CKY Parsing With Independence Constraints

Joseph Irwin
Graduate School of Information Science
Nara Institute of Science and Technology
Nara, Japan
joseph-i@is.naist.jp

Yuji Matsumoto
Graduate School of Information Science
Nara Institute of Science and Technology
Nara, Japan
matsu@is.naist.jp

Abstract

The CKY algorithm is an important component in many natural language parsers. We propose a novel type of constraint for context-free parsing called independence constraints. Based on the concept of independence between words, we show how these constraints can be used to reduce the work done in the CKY algorithm. We demonstrate a classifier which can be used to identify boundaries between independent words in a sentence using only surface features, and show that it can be used to speed up a CKY parser. We investigate the trade-off between speed and accuracy, and indicate directions for improvement.

1 Introduction

The CKY algorithm is an $\mathrm{O}\big(|G|n^3\big)$ dynamic programming algorithm for finding all of the possible derivations of a sentence in a context-free language. Its complexity depends on both the sentence length n and the size of the grammar $|G|$. Methods for improving parsing accuracy typically increase the size of the grammar (Klein and Manning, 2003; Petrov and Klein, 2007) or even the exponent of n (Eisner and Satta, 1999). More powerful "deep" grammar formalisms multiply the computational complexity even more (Bangalore and Joshi, 1999).

A common technique for speeding up such parsers is coarse-to-fine parsing, where input is first parsed using a much simpler (and thus smaller) grammar, and the content of the chart is then used to constrain the search over the final grammar (Torisawa et al., 2000; Charniak and Johnson, 2005; Petrov and Klein, 2007). Even with a much smaller grammar, the CKY algorithm may be expensive—Roark et al. (2012) report that the initial CKY step in the Berkeley Parser takes half of the total parse time.

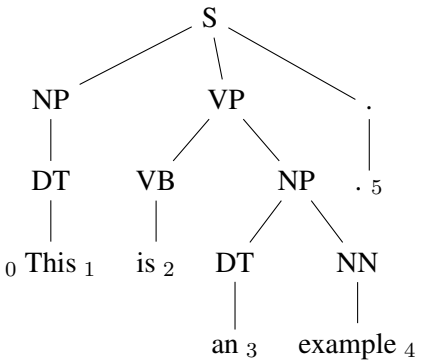

Figure 1: In this tree 'This' and 'is' are independent, while 'is' and 'an' are not.

Other techniques can be used to prune cells in the chart. Roark et al. (2012) use a finite-state model to label words that do/don't begin/end spans, and skip cells that don't satisfy the labels. Bodenstab et al. (2011) directly apply a classifier to each cell to decide how many spans to keep. Both approaches reduce the work done by the parser while preserving accuracy.

We propose a novel type of top-down constraint for a CFG parser that we call independence constraints, described in Section 2. In Section 3 we show how the CKY algorithm can be easily modified to accommodate these constraints, and in Section 4 we describe a classifier which can provide the constraints to a parser. We integrate the constraints into the Stanford Parser CKY implementation and show the results in Section 5.

2 Independence Constraints

We propose a concept we call **independence**. Given a sentence $s = w_1 w_2 \dots w_n$ and a context-free derivation (parse tree) t of s, words w_i and w_{i+1} are **independent** if every node in t that dominates both w_i and w_{i+1} also dominates w_1 and w_n. Furthermore, if w_i and w_{i+1} are independent, then $\forall j, k$ s.t. $j \leq i$ and $k > i$, w_j and w_k are

Proceedings of the 14th International Conference on Parsing Technologies, pages 97–106,
Bilbao, Spain; July 22–24, 2015.

independent. Less formally, if the children of the top node of a parse tree were split into separate subtrees, two words are independent if they would end up in different subtrees.

An example is shown in Figure 1. Here, 'This' and 'is' are independent, as are 'example' and '.'. The independent spans are ('This'), ('is', 'an', 'example'), and ('.'), with boundaries 1 and 4. The independent spans and independent span boundaries can be derived straightforwardly from the definition of independent words: the locations between consecutive words which are independent are the independent span boundaries, and the independent spans are simply the spans in between consecutive boundaries.

