Optimistic Backtracking
A Backtracking Overlay for Deterministic Incremental Parsing

Gisle Ytrestøl
Department of Informatics
University of Oslo
gisley@ifi.uio.no

Abstract

This paper describes a backtracking strategy for an incremental deterministic transition-based parser for HPSG. The method could theoretically be implemented on any other transition-based parser with some adjustments. In this paper, the algorithm is evaluated on CuteForce, an efficient deterministic shift-reduce HPSG parser. The backtracking strategy may serve to improve existing parsers, or to assess if a deterministic parser would benefit from backtracking as a strategy to improve parsing.

1 Introduction

Incremental deterministic parsing has received increased awareness over the last decade. Processing linguistic data linearly is attractive both from a computational and a cognitive standpoint. While there is a rich research tradition in statistical parsing, the predominant approach derives from chart parsing and is inherently non-deterministic.

A deterministic algorithm will incrementally expand a syntactic/semantic derivation as it reads the input sentence one word/token at the time. There are a number of attractive features to this approach. The time-complexity will be linear when the algorithm is deterministic, i.e. it does not allow for later changes to the partial derivation, only extensions to it. For a number of applications, e.g. speech recognition, the ability to process input on the fly per word, and not per sentence, can also be vital. However, there are inherent challenges to an incremental parsing algorithm. Garden paths are the canonical example of sentences that are typically misinterpret due to an early incorrect grammatical assumption.

(1) The horse raced past the barn fell.

The ability to reevaluate an earlier grammatical assumption is disallowed by a deterministic parser. *Optimistic Backtracking* is an method designed to locate the incorrect parser decision in an earlier stage if the parser reaches an illegal state, i.e. a state in which a valid parse derivation cannot be retrieved. The *Optimistic Backtracking* method will try to locate the first incorrect parsing decision made by the parser, and replace this decision with the correct transition, and resume parsing from this state.

2 Related Work

Incremental deterministic classifier-based parsing algorithms have been studied in dependency parsing (Nivre and Scholz, 2004; Yamada and Matsumoto, 2003) and CFG parsing (Sagae and Lavie, 2005). Johansson and Nugues (2006) describe a non-deterministic implementation to the dependency parser outlined by Nivre and Scholz (2004), where they apply an *n*-best beam search strategy.

For a highly constrained unification-based formalism like HPSG, a deterministic parsing strategy could frequently lead to parse failures. Ninomiya et al. (2009) suggest an algorithm for deterministic shift-reduce parsing in HPSG. They outline two backtracking strategies for HPSG parsing. Their approach allows the parser to enter an old state if parsing fails or ends with non-sentential success, based on the minimal distance between the best candidate and the second best candidate in the sequence of transitions leading up to the current stage. Further constraints may be added, i.e. restricting the number of states the parser may backtrack. This algorithm is expanded by using a beam-thresholding best-first search algorithm, where each state in the parse has a state probability defined by the product of the probabilities of the selecting actions that has been taken to reach the state.

3 CuteForce

Optimistic Backtracking is in this paper used to evaluate CuteForce, an incremental deterministic HPSG parser currently in development. Similar to MaltParser (Nivre et al., 2007), it employs a classifier-based *oracle* to guide the shift-reduce parser that incrementally builds a syntactic/semantic HPSG derivation defined by LinGO English Resource Grammar (ERG) (Flickinger, 2000).

Parser Layout CuteForce has a more complex transition system than MaltParser in order to facilitate HPSG parsing. The sentence input buffer β is a list of tuples with token, part-of-speech tags and HPSG lexical types (i.e. supertags (Bangalore and Joshi, 1999)).

Given a set of ERG rules R and a sentence buffer β, a parser configuration is a tuple $c = (\alpha, \beta, \iota, \pi)$ where:

- α is a stack of “active” edges[1]
- β is a list of tuples of word forms W, part of speech tags POS and lexical types LT derived from a sentence $x = ((W_1, POS_1, LT_1), ...(W_n, POS_n, LT_n))$.
- ι is the current input position in β
- π is a stack of passive edges instantiating a ERG rule

The stack of passive edges π makes up the full HPSG representation of the input string if the string is accepted.

