Decision Strategies for Incremental POS Tagging

Niels Beuck, Arne Köhn and Wolfgang Menzel
Department Informatik, University of Hamburg
Vogt-Kölln-Straße 30, 22527 Hamburg, Germany
{beuck, 5koehn, menzel}@informatik.uni-hamburg.de

Abstract

In an incremental NLP pipeline every module needs to work incrementally. However, an incremental processing mode can lead to a degradation of accuracy due to the missing context to the right. We discuss three properties of incremental output that can be traded for accuracy, namely *timeliness*, *monotonicity* and *decisiveness*. The consequences of these trade-offs are evaluated systematically for the task of part-of-speech tagging.

1 Introduction

Incremental language processing does not consume input at once but in a word-by-word manner and a sequence of incomplete, but successively more complete interpretations is generated as output. Such a processing mode is especially beneficial in scenarios where language input evolves over time like in human-computer or human-robot interaction. By processing the input while it is still incomplete, a speed-up can be achieved by using production time as processing time. Another benefit is the possibility to immediately respond to partial input, e.g. by providing non-verbal feedback to the speaker. This requires the system to be able to produce a partial analysis for partial input, which, of course, needs to be available early enough. Otherwise, the receiver could as easily wait until the whole utterance is complete and the benefit of incremental processing vanishes.

In natural language the correct interpretation of a word often depends on the context to the right. Therefore, an incremental NL processor that is selecting an interpretation for a word without being aware of the right context is likely to select the wrong interpretation more often than a non-incremental processor. There are several possible strategies to deal with this problem. In the first part of this paper we will discuss these strategies and how they can be implemented for a simple NLP task, namely part-of-speech (POS) tagging. In the second part we will present a quantitative evaluation and compare the different trade-offs made by the different strategies. To our knowledge there is no previous work evaluating these trade-offs systematically.

POS tagging is particularly attractive for an initial investigation of incremental processing behavior, as in this task the input consists of a small number of discrete tokens (in contrast to speech recognition) where each of them has to be mapped to exactly one output token (in contrast to syntactic or semantic structures which combine several input tokens). Moreover POS tagging does not depend on previous processing steps. Therefore, the results of this paper provide the basis for a broader range of investigations with more complex NLP tasks.

In Section 2 we will define the notion of incrementality used in this paper. Based on this, the challenges of disambiguation in incremental NLP and several strategies to meet them are discussed in Section 3. Section 4 describes the POS taggers used in the evaluation. The results of the evaluation are presented and discussed in Section 5. We finish with an outlook on future work in Section 6.

2 Incrementality

Incremental processing can be realized for procedures that take a sequence of input tokens and generate a sequence of output tokens[1]. The output is called the analysis of the input. Depending on the task an input token could be a word from an utterance perhaps accompanied by additional morphosyntactic information like a POS tag. Exam-

[1]In contrast to what Wirén (1992) defines as full incrementality, we only regard left-right-incremental processes here. This restricts the modifications of the input to simply adding new tokens at the end of the sequence.

ples for output tokens could be phrases, chunks, POS tags associated to the input words or dependencies to other words.

Incrementality is not a binary feature but comes in different grades and flavors. First of all, we have to distinguish between incremental interfaces and incremental algorithms. According to Wirén (1993) an algorithm is incremental *"if it uses information from an old analysis in computing the new analysis."* The interface is incremental, if it accepts partial input and provides partial output. On the one hand, a process could provide an incremental interface without a corresponding algorithm by applying a non-incremental algorithm on successively extended prefixes of the input sequence. On the other hand, an incremental algorithm does not automatically facilitate an incremental interface.

With respect to the kind of interface we can distinguish incremental input consumption (IIC) from incremental output production (IOP) (Kilger and Finkler, 1995). While the former is the ability to start processing on partial input, the latter is the ability to produce a sequence of partial analyses and provide them as output as soon as they become available.

A system without IIC will have to wait with the computation until the input is complete and a system without IOP will not start to generate output before the input is complete. That means, a combination of both processing modes is needed to be able to provide output while the input has not yet been completed. We shall call a system that applies both, IIC and IOP, to be input/output-incremental or IO-incremental. For a system consisting of several processing modules to be IO-incremental, each of its modules has to be IO-incremental, otherwise the final output of the system is delayed at least until the end of the input.