3 Modifying The CKY Algorithm With Independence Constraints

Conceptually, if a CKY parser knows the locations of the independent span boundaries for a sentence, it can perform the normal CKY algorithm for each independent span separately, and simply join the spans at the top of the tree to finish the parse, thereby avoiding work which would otherwise be done while still obtaining the desired 1-best parse. Two issues make the task more complicated than this.

The first complication is that if we assume that the independent boundaries will be identified automatically, we must allow for errors. If a location which is not an independent span boundary is given as one, the parser will make an error it would not have otherwise. On the other hand, if a location which is an independent span boundary is not marked as such, the parser may account for this at the cost of not achieving the minimum computation possible. By allowing for this second type of error, the algorithm is made more robust, and allows the independent boundary identification step to prioritize precision over recall to lessen negative impact on the parser's accuracy.

The second issue is caused by the binarization of the context-free grammar used in the CKY algorithm. Because the CKY algorithm requires a binary grammar, any rules in the original grammar that have more than two symbols on the right-hand side must be converted into a sequence of binary rules. The extra rules created in this process are called incomplete[1] rules. The topmost span in particular will usually need to be constructed in several steps, applying multiple incomplete rules before creating a complete span. If the grammar rules are always binarized from the left (right), then only cells on the left (right) edge of the chart can affect the top span; however, the grammar used in our parser is binarized "head-outward" (Klein and Manning, 2003), which means that potentially any cell in the chart can be used to create the top span.

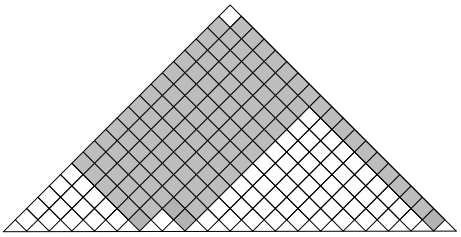

Figure 2: Example of a CKY chart with independence constraints. In the gray cells the modified algorithm will only loop over incomplete rules.

The combination of these two issues means that in order to correctly parse a sentence when an independent span boundary is missing from the input the modified CKY algorithm must process incomplete rules even at positions in the chart that cross a boundary. Thus in the modified algorithm, cells which do not cross an independent boundary are processed normally, and in cells which do cross a boundary the algorithm will avoid looping over complete binary rules.[2] Figure 2 shows an example CKY chart where boundary-crossing cells are colored gray.

3.1 How much work can we expect to save?

The core of the CKY algorithm is shown in Algorithm 1. For our purposes, we can consider the amount of work done in the CKY algorithm to be the number of binary edges visited in the inner loop (lines 4-10). For each cell the algorithm iterates over the binary rules in the grammar, calculating the probability of the left-hand-side at each split point. The number of these binary edges is

[1] E.g., if a rule $A \rightarrow BCD$ becomes $@_{BC} \rightarrow BC$ and $A \rightarrow @_{BC}D$, then the former is *incomplete* and the latter is *complete*.

[2] While boundary-crossing cells depend on non-crossing cells, the reverse is not the case; thus the non-crossing cells can all be processed before the crossing cells, or the cells can be looped over in the regular order, with a check inside the loop. While this may have implications for e.g. parallelization, we do not explore this idea further here.

```
for 1 ≤ i ≤ n do
    T_{i,i+1} ← {A|A → a ∈ G ∧ w_i = a}
end
for 2 ≤ j ≤ n do
    for 1 ≤ i ≤ n − j + 1 do
        for i < k < i + j do
            T_{i,i+j} ← {A|A → BC ∈
            G ∧ B ∈ T_{i,k} ∧ C ∈ T_{k,i+j}}
        end
    end
end
```

Algorithm 1: The CKY algorithm. $T_{i,j}$ is the cell corresponding to words $w_i \dots w_{j-1}$.

$$|G| \left[\frac{n^3}{6} - \frac{n}{6} \right] \tag{1}$$

The amount of work saved depends on the number and locations of the independent span boundaries, as well as the proportion of complete rules in the grammar, denoted $\frac{|G_{comp}|}{|G|}$. We can consider two idealized scenarios: a) one boundary at $\frac{n}{K}$, and b) $K-1$ boundaries at $\frac{n}{K}, \frac{2n}{K}, \dots, \frac{(K-1)n}{K}$, where K is an integer $1 < K < n$.