[1]An “active” edges in our sense is a hypothesis of an application of a binary rule where the left daughter is known (an element of π), and the specific binary ERG rule and the right daughter is yet to be found.

Transition System The shift-reduce parser has four different transitions, two of which are parameterized with a unary or binary ERG rule, which are added to the passive edges, hence building the HPSG structure. The four transitions are:

- ACTIVE – (adds an active edge to stack α, and increments ι)
- UNIT(R^1) – (adds unary passive edge to π instantiating unary ERG rule (R^1))
- PASSIVE(R^2) – (pops α and adds binary passive edge to π instantiating binary ERG rule (R^2))
- ACCEPT – (terminates the parse of the sentence. π represents the HPSG derivation of the sentence)

Derivation Example Figure 1 is a derivation example from Redwoods Treebank (Oepen et al., 2002). We note that the tree derivation consists of unary and binay productions, corresponding to the UNIT(R^1) and PASSIVE(R^2) parser transitions. Further, the pre-terminal lexical types have a *_le* suffix, and are provided together with the terminal word form in the input buffer for the parser.

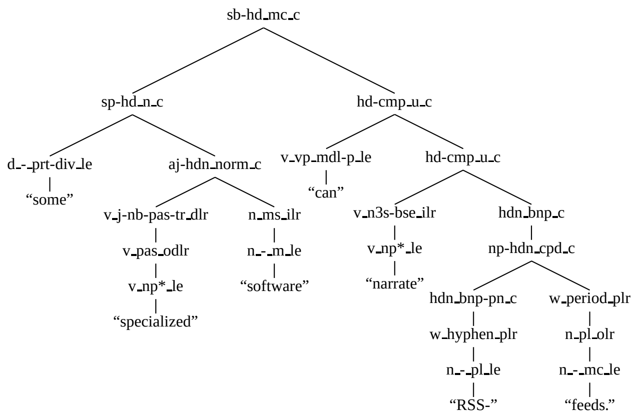

Figure 1: HPSG derivation from Redwoods Treebank.

Parsing Configuration Mode CuteForce can operate in three different oracle configurations: HPSG Unification mode, CFG approximation mode and unrestricted mode.

In HPSG Unification mode, the parser validates that no oracle decisions lead to an invalid HPSG derivation. All UNIT and PASSIVE transitions are an implicit unification. For each parsing stage, the parsing oracle returns a ranked list of transitions. The highest-ranked transition not violating a unification constraint will be executed. If no transition yields a valid unification, parsing fails for the given sentence.

In CFG mode, a naive CFG approximation of the ERG is employed to guide the oracle. The CFG approximation consists of CFG rules harvested from the treebanks used in training the parser – for this purpose we have used existing Redwoods treebanks used in training, and augmented with derivations from WikiWoods, in total 300,000 sentences. Each ERG rule instantiation, using the identifiers shown in Figure 1 as non-terminal symbols, will be treated as a CFG rule, and each parser action will be validated against the set of CFG rules. If the parser action yields a CFG projection not found among the valid CFG rules in the CFG approximation, the CFG filter will block this transition. If the parser arrives at a state where the CFG filter blocks all further transitions, parsing fails.

In unrestricted mode, the oracle chooses the highest scoring transition without any further restrictions imposed. In this setting, the parser typically reaches close to 100 % coverage – the only sentences not covered in this setting are instances where the parser enters an infinite unit production loop. Hence, we will only evaluate the parser in CFG and Unification mode in this paper.

4 Optimistic Backtracking

Optimistic Backtracking can be added as an overlay to a transition-based parser in order to evaluate the parser in non-deterministic mode. The overlay has a linear time-complexity. This backtracking method is, to the best of our knowledge, the only method that applies ranking rather than some probability-based algorithm for backtracking. This aspect is critical for classification-based parsing oracles that do not yield a probability score in the ranking of candidate transitions.