3 Disambiguating in incremental processing

In many applications, like POS tagging or syntactic parsing, the not yet seen input may have an influence on the analysis of the current input token. Therefore, incremental processing has only a limited access to the disambiguating context information. There are several strategies to mitigate the impact of limited context information on the output accuracy but they affect other properties of the output. In addition to accuracy we have identified three other parameters, namely timeliness, monotonicity and decisiveness, necessary to sufficiently characterize incremental processing behavior results. Timeliness and monotonicity are features unique to incremental output while decisiveness can also be applied to non-incremental output.

Timely output is generated for every input increment before the next input token is available. **Delayed** output lags behind the input stream. The delay can be a fixed number of tokens due to a lookahead of a fixed window size, or a dynamic range as in stack based approaches.

Monotonicity is given if the output stream can only be modified by adding new information. Once committed information may not be changed later on. In contrast, **non-monotonic** input implies that previous output can be revoked or changed. By allowing non-monotonic updates intermediate output becomes unreliable to a certain degree.

Decisiveness is the property of the system to commit to one analysis at a time. In contrast to this, **inconclusive** output consists of several possible alternatives. These alternatives can be stated implicitly by leaving features of the output unspecified, by enumerating alternatives for each token or by explicitly representing alternatives for the overall output.

Delay and non-monotonicity can be interpreted as a gradual reduction of IIC and IOP respectively. A process which cannot start before the input sequence is complete behaves like a process without IIC. On the other hand, a process with extreme non-monotonic behavior cannot guarantee persistence. This leads to a high degree of unreliability that might render all intermediate partial analyses except the very last one useless to their consumer. Such a module has to be considered as not exhibiting IOP at all. Inconclusiveness is equivalent to a reduction of the informational contribution of the output. A totally inconclusive output that (explicitly or implicitly) permits all analyses contains no information at all.

In summary there is a trade-off between timeliness, monotonicity, decisiveness and accuracy. The first three properties can be exchanged against accuracy which in the ideal case converges to the accuracy of non-incremental processing. Whenever the decision on the current token depends on a not yet observed token, four options are available. Either the decision can be delayed, a range of possible variants is provided (inconclusiveness), or a possibly erroneous decision is either accepted as unavoidable (loss of accuracy) or perhaps can be corrected later on (non-monotonicity).

3.1 General strategies

Based on the trade-offs presented in the last section different strategies are possible. We will call a strategy which takes decisions without considering the right context and accepting a loss of accuracy instead of affecting one of the other three properties best-guess (BG).

A common strategy for disambiguation using delay is lookahead (LA). Here a fixed number of tokens to the right of the current token is taken into account when calculating the output. As processing cannot start before the lookahead window is filled with input, a lookahead of n tokens makes the output lag behind the input by the same amount of tokens. Additionally, this strategy can only resolve ambiguity regarding the near future. Long distance influence cannot be captured, e.g. if a later word in a sentence possibly invalidates the previously more probable interpretation like in a German sub-clause where the verb is placed last. A lookahead size of zero is equivalent to the BG strategy.

A strategy resulting in non-monotonic output is reanalysis (RA). Here previous output is recalculated if new information is available and thus earlier decisions can be changed. Ideally reanalysis does not take as long as the initial processing, as otherwise one of the advantages of incremental processing would disappear. Pruning of pending alternatives or a revision of earlier results are possible techniques. In POS tagging where processing speed is fast compared to speech rate or the processing speed of other modules this is of little concern. Reanalysis can deal with long distance influences and, therefore, is able to provide the optimal analysis of the full sentence, but partial output becomes non-monotonic and thus unreliable to a certain degree.

A general strategy to deal with ambiguity in general consists in providing several possible solutions instead of committing to a single one. In contrast to the two previously mentioned ones, the multiple alternatives (MA) strategy affects not only the partial solutions but also the form of the complete analysis. Hybrid strategies are possible like a MA strategy where the alternatives are pruned by reanalysis later on.