For the first case, the ratio of work saved approaches

$$\frac{|G_{comp}|}{|G|} \left[\frac{3}{K} - \frac{3}{K^2} \right] \tag{2}$$

as n grows. This limit converges quickly for $n \geq 10$. If we approximate $|G_{comp}|/|G|$ as 0.5 (for the grammar used by the parser in Section 5, it is $\approx .54$), then for $K = 2, 3, 4, \dots$, the values are $\frac{3}{8}, \frac{1}{3}, \frac{3}{32}, \dots$ Intuitively, for one boundary, the best location is exactly in the center of the sentence, and the upper limit on how much work is saved is about 37%.

For the case of $K-1$ boundaries equally spaced, the ratio is

$$\frac{|G_{comp}|}{|G|} \frac{K^2 - 1}{K^2} \tag{3}$$

The values for $K = 2, 3, 4, \dots$ are $\frac{3}{8}, \frac{4}{9}, \frac{15}{32}, \dots$ Clearly, the smaller pieces a sentence can be divided into the less work the parser will do; however, realistically most sentences will not have a large number of independent spans, and they will not be equal in length. We might take $K = 3$ as best-case estimate, giving us about 44%. Thus we can guess that a parser will be able to save around 35-45% of the work it does in the CKY algorithm loop by using independence constraints.

The derivations of Equations 1-3 are shown in Appendix A.

Local Features	
t_{k-1}	t_k
t_{k-2}, t_{k-1}	t_k, t_{k+1}
$t_{k-3}, t_{k-2}, t_{k-1}$	t_k, t_{k+1}, t_{k+2}
Global Features	
t^l_i	$1 \leq i < k-1$
t^l_i, t^l_{i+1}	$1 \leq i < k-2$
$t^l_i, t^l_{i+1}, t^l_{i+2}$	$1 \leq i < k-3$
t^l_i	$k \leq i < n-1$
t^l_i, t^l_{i+1}	$k \leq i < n-2$
$t^l_i, t^l_{i+1}, t^l_{i+2}$	$k \leq i < n-3$

Table 1: Feature templates. k is the boundary position, t_k is the k th POS tag (level 0), and t^l_i is the i th POS tag in the l-level POS tag sequence.

4 Classifying Independent Span Boundaries

In order to use independence constraints in a parser, we need to be able to identify boundaries between independent words in a sentence using only surface features (words and part-of-speech tags). We created a binary classifier which, given a POS-tagged sentence and a position between two words, decides whether those two words are independent or not. Our classifier currently uses only POS tags as features. We used `opal` (Yoshinaga and Kitsuregawa, 2010), a tool for fast online classification, to train and test the models, training on sentences from Penn Treebank section 02-21 and testing on section 22. We set opal to use the passive-aggressive perceptron update, and output probabilities in order to use a threshold to trade off precision and recall.

4.1 Features

We use only part-of-speech tags to create features for the classifier (adding lexical or other features is left to future work). The property of independence between two words is inherently global, as it can be affected by structure arbitrarily far away. Thus we have both local and global features. The global features are furthermore distinguished by **POS level**, explained in detail in the next section. The specific feature templates are shown in Table 1.

Features	#feats	Acc	Prec	Rec	F_1	$F_{0.5}$	TP	FP	FN	TN
p	37001	93.71	80.73	70.49	75.27	78.45	3679	878	1540	32320
P_0	33167	87.16	51.69	83.98	63.99	55.99	4383	4097	836	29101
p,P_0	70168	95.21	87.38	75.65	81.09	84.75	3948	570	1271	32628
p,P_1	37055	94.81	78.38	85.38	81.73	79.69	4456	1229	763	31969
p,P_2	39336	95.34	84.25	80.76	82.47	83.53	4215	788	1004	32410
p,P_3	46861	95.04	89.47	71.95	79.76	85.31	3755	442	1464	32756
p,P_0,P_1	70222	**95.48**	**88.95**	76.16	82.06	**86.06**	3975	494	1244	32704
p,P_0,P_2	72503	95.09	88.28	73.60	80.27	84.89	3841	510	1378	32688
p,P_0,P_3	80028	94.84	88.81	70.99	78.91	84.56	3705	467	1514	32731
p,P_1,P_2	39390	95.27	80.99	85.21	**83.04**	81.80	4447	1044	772	32154
p,P_1,P_3*	41553	**95.44**	**89.05**	75.74	81.86	**86.03**	3953	486	1266	32712
p,P_0,P_1,P_2,P_3	82417	95.35	86.89	77.49	81.92	84.83	4044	610	1175	32588

Table 2: Results of classifier using different combinations of features. (*Final feature configuration.)