Treating backtracking as a ranking problem has several attractive features. It may combine global and local syntactic and semantic information related to each candidate transition, contrary to a probabilistic approach that only employs the local transition probability. Utilizing global information also seems more sound from a human point of view. Consider sentence (1), it's first when the second verb (*fell*) is encountered that we would re-evaluate our original assumption, namely that *raced* may not be the head verb of the sentence. That *fell* indeed is a verb is surely relevant information for reconsidering *raced* as the head of a relative clause.

When the parser halts, the backtracker will rank each transition produced up until the point of failure according to which transition is most likely to be the first incorrect transition. When the best scoring transition is located, the parser will backtrack to this position, and replace this transition with the parsing oracle's second-best scoring transition for this current parsing state. If the parser later comes to another halt, only the transitions occurring after the first backtrack will be subject to change. Hence, the backtracker will always assume that its last backtrack was correct (thus being *Optimistic*). Having allowed the parser to backtrack unrestrictedly, we could theoretically have reached close to 100 % coverage, but the insights of parsing incrementally would have become less pronounced.

The search space for the backtracker is $n * m$ where n is the number of candidate transitions, and m is the total number of parser transitions. In *Optimistic Backtracking* we disregard the m dimension altogether by always choosing the second-best transition candidate ranked by the parsing oracle, assuming that the second-ranked transition in the given state actually was the correct transition. Hence we reduce the search-space to the n-dimension. In this paper, using CuteForce as HPSG parser, this assumption holds in about 80-90 % of the backtracks in CFG mode, in HPSG Unification mode this number is somewhat lower.

4.1 Baseline

As a baseline for identifying the incorrect transition, we use a strategy inspired by Ninomiya et al. (2009), namely to pick the candidate transition with the minimal probability difference between the best and the second best transition candidate. However, since we do not have true probability, a pseudo-probability is computed by taking the dot product of the feature vector and weight-vector for each best-scoring (P) and second-best scoring (P2) candidate transition, and use the proportion of the second-best score over the joint probability of the best and second-best scoring transition: $\frac{P2}{P+P2}$

In our development test set of 1794 sentences, we ran the parser in CFG and HPSG unification mode in deterministic and non-deterministic mode. The baseline results are found in Table 1 (CFG-BL) and Table 2 (UNI-BL). In CFG mode (Table 1), we obtain a 51.2 % reduction in parsing failure. In unification mode (Table 2) the parser is much more likely to fail, as the parse derivations are guaranteed to be a valid HPSG derivation. Baseline backtracking yields a mere 10 % reduction in parsing failures.

4.2 Feature Model

Each candidate transition is mapped to a feature vector that provides information about the transition. The task for the ranker is to identify the first incorrect transition in the sequence of transitions. The feature model used by the ranker employs features that can roughly be divided in three. First, the transition-specific features provide information on the nature of the candidate transition and surrounding transitions. Here we also have features related to the pseudo-probability of the transition (provided by the parsing oracle), and the oracle score distance between the best-scoring and second-best scoring transition for each given state. Secondly we have features related to the last token that was processed by the parser before it reached an invalid state, and the information on the incomplete HPSG derivation that was built at that state. These features are used in combination with local transition-specific features. Third, we have features concerning the preliminary HPSG derivation in the actual state of the transition.

Feature Types The list of transitions T = t_0, t_1, ... t_n comprises the candidate transitions that are subject to backtracking upon parsing failure. The feature types used by the backtracker includes:

- the pseudo-probability of the best scoring (P) and second best scoring (P2) transition
- the transition category of the current transition
- the probability proportion of the second best scoring transition over the joint probability $\frac{P2}{P+P2}$
- the transition number in the list of applicable candidates, and the number of remaining transitions, relative to the list of candidates
- the last lexical tag and part-of-speech tag that were processed before parsing failure
- the head category of the HPSG derivation and the left daughter unification candidate for the HPSG derivation in the current position
- the lexical tag relative to the current position in the buffer

The backtracker is trained as a linear SVM using SVM^{rank} (Joachims, 2006). Totally, the feature vector maps 24 features for each transition, including several combinations of the feature types above.