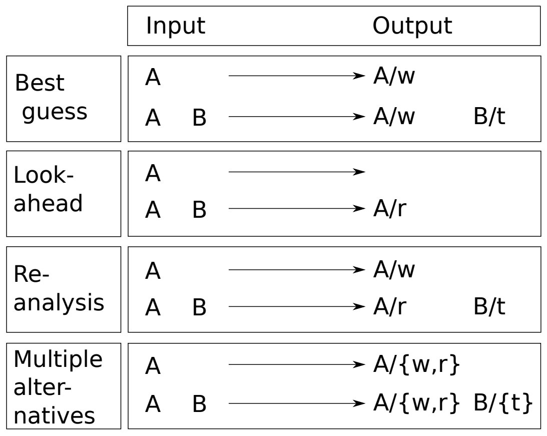

Figure 1: Four strategies to deal with a situation where a decision between a correct tag 'r' and a wrong one 'w' has to be made for the input token 'A' but depends on a subsequent input token 'B'

An illustration of the four strategies is given in Figure 1. Each strategy has certain implications for subsequent processing modules and the behavior of the overall system. First of all, even in the absence of further requirements, delay accumulates in a pipe of modules. Non-monotonicity requires subsequent modules to be able to handle changing input, i.e. be able to perform reanalysis itself. Finally, output consisting of multiple analyses requires a consumer to be able to handle that kind of input, either by selecting one interpretation or by passing the ambiguity on along the processing chain.

3.2 Quantification of delay, non-monotonicity and inconclusiveness

The delay induced by a fixed lookahead can be quantified by the size of lookahead. Dynamic delay could be quantified by the average of the actual delays for each input token or by a recall value measuring the completeness of the output. As we only regard POS tagging strategies with a fixed delay, a predefined lookahead size is used here. The final accuracy is determined for different lookahead sizes.

The degree of non-monotonicity in a RA strategy is not as easy to define or guarantee. One could restrict the possibilities for reanalysis to a window of a certain size or constrain the kind of changes that are allowed. Another possibility consists in restricting reanalysis beforehand but to determine empirically how often a process actually does change its output. This approach was used by Baumann et al. (2009) for incremental speech recognition. Schlangen et al. (2009) applied it to incremental reference resolution where the so called edit overhead was measured. We also use this latter approach by determining the percentage of output tokens which will not be be changed in further processing steps as a stability measure. To be able to decide on an acceptable trade-off between delay, accuracy and non-monotonicity we will plot stability and accuracy for different delays.

To quantify inconclusiveness we use the number of output alternatives to be considered and measure accuracy for different numbers. This requires the number of alternatives to be configurable or to be ranked so that further alternatives can be ignored. Weights, e.g. assigned probabilities, are only used for ranking and are otherwise ignored in this measure.

3.3 Incremental POS tagging algorithms

POS tagging algorithms typically consist of two steps. First, tag probabilities are determined incrementally on a word-by-word basis. Here only local features, e.g., the trigram probability of the current word depending on the tags of the two previous words, or features of adjacent words in support vector machines (SVMs) are used. The second optional step is a global optimization. When only applying the first phase, incremental output of tags is possible in the sense of IOP and the output is monotonic. Approaches using Hidden Markov Models do not use a lookahead (n-grams only regard the tags to the left of the current word), while SVM approaches may use a lookahead by including features of the words to the right.

In the optional second phase, global optimization, either the optimal path is determined, e.g. by the Viterbi algorithm, or an algorithm like the forward backwards algorithm is used to compute probabilities. Theoretically this optimization can also be applied to every prefix analysis. In this case tags to the right can influence tags further left, resulting in a non-monotonic behavior every time a new word changes the optimal path for the previous words. Care has to be taken not to introduce errors by handling prefixes as full sentences. An example for this are end of sentence tags sometimes automatically inserted by a tagger at the end of the current tag sequence. They are likely to influence the best path and thus should not be used in prefixes.

Global optimization can be modified to provide lookahead by not including the rightmost n words of a prefix in the output, but only including them in the calculation of the best path.

Most tagging algorithms work with multiple tag hypotheses for each word internally. A MA strategy can thus be achieved by providing a ranked list of all alternative tags for each word (multi-tagging), while uni-tagging is achieved by suppressing all but the best tag.

As we have seen, tagging algorithms are able to realize all three proposed strategies directly: lookahead by feature lookahead in SVMs, reanalysis by carrying out a global optimization on prefixes and multi-tagging by considering all possible tags for each word. In the next section we will present the POS taggers used in the evaluation and discuss the strategies they are compatible with.