Lvl0	Lvl1	Lvl2	Lvl3	Lvl0	Lvl1	Lvl2	Lvl3
NN	N	N	N	CD	X	X	#
NNP	N	N	N	-LRB-	X	X	B
NNPS	N	N	N	-RRB-	X	X	B
NNS	N	N	N	DT	X	X	D
PRP	N	N	N	PDT	X	X	D
VB	V	V	V	PRP$	X	X	D
VBD	V	V	V	WP$	X	X	D
VBG	V	V	V	JJ	X	X	J
VBN	V	V	V	JJR	X	X	J
VBP	V	V	V	JJS	X	X	J
VBZ	V	V	V	-RQ-	X	X	Q
,	X	,	,	-LQ-	X	X	Q
.	X	.	.	RB	X	X	R
:	X	:	:	RBR	X	X	R
CC	X	C	C	RBS	X	X	R
IN	X	I	I	EX	X	X	X
RP	X	I	I	FW	X	X	X
TO	X	T	T	LS	X	X	X
WDT	X	W	W	MD	X	X	X
WP	X	W	W	POS	X	X	X
WRB	X	W	W	SYM	X	X	X
#	X	X	#	UH	X	X	X
$	X	X	#				

Table 3: For each POS level, the original tag is replaced with the corresponding value.

4.2 POS Level

In previous unpublished work on a similar task, we found that heuristically transforming the POS tag sequence to create additional features can be beneficial. We refer to these transformations as **POS levels**. In this classifier we implemented three levels, in addition to the original POS tags as level 0.

We show all levels in Table 3. Each level specifies a value by which each level 0 tag is replaced during the transformation. The motivation behind each transformation is roughly as follows: level 1 is meant to capture clause nuclei; level 2 is further intended to show boundaries between clauses; and level 3 expands almost all the way back to the original tags, but with some distinctions erased, mostly to reduce the number of features.

4.3 Which Features Are Useful?

In order to find the best configuration of features for the classifier, and to evaluate the proposed POS levels, we tested the classifier using several different combinations. Selected results are shown in Table 2. In the "Features" column, p denotes the local features, and P_l denotes the global features from POS level l.

There are several things worth noting in these results. First, neither local nor global features are sufficient alone; it appears that local features promote precision, while global features promote recall. Second, examining the cases where global features are limited to a single POS level, it is apparent that each POS level has a different effect on precision and recall, thus confirming that the classifier is able to extract different signals from the different POS levels, as intended. Finally, combining all POS levels together actually reduces accuracy, possibly because the features are highly correlated (although see the discussion of the kernel classifier).

4.4 Results

To avoid degrading the accuracy of the parser as much as possible, we selected the feature configuration based on $F_{0.5}$ score, a measure which favors precision over recall. We chose p, P_1, P_3 over p, P_0, P_1 because the former had a slight edge in precision and fewer features.

More detailed results are shown in Table 4. We used a threshold on the score output by the classifier to reverse some of the classifier's decisions in a post-process step. Although it doesn't improve on the classifier in accuracy, the `precision` thresh-

Features	Threshold	Acc	Prec	Rec	F_1	$F_{0.5}$	TP	FP	FN	TN
p,P_1,P_3	default	95.44	89.05	75.74	81.86	86.03	3953	486	1266	32712
p,P_1,P_3	precision	94.99	91.65	69.44	79.01	86.14	3624	330	1595	32868
p,P_1,P_3	max precision	92.10	95.80	43.74	60.06	77.38	2283	100	2936	33098
p,P_1,P_3	recall	94.28	73.82	89.65	80.97	76.53	4679	1659	540	31539

Table 4: Results of linear classifier using different score thresholds.

Features	Threshold	Acc	Prec	Rec	F_1	$F_{0.5}$	TP	FP	FN	TN
p,P_0,P_1,P_2,P_3	default	97.47	92.17	88.91	90.51	91.50	4640	394	579	32804
p,P_0,P_1,P_2,P_3	precision	97.27	92.95	86.43	89.58	91.57	4511	342	708	32856
p,P_0,P_1,P_2,P_3	max precision	96.57	94.22	79.63	86.31	90.89	4156	255	1063	32943
p,P_0,P_1,P_2,P_3	recall	97.15	88.16	91.32	89.71	88.78	4766	640	453	32558

Table 5: Results of polynomial classifier using different score thresholds.

old did slightly improve in $F_{0.5}$.

