result (which is unique, for a given sentence and a given set of negative n-grams), any disambiguation which is conservative has a 100% recall, while any total disambiguation has a 100% precision.

3 The generic disambiguator

In this section, we define the *generic disambiguator* algorithm, which is a trivial, but time-exponential instance of a disambiguation algorithm producing total conservative disambiguations.

Let S be an input sentence of length k and let B be the input set of negative n-grams. In such case, let the following algorithm be defined:

1. *Start:* Let $S' = W_1 \dots W_k$ be set to a sentence of length k with all words empty, i.e. $W_i = \emptyset$ holds for every $1 \leq i \leq k$;
2. if there exists a reading $R = V_1 \dots V_k$ of the sentence S that has not been investigated yet, continue with the next step; otherwise, return S' and finish;
3. check the reading R using n-grams from B; if R is not compatible with B, continue with step 2;
4. if R is compatible with B, then for every $1 \leq i \leq k$ set $W_i = W_i \cup V_i$, and continue with step 2.

It is obvious that every result of the generic disambiguator is a total and conservative disambiguation, since every tag t of a word W from S is present in the output S' if and only if t is contained in some compatible reading R and every compatible reading R from S is contained in S'.

On the other hand, it is also obvious that the loop over all the readings of the input sentence S has to be passed an exponential number of times (in the sentence length).

[3] In other words: any algorithm which issues a conservative disambiguation must not delete any tag taking part in a compatible reading.

[4] In other words: any algorithm which issues a total disambiguation must delete all tags which do not take part in some compatible reading. Please note that *tag* in the definition means the particular tag in the particular word, not any tag from T.

4 NP-hardness

In this section several formal languages based on the various types of partial disambiguation are defined and shown to be NP-hard to recognize. First, however, some basic definitions from the theory of NP-completeness are recalled.

Definition 9. *A formal language L (over an alphabet X) belongs to the* class NP *if and only if the problem $(x \in L)$ can be decided by a non-deterministic Turing machine (NTM) T in polynomial time (in the size of x) for every $x \in X^*$ — if there is at least one accepting computation of T, then $x \in L$, otherwise $x \notin L$.*

A language L (over an alphabet X) is polynomially-Turing-reducible *to a language M (over Y) if the problem $(x \in L)$ can be decided by DTM in polynomial time (in the size of x) with oracle M[5] for all $x \in X^*$.*

A language L is NP-hard *whenever every language $M \in NP$ is polynomially-Turing-reducible to L. L is* NP-complete *if it is NP-hard and $L \in NP$.*

If M is NP-complete language polynomially-Turing-reducible to a language L, then L is NP-hard.

Definition 10. *We define the following formal languages (T is a tagset, S, S' are sentences, B is a set of loosened negative n-grams, all based on T):*

$\text{LVALID} = \{(T, B, S) | S \textit{ contains a reading compatible with } B\}$
$\text{LCONSERVATIVE} = \{(T, B, S, S') | S' \textit{ is conservative disamb. of } S \textit{ w.r.t. } B\}$
$\text{LTOTAL} = \{(T, B, S, S') | S' \textit{ is total disambiguation of } S \textit{ w.r.t. } B\}$
$\text{LBEST} = \{(T, B, S, S') | S' \textit{ is total conservative disambiguation of } S \textit{ w.r.t. } B\}$

We shall show that the above languages (except LBEST) are NP-hard.

Proposition 2. *The problem of recognition of* LVALID *is NP-hard.*

Proof. NP-hardness of LVALID will be proved by a polynomial Turing-reduction from the problem SAT (satisfiability of logical formula in conjunctive normal form (CNF)). Determining whether a given formula in CNF is satisfiable or not is a well-known NP-complete problem (cf. [3]).

Let F be a formula in CNF, i.e. F is a conjunction of several simpler formulae $F_1, \ldots, F_k$. Each formula F_i (for $1 \leq i \leq k$) is a disjunction of logical variables and/or their negation, e.g. $(a \vee b \vee \neg c) \wedge (\neg a \vee b) \wedge (c \vee \neg b)$. In the example, the formula can be satisfied by setting b and c to *true* and a to anything.

The input formula F of the problem SAT can be transformed to the input of LVALID as follows. The tagset T contains exactly all the logical variables of F and their negation (in the example, $T = \{a, b, c, \neg a, \neg b, \neg c\}$).

The set B will contain all loosened negative bigrams in the forms $x, T, \neg x$ and $\neg x, T, x$ for every logical variable x in F. In the example, the set B is hence defined as follows:

$$B = \{(a, T, \neg a), (b, T, \neg b), (c, T, \neg c), (\neg a, T, a), (\neg b, T, b), (\neg c, T, c)\}.$$

[5] The oracle M is expected to decide the problem $(y \in M)$ for any $y \in Y^*$. During the computation the oracle can be asked any number of times and the time consumed by the oracle itself is not counted into the total time consumed by the DTM in question.