5 Evaluation

In this paper we trained CuteForce with data from Redwoods Treebank, augmented with derivations from WikiWoods (Flickinger et al., 2010). The test set contains a random sample of 1794 sentences from the Redwoods Treebank (which was excluded from the training data), with an average length of 14 tokens. Training data for the backtracker is extracted by parsing derivations from WikiWoods deterministically, and record transition candidates each time parsing fails, labeling the correct backtracking candidate, backtrack to this point, and resume parsing from this state.

5.1 Results

The first column (CFG-NB and UNI-NB) in Table 1 and 2 indicates the scores when the parser is run in deterministic mode, i.e. without backtracking. The second and third column contain results for baseline and *Optimistic* backtracking. *Coverage* refers to the proportion of sentences that received a parse. *Precision* refers to the backtracker's precision with respect to identifying the incorrect transition among the candidate transitions. $\sim$ *BT Cand* is the average number of candidate transitions the backtracker ranks when trying to predict the incorrect transition, and $\sim$ *BT Cand,1st* is the number of candidates at the initial point-of-failure. *Exact Matches* is the total number of parse derivations which are identical to the gold standard.

For *Ms per Sent* (milliseconds per sentence) it should be said that the code is not optimized, especially with respect to the HPSG unification algorithm[2]. How the figures relate to one another should however give a good indication on how the computational costs vary between the different configurations.

	CFG -NB	CFG -BL	CFG -Opt
Coverage	0.754	0.880	0.899
Precision	N/A	0.175	0.235
∼BT Cand	N/A	26.1	30.6
∼BT Cand,1st	N/A	51.5	51.5
Exact Matches	727	746	742
Ms per Sent	10.7	45.0	72.5

Table 1: Results – CFG mode

	UNI -NB	UNI -BL	UNI -Opt
Coverage	0.574	0.598	0.589
Precision	N/A	0.183	0.206
∼BT Cand	N/A	12.89	20.12
∼BT Cand,1st	N/A	51.6	51.6
Exact Matches	776	777	776
Ms per Sent	1801.4	5519.1	5345.2

Table 2: Results – HPSG unification mode

5.2 CFG approximation

The number of failed sentences is greatly reduced when backtracking is enabled. Using baseline backtracking, the reduction is 51.2 %, whereas *Optimistic* backtracking has a 59.1 % reduction in parse failures. Further, *Optimistic Backtracker* performs substantially better than baseline in identifying incorrect transitions.

The average number of candidate transitions ranged from 26 to 30 for the baseline and *Optimistic* backtracking strategy. It's interesting to observe that even with a success rate of about 1/5 in identifying the incorrect transition, the coverage is still greatly increased. That backtracking manages to recover so many sentences that initially failed, even if it does not manage to identify the incorrect transition, would seem to indicate that even when mistaken, the backtracker is producing a good prediction. On the other hand, the exact match score does not improve the same way as the coverage, this is directly related to the fact that the backtracker still has relatively low precision, as only a perfect prediction would leave the parser capable of deriving an exact match.

The success rate of about 0.23 in picking the incorrect transition in a set of in average 30 candidates indicates that treating the backtracking as a ranking problem is promising. The precision rate in itself is however relatively low, which serves as an indication of the difficulty of this task.

5.3 HPSG Unification

In unification mode the we see no substantive difference between deterministic mode, and baseline and *Optimistic* backtracking, and practically no improvement in the quality of the parses produced. In Table 2 we see that the only striking difference between the figures for the parser in backtracking mode and deterministic mode is the efficiency – the time consumption is increased by approximately a factor of 3.