4.5 Efficiency of the Classifier

The efficiency of the classifier is as important as the accuracy—it doesn't matter how much time is saved during parsing if it takes even longer to run the classifier. `opal` takes less than half a second to run on the instances from section 22; however, the instances are created by a Python script, which is not very optimized. This script takes about 100 seconds to run on the machine described in Section 5.1. While this time is already less than the time saved in the parser (see Section 5.2), it could be significantly reduced by re-implementing in Java or even C++. Thus the potential gains offered by this approach are not just theoretical.

4.6 Polynomial Kernel

For comparison with the linear classifier, we trained another classifier using a polynomial kernel (with degree 3) with all the features. The results are shown in Table 5. The polynomial kernel improves over the linear classifier in accuracy by 2%, in precision by 3 points, and in recall by just over 13 points. This suggests that there is a large potential for improving the linear classifier by adding conjunctive features. Alternatively, there are methods for effectively linearizing a kernel-based classifier, e.g. (Kudo and Matsumoto, 2003; Isozaki and Kazawa, 2002). Currently, the polynomial classifier takes over 2 hours to run on section 22 (training the model took almost 4 days).

5 Parsing With Independence Constraints

In order to demonstrate use of the independent constraints in a parser, we modified the CKY parser included in the Stanford Parser distribution to accept independent span boundaries as constraints and to use the modified CKY algorithm described above. Our modifications are:

- after reading in the grammar, index the incomplete binary rules
- read in the file containing the boundaries output by the classifier from the previous section
- for each CKY cell, if the cell spans a boundary then loop over just the incomplete binary rules
- if at the end of the CKY loop a parse was not successful, then loop again over just the cells which span a boundary and process all of the binary rules
- output the total number of times entering the inner loop as well as the number of times the parser failed

5.1 Experimental Setup

We used the modified Stanford Parser described above, with an unlexicalized grammar[3] extracted from the WSJ sections 02-21, and evaluated its performance on section 22 using output from the classifier as constraints. For the baseline, the

[3]The grammar was extracted using the Stanford Parser with command-line options `-acl03pcfg -noRebinarization -compactGrammar 1`

Parser	Time (s)	Speedup	# Binary Edges	F_1	Parse Failures
baseline	1558	-	1.75×10^{10} (100%)	85.85	0
linear	1283 (+100)	1.21× (1.12×)	1.08×10^{10} (62%)	83.71 (-2.14)	15
poly	1106 (+2h)	1.41× (.19×)	9.74×10^{09} (56%)	84.85 (-1.00)	6
oracle	1016	1.53×	8.47×10^{09} (48%)	86.71 (+0.86)	4

Table 6: Results of parsing with independence constraints. Results for both linear and polynomial classifiers are shown, as well as for the gold independent span boundaries. The times in parentheses are the classifier run times.

parser was given null constraints. The accuracies and times shown are those reported by the Stanford Parser.

All experiments were run on a DELL Precision 690, with 8 cores and 32G of RAM. Unless otherwise noted multiple processes were run in parallel, and times reported were not averaged over multiple runs. Since we saw significant variation of up to 10%, the times should be taken with a grain of salt. The computation done in the CKY algorithm is measured in the number of binary edges visited in the inner loop. A binary edge is a tuple of a span (begin & end), a binary rule $A \rightarrow BC$, and a split point (the position where B and C meet).

5.2 Results

The results of running the parser on section 22 using the linear classifier from Section 4.4 are shown in Table 6. The table shows the total time taken, the total times entering the inner loop, the F_1 and difference from the baseline, and the number of times the parse failed using the constraints. The parser with independence constraints saves 38% of the computation inside the CKY loop over the baseline, corresponding to about 20% reduction in total parse time (12% if the running time of the classifier is included), at the cost of a 2-point drop in F-score. Detailed results of further experiments with various thresholds on sentence length and classifier score are shown in Appendix B.