4 Experimental setup

4.1 POS taggers

In order to compare the different strategies of incremental processing in the task of POS tagging we will compare different taggers in different configurations. The taggers used are TnT[2] (Brants, 2000), SVMTool[3] (Giménez and Màrquez, 2004) and HunPos (Halácsy et al., 2007) modified to work incrementally[4].

TnT is a statistical POS tagger implementing the Viterbi algorithm for second order Markov models. As such it does not support incremental processing, but can be made to simulate a non-monotonic incremental mode by tagging successively extended prefixes of the input sequence, thereby providing an incremental interface.[5] To force TnT into a monotonic mode, only the tag for the new word in each prefix is added, the other tags are taken from the tagging results for the previous prefix. This mode, of course, diminishes the utility of the Viterbi algorithm. A lookahead of size n is simulated by introducing a temporal difference between the current token of the input and the output where the former is n tokens ahead of the latter. Therefore, the n rightmost tags are ignored in the

[2]http://www.coli.uni-saarland.de/ thorsten/tnt/

[3]http://www.lsi.upc.es/ nlp/SVMTool/

[4]http://gitorious.org/hunpos

[5]Based on our tests, we believe that TnT doesn't treat such prefixes as whole sentences, which would lead to some errors such as not assigning tags that are improbable for the last word of a sentence.

output (they belong to the lookahead), while the preceding ones are passed on.

SVMTool is a tagger generator based on support vector machines. It can be configured with arbitrary feature sets thereby supporting incremental tagging with various lookahead sizes, including no lookahead at all. Every feature set can be used with the Viterbi algorithm for global optimization. We have applied a feature window consisting of two words to the left of the current word and zero (LA0), one (LA1) or two (LA2, non-incremental) to the right of it.

HunPos is an open source statistical trigram tagger resembling the architecture of TnT. It was specifically modified by us for incremental use by removing the global optimization. In this mode, HunPos keeps a list of possible tag sequences along with their probabilities. For each word new sequences are created by appending each possible tag to all sequences. Given a sequence of tags S_{curr}, the probability of the subsequent sequence assigning tag t_i to the current word is the probability of S_{curr} times the probability of the assignment given S_{curr}. The probability of the assignment of a tag t to the ith word is the sum of all sequences that assign t to the ith word. The algorithm starts with the empty sequence that has a probability of 1. With this modification we obtain an incremental and monotonic tagger that is able to assign probabilities to tags.

As a baseline we used a simplistic tagger that uses unigrams and assigns "normal noun" to each unknown word.

4.2 Data

Evaluation is carried out on sentences from NEGRA Corpus (German), a subset of the WSJ corpus (English), and the Danish and Swedish corpora from the CoNLL-X shared task (Buchholz and Marsi, 2006).

For German and English, the taggers were trained on a subset of 15000 sentences, the evaluation was carried out on the remaining 4058 sentences. The Swedish dataset consists of 11042 sentences for training and 322 for evaluation, the Danish one consists of 5190 and 322 sentences.

To be able to estimate the reliability of the accuracy numbers in the face of the different corpora sizes, we performed a 10-fold cross-validation. Average and standard error values are given for every measure.

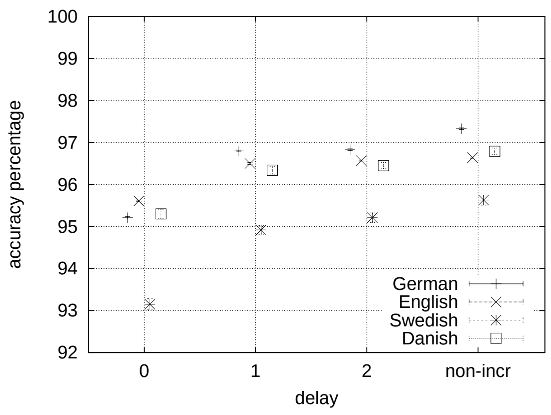

Figure 2: Tagging accuracy of SVMT for different delay sizes

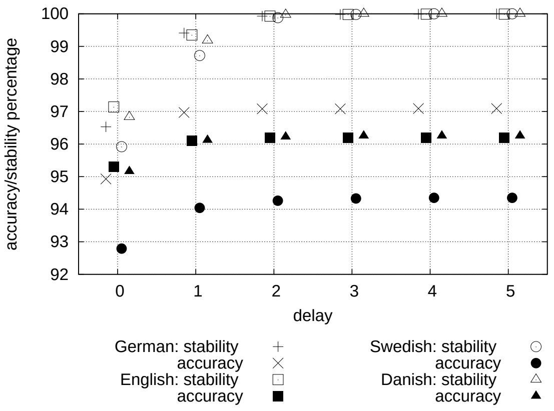

Figure 3: Tagging accuracy and stability of intermediate tags in non-monotonic tagging with TnT. On the x-axis the number of tokens since the word first appeared is given.