5.4 Conclusion

The findings in this paper are specific to CuteForce. It is however very likely that the results would be similar for other deterministic HPSG parsers.

In CFG mode, the number of failed parses are more than halved compared to deterministic mode. It is likely that further increase could be obtained by relaxing constraints in the *Optimistic* algorithm.

In Unification mode, we experienced only a slight increase in coverage. By relaxing the *Optimistic* constraints, the time-complexity would go up. Considering how little the parser benefited from backtracking in unification mode with *Optimistic* constraints, it seems implausible that the parser will improve considerably without a heavy relaxation of the constraints in the *Optimistic* algorithm. If doing so, the attractive features of the parser's inherently deterministic nature will be overshadowed by a very large number of backtracks at a heavy computational cost. Hence, it's hard to see that such a semi-deterministic approach could have any advantages over other non-deterministic HPSG parsers neither in computational cost, performance or on a cognitive level.

[2]Specifically, the current unification back-end preforms non-destructive unification, i.e. it does not take advantage of the deterministic nature of CuteForce

Acknowledgements

The author would like to thank Stephan Oepen (University of Oslo) and Joakim Nivre (Uppsala University) for their valued input and inspiring feedback during the writing of this paper, and in the PhD project. Experimentation and engineering was made possible through access to the TITAN high-performance computing facilities at the University of Oslo (UiO), and we are grateful to the Scientific Computation staff at UiO, as well as to the Norwegian Metacenter for Computational Science.

References

Srinivas Bangalore and Aravind K. Joshi. 1999. Supertagging: an approach to almost parsing. *Computational Linguistics*, pages 237–265.

Dan Flickinger, Stephan Oepen, and Gisle Ytrestøl. 2010. Wikiwoods: Syntacto-semantic annotation for english wikipedia. In *Proceedings of the Seventh conference on International Language Resources and Evaluation (LREC'10)*. European Language Resources Association (ELRA).

Dan Flickinger. 2000. On building a more efficient grammar by exploiting types. *Natural Language Engineering*, 6 (1):15 – 28.

Thorsten Joachims. 2006. Training linear SVMs in linear time. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 217–226. ACM.

Richard Johansson and Pierre Nugues. 2006. Investigating multilingual dependency parsing. In *Proceedings of the Tenth Conference on Computational Natural Language Learning*, pages 206–210. Association for Computational Linguistics.

Takashi Ninomiya, Nobuyuki Shimizu, Takuya Matsuzaki, and Hiroshi Nakagawa. 2009. Deterministic shift-reduce parsing for unification-based grammars by using default unification. In *Proceedings of the 12th Conference of the European Chapter of the Association for Computational Linguistics*, pages 603–611. Association for Computational Linguistics.

Joakim Nivre and Mario Scholz. 2004. Deterministic dependency parsing of English text. In *Proceedings of the 20th international conference on Computational Linguistics*. Association for Computational Linguistics.

Joakim Nivre, Johan Hall, Jens Nilsson, Atanas Chanev, Gülsen Eryigit, Sandra Kübler, Svetoslav Marinov, and Erwin Marsi. 2007. Maltparser: A language-independent system for data-driven dependency parsing. *Natural Language Engineering*, 13(2):95–135.

Stephan Oepen, Kristina Toutanova, Stuart Shieber, Chris Manning, Dan Flickinger, and Thorsten Brants. 2002. The LinGO Redwoods treebank. Motivation and preliminary applications. In *Proceedings of the 19th International Conference on Computational Linguistics*.

Kenji Sagae and Alon Lavie. 2005. A classifier-based parser with linear run-time complexity. In *Proceedings of the Ninth International Workshop on Parsing Technology*, pages 125–132. Association for Computational Linguistics.

Hiroyasu Yamada and Yuji Matsumoto. 2003. Statistical dependency analysis with support vector machines. In *Proceedings of the 8th International Workshop on Parsing Technologies*, pages 195–206.