5.3 Polynomial Kernel

A difference of 2 F_1 score is not small, but on the other hand it is about by how much the unlexicalized Stanford Parser trails the Collins parser, for example. However, as shown above in Section 4.6, there is room to improve the linear classifier through conjunctive features. As an indication of an upper bound of the achievable performance, we tried using the output of the kernel classifier in the parser as above, while acknowledging that at present the time needed to produce the classifier output dwarfs the time needed to actually parse the test data.

Parser	Time (s)	Speedup	F_1
baseline	1538		85.54
linear	1106 (+100)	1.39× (1.28×)	83.55 (-1.99)
poly	1040	1.48×	84.57 (-0.97)

Table 7: Results of parsing WSJ section 23.

The results of running the parser on section 22 with the polynomial classifier output are shown with the previous results in Table 6. With the more accurate classifier, the parser is able to reduce the necessary computation even further, by 44%, while losing less accuracy.

5.4 Gold Independent Span Boundaries

For another comparison, we tested the parser using the gold independent span boundaries. The results for section 22 are shown in Table 6. The number of binary edges visited is cut in half, and parsing accuracy is improved by almost 1 point. It is interesting to note that the parser was unable to parse 4 sentences with the gold constraints (the grammar only allowed a parse that violated the gold boundaries).

5.5 WSJ Section 23

To compare with previous work on parsing using the Penn Treebank, we show the time and accuracy for parsing section 23, using both linear and kernel classifier output along with the baseline parser in Table 7. The times reported are the average of three runs each. Because there was significant variation in parse time when multiple processes were run in parallel, for these results only one process was run at a time. The results parallel those shown on the development data.

As a point of comparison, Roark et al. (2012) reported speedups of 1.6-2x with no loss of accuracy. These results are not directly comparable

due to differences in parser (their parsers use beam search variants of CKY and coarse-to-fine pruning) and grammar (they used the Berkeley latent variable grammar and a lexicalized grammar).

6 Related Work

There are several strains of research related to adding constraints to the CKY chart. (Roark et al., 2012) describes an approach using finite-state taggers to decide whether each word in a sentence begins or ends a multiword constituent and has a unary span or not. They show that their tagger is able to achieve very high precision, reducing parse time without negatively affecting accuracy.

(Bodenstab et al., 2011) proposes a classifier which directly decides for each cell in the chart how many constituents should be created. Their parser uses beam search with a FOM and a beam for each chart cell.

Like these approaches, our method uses a classifier to avoid doing work in certain chart cells. While not completely orthogonal, we believe our independence constraints are complementary. A single decision by our classifier closes a large swath of cells based on the global structure, while their methods make local decision using local information. The high accuracy of their classifiers shows the necessity of improving our model.

(Yarmohammadi et al., 2014) proposes a concept of 'hedge' parsing, where only spans below a certain length are allowed, and show how this reduces the computation done by the CKY algorithm. Their system does not create spans of length larger than the threshold and thus doesn't follow the original treebank annotation, while our approach is able to return the original gold parse tree, provided that the classifier does not output a false positive. Their approach of segmenting a sentence before parsing is essentially the same as ours, but they segment based on a maximum span length and their classifier is based on a finite-state sequence model.

There is some previous research using clause boundaries to constrain dependency parsers (Ohno et al., 2006; Husain et al., 2011; Kim and Lee, 2004). This is more linguistically motivated than our constraints; indeed, the approaches appear to rely on processing specific to each language. It is difficult to compare with these results directly; however, although only (Ohno et al., 2006) reported parse times, all three papers reported improved accuracy.

7 Conclusions

We have proposed a property of **independence** between words in a sentence, and shown how to use this property to create top-down constraints which can be used to reduce the computation done by the CKY algorithm. We demonstrated two classifiers for identifying boundaries between independent words given a sentence with only surface features, a linear classifier which is fast but less accurate, and a classifier with a polynomial kernel which is much more accurate but very slow. We then showed a significant improvement in speed over a strong baseline CKY parser by using the output of these classifiers to create top-down constraints at the cost of some accuracy.

Although the loss of accuracy when using the linear classifier is currently uncomfortably large, there are several possible avenues for improvement. The performance of the kernel classifier indicates that there is room for improvement by manually adding conjunctive features to the linear classifier or using a method to automatically linearize the model. Features based on words as well as POS tags may also be beneficial. Changing the model itself to, e.g., a sequence model might also help. However, the current approach has several weaknesses which should