5 Results and discussion

Table 1 lists the tagging accuracy for the different incremental strategies. Compared to non-incremental tagging, BG incremental tagging leads to an absolute reduction of tagging accuracy by between 0.69% (HunPos, English) and 2.48% (SVMT, Swedish). Language has a higher influence than the tagger used, with the exception of Swedish, where the 3 evaluated taggers performed very differently. The impact of the incremental mode is generally highest for German and lowest for English.

This loss of accuracy is mitigated by a lookahead of 1 word to around 0.1% and nearly vanishes with a lookahead of 2 words for HunPos and TnT. For SVMT the gap of accuracy between the lookahead of 2 and the non-incremental configu-

Strategy	Tagger	LA	German $\bar{X}(SE)$	English $\bar{X}(SE)$	Swedish $\bar{X}(SE)$	Danish $\bar{X}(SE)$
NonIncr/	TnT		97.14% (0.014)	96.26% (0.004)	94.41% (0.106)	96.45% (0.099)
Reana-	SVMT		**97.33%** (0.020)	**96.64%** (0.014)	**95.63%** (0.121)	**96.79%** (0.079)
lysis	HunPos		97.10% (0.018)	96.32% (0.004)	94.76% (0.135)	96.55% (0.082)
Best	TnT		94.93% (0.024)	95.30% (0.014)	92.79% (0.090)	95.16% (0.106)
Guess	SVMT		**95.21%** (0.031)	95.61% (0.019)	93.15% (0.133)	**95.30%** (0.108)
	HunPos		95.13% (0.029)	**95.63%** (0.006)	**93.64%** (0.095)	**95.30%** (0.106)
Look-	TnT	1	96.97% (0.017)	96.10% (0.003)	94.04% (0.101)	96.12% (0.102)
ahead	TnT	2	97.08% (0.016)	96.19% (0.003)	94.26% (0.112)	96.22% (0.107)
	SVMT	1	96.80% (0.020)	96.50% (0.025)	94.92% (0.105)	96.34% (0.092)
	SVMT	2	96.83% (0.021)	**96.57%** (0.012)	**95.21%** (0.115)	96.45% (0.075)
	HunPos	1	96.99% (0.020)	96.27% (0.004)	94.67% (0.121)	96.40% (0.090)
	HunPos	2	**97.09%** (0.021)	96.32% (0.004)	94.78% (0.136)	**96.56%** (0.082)
Multi-	TnT		**98.63%** (0.018)	98.62% (0.007)	97.99% (0.064)	98.71% (0.029)
tagging	SVMT		98.51% (0.019)	98.60% (0.008)	97.60% (0.092)	98.46% (0.049)
2 tags	HunPos		98.60% (0.020)	**98.70%** (0.004)	**98.20%** (0.075)	**98.74%** (0.033)
Baseline			90.66% (0.027)	90.97% (0.011)	89.65% (0.105)	89.98% (0.184)

Table 1: Tagging accuracy on final results for different combinations of taggers, strategies and languages averages and standard errors from the cross-validation are given

tagger	German	English	Swedish	Danish
TnT	**96.53%**	97.14%	95.14%	**94.93%**
SVMT	93.12%	96.43%	**95.92%**	94.41%
HunPos	96.52%	**97.29%**	95.57%	96.11%

Table 2: Stability numbers of non-monotonic tagging, i.e., the percentage of tags that did not get changed in later output increments

ration is still about 0.5%. This can be explained by the fact that the incremental SVMT configurations used here do not include the global Viterbi optimization. Only feature lookahead is used for SVMT, while for the other two taggers the global optimization was used to implement the lookahead strategy (c.f. Section 4.1).