The input sentence S will contain k words $W_1, \ldots, W_k$. Each word W_i arises from F_i by assigning all logical variables and their negations in F_i as tags to W_i.

It can be immediately seen that the transformation of F into (T, B, S) is a polynomial task (in the size of the set of all logical variables used in the original formula).

Now, if there exists a reading R of S that is compatible with B, then the formula F is satisfiable since the reading R does not contain both a logical variable and its negation (this is a trivial consequence of the construction of B).

On the other hand, if no reading R of S is compatible with B, then every selection of single token (variable or its negation) from each sub-formula F_i (over all i) will contain both a logical variable and its negation and hence F cannot be satisfied. □

Corollary 1. LVALID *is NP-complete.*

First, take an input (T, B, S) of LVALID to be decided. Let Q be a NTM that works as follows: it non-deterministically selects one reading R of S and verifies (via Proposition 1) the compatibility of R with B. Now $(T, B, S) \in$ LVALID if and only if there exists an accepting computation of Q, hence LVALID $\in$ NP. The Proposition 2 completes the proof. □

Proposition 3. LCONSERVATIVE *is NP-hard.*

Proof. Let (T, B, S) be an input of LVALID and let S' be a sentence of the same length as S with all words empty. Now note that S' is conservative partial disambiguation of S w.r.t. B if and only if no reading R of S is compatible with B. Thus (T, B, S) belongs to LVALID if and only if (T, B, S, S') is not in LCONSERVATIVE. Hence LVALID has been Turing-reduced to LCONSERVATIVE. □

Proposition 4. LTOTAL *is NP-hard.*

Proof. Let (T, B, S) be an input of LVALID. If no reading of S is compatible with B, the only total disambiguation is the sentence with all words empty. If there exists a reading of S compatible with B, non-empty total disambiguation exists.

Let n be the length of S. For each word W_i of S and every tag $t \in W_i$ we create a sentence $S_i^t = \emptyset, \ldots, \emptyset, \{t\}, \emptyset, \ldots, \emptyset$ of length n, where $\{t\}$ is on the position i. Each of these sentences will be passed as an input (T, B, S, S_i^t) to LTOTAL. The initial input (T, B, S) belongs to LVALID if and only if any of (T, B, S, S_i^t) belongs to LTOTAL.

Because there is a linear amount of sentences S_i^t arising from S, the Turing reduction is finished. □

5 Conclusions and future work

For the theory of NP-completeness, it could be interesting to determine whether LBEST is also an NP-hard problem and whether LCONSERVATIVE, LTOTAL and LBEST are NP-complete.

In practice, however, we rather study the complexity of algorithms that perform the disambiguation itself, mainly the algorithms that achieve total conservative disambiguation of the input. As for these disambiguators, the NP-hardness of LVALID implies[6] non-existence of deterministic polynomial total conservative disambiguator (i.e. if there exists such a total conservative disambiguator, LVALID could be solved deterministically in polynomial time).

Moreover, in practice we usually deal with a fixed set of negative n-grams and the real input is only the sentence to be disambiguated, hence the set of n-grams could be possibly pre-compiled before the disambiguation itself. We have only shown that any polynomial pre-compilation still keeps the problems NP-hard. Hence, it remains a matter of further research whether n-grams could be pre-compiled (in an exponential time) into some machine that will achieve total conservative disambiguation of the input sentence in polynomial time. In other words, building a total conservative disambiguator using loosened n-grams more efficient than the generic one is still an open problem.

Acknowledgment

This work has been sponsored in part by the *Fonds zur Foerderung der wissenschaftlichen Forschung (FWF)*, Grant No. P12920, by the *Grant Agency of the Czech Republic (GACR)*, Grant No. 405/03/0913, and by the *Czech Ministry of Education, Youth and Sports (MSMT)*, Grant No. LN00A063.

The *Austrian Research Institute for Artificial Intelligence (OeFAI)* is supported by the *Austrian Federal Ministry of Education, Science and Culture.*

References

1. Karlsson F., A. Voutilainen, J. Heikkila, and A. Antilla (eds.): *Constraint Grammar - A Language-Independent System for Parsing Unrestricted Text*, Mouton de Gruyter, Berlin & New York 1995
2. Květoň, P. and K. Oliva: *Achieving an Almost Correct PoS-Tagged Corpus.* In: "Proceedings of the 5th international conference Text, Speech and Dialogue TSD 2002", Lecture Notes in Artificial Intelligence vol. 2448, Springer, Berlin 2002, pp. 19-26
3. Garey, M. R. and D. S. Johnson: *Computers and Intractability: A Guide to the Theory of NP-completeness.* W. H. Freeman, San Francisco, 1979

[6] With the silent assumption of NP≠P.