Alternatively non-incremental output accuracy can be preserved, if between 2.7% (English, HunPos) and 6.9% (German, SVMT) of the tags can be changed within 2 words after they have first been assigned. As can be seen in Figure 3, changes are marginal after that, which is not surprising given the fact that no parser uses features with a distance of more than 2 words. While the delay is the same, reanalysis provides an advantage over lookahead because it makes a tag available immediately with only a minor loss of accuracy. Of course, the consumer of the output needs to be able to process non-monotonic output.

If the top-most two tags from incremental multi-tagging output are considered, the likelihood of the correct one being among them is higher than the accuracy in non-incremental single best mode for all considered languages, as can be seen in Table 1 and in Figure 4. Of course, multi-tagging can also be applied to the non-incremental case, where it produced a slightly better performance than in incremental multi-tagging, e.g., 0.3% for TnT in German (c.f. Figure 4).

In Table 3 the errors contributing most to the accuracy drop in best-guess incremental tagging are shown for English and German. In German a major source of the errors are determiner pronoun confusions and in English a third of all the errors (rows 1, 4 and 7 in Table 3) are wrongly assigned preposition (IN) tags. Ambiguities like these can usually be resolved given the next word and thus explain the big improvement of accuracy between a lookahead of zero and one. A noteworthy observation is the absence of confusions between nouns and proper nouns. They rank among the most common tagging errors in many languages (e.g., 18% in German with TnT for NN $\leftrightarrow$ NE

TnT German		TnT English	
PTKVZ tagged as APPR	11.8%	WDT tagged as IN	19.3%
ART tagged as PRELS	10.6%	JJ tagged as NN	12.8%
ART tagged as PDS	6.2%	VBN tagged as VBD	10.0%
APPR tagged as PTKZU	6.1%	RB tagged as IN	9.0%
VVINF tagged as VVFIN	5.3%	JJ tagged as NNP	6.3%
PRELS tagged as ART	4.9%	NNP tagged as JJ	4.6%
VVFIN tagged as VVINF	4.6%	DT tagged as IN	4.6%
PDS tagged as ART	4.0%	NN tagged as JJ	4.2%
ADJA tagged as NN	3.6%	RBR tagged as JJR	4.2%
NN tagged as ADJA	3.0%	RB tagged as JJ	4.0%
...			
VMFIN tagged as VMINF	−0.7%	DT tagged as WDT'	−1.8%
NE tagged as FM	−0.8%	VBP tagged as VB'	−2.1%
ADV tagged as PIAT	−0.8%	IN tagged as RB',	−2.9%

Table 3: Errors types contributing to the increased error rate of best-guess incremental tagging compared to non-incremental tagging, exemplary for TnT for German and English. Listed are the 10 error types which contribute most to the loss of accuracy and 3 others which even let to an improvement, ranked according to their relative share of the overall error increase.

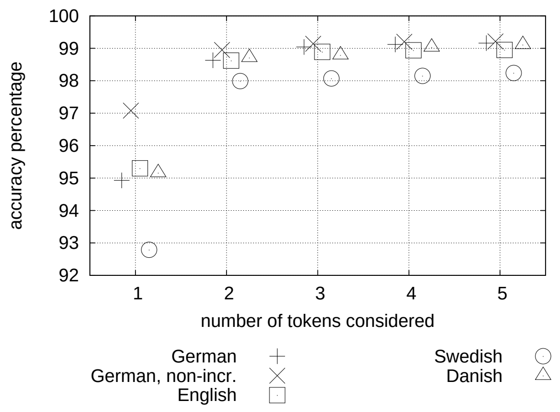

Figure 4: Tagging accuracy of incremental and non-incremental multi-tagging with TnT for different numbers of tags considered

confusion) but have little effect on further processing, as noun and proper noun behave syntactically very similar. This indicates that a BG incremental tagger produces additional errors which are rather severe and likely to have a high negative impact on performance of a consumer component (like a parser).

6 Conclusions and outlook

In this paper we have described the impact of incremental processing modes on the task of POS tagging and how accuracy can be traded against other output parameters like timeliness, monotonicity and decisiveness. A summary of possible trade-offs is given in Figure 5.

It depends on the application which of these trade-offs is acceptable. For non-monotonic and inconclusive output, a consumer is needed that is able to handle such output. Even a slight delay might be unacceptable in applications where an immediate analysis of the most recent input element is needed at all times. Moreover, delay accumulates for all modules in a processing pipeline.

The cost of incremental tagging are one of

- an accuracy drop between 0.7% and 2.5% depending on the language.
- a delay of 2 words (a delay of 1 already considerably reduces the accuracy drop to ca. 0.1%)
- a 2.7%−6.9% chance that the output will be changed later on
- or an ambiguity factor of 2, i.e the two best tags given by the tagger need to be considered.

Figure 5: A summary of the possible trade-offs in incremental POS tagging

The three identified parameters are not specific for POS tagger output but can also be applied to other NLP tasks. Related work has been done for speech recognition (Baumann et al., 2009) and reference resolution (Schlangen et al., 2009). Both papers consider non-monotonic systems and focus on the trade-off between delay and edit overhead.

To our knowledge no comparable investigation has been carried out for syntactic parsing. Besides studying the above mentioned trade-off for the parser itself it would also be interesting to measure the impact of different kinds of incremental POS tagging on the performance of the parser. Dependency parsing lends itself particularly well to such an investigation, because by assigning attachments to input words it shares crucial similarities to POS-tagging. Parsing differs from tagging however, since it considers relations instead of atomic labels. Therefore, the output of dependency arcs needs to be either delayed at least until both ends of the dependency are known or otherwise underspecified dependency arcs need to be produced. The former strategy is applied, e.g., in MaltParser (Nivre, 2004) where words are kept on the stack until a possible attachment becomes available, adding a dynamic delay in addition to the one already caused by lookahead. The incremental variant of the WCDG parser (Foth, 2006) used in Menzel (2009) instead applies the latter approach and provides a placeholder for future input words so that partially specified dependencies can be generated. MaltParser classifies as a monotonic but delayed incremental parser, while output of WCDG is timely but non-monotonic. To deal with and compare dynamic delay and underspecified dependencies the evaluation methods presented here will have to be adapted.

Acknowledgement

This research was funded by the Deutsche Forschungsgemeinschaft (DFG) as part of the International Graduate Research Group CINACS.

References

Timo Baumann, Michaela Atterer, and David Schlangen. 2009. Assessing and improving the performance of speech recognition for incremental systems. In *NAACL '09: Proceedings of Human Language Technologies*, pages 380–388, Morristown, NJ, USA. Association for Computational Linguistics.

Thorsten Brants. 2000. Tnt - a statistical part-of-speech tagger. In *ANLC 00 Proceedings of the sixth conference on Applied natural language processing*.

Sabine Buchholz and Erwin Marsi. 2006. Conll-x shared task on multilingual dependency parsing. In *Proceedings of CoNLL-X*.

Kilian A. Foth. 2006. *Hybrid Methods of Natural Language Analysis*. Ph.D. thesis, Universität Hamburg, Fachbereich Informatik.

Jesús Giménez and Lluís Màrquez. 2004. Svmtool: A general pos tagger generator based on support vector machines. In *Proceedings of the 4th LREC*.

Péter Halácsy, András Kornai, and Csaba Oravecz. 2007. Hunpos - an open source trigram tagger. In *ACL*.

Anne Kilger and Wolfgang Finkler. 1995. Incremental generation for real-time applications. Technical report, Deutsches Forschungzentrum für Künstliche Intelligenz GmbH (DFKI).

Wolfgang Menzel, 2009. *Recent Advances in Natural Language Processing V*, chapter Towards radically incremental parsing of natural language, pages 41–56. Number 309 in Current Issues in Linguistic Theory. John Benjamin's Publisher.

Joakim Nivre. 2004. Incrementality in deterministic dependency parsing. In *Incremental Parsing: Bringing Engineering and Cognition Together, Workshop at ACL-2004, Barcelona, Spain*.

David Schlangen, Timo Baumann, and Michaela Atterer. 2009. Incremental reference resolution: the task, metrics for evaluation, and a bayesian filtering model that is sensitive to disfluencies. In *SIGDIAL '09: Proceedings of the SIGDIAL 2009 Conference*, pages 30–37, Morristown, NJ, USA. Association for Computational Linguistics.

Mats Wirén. 1992. *Studies in Incremental Natural-Language Analysis*. Ph.D. thesis, Linköping University, Department of Computer and Information Science, Linköping.

Mats Wirén. 1993. Bounded incremental parsing. In *6th Twente Workshop on Language Technology (TWLT-6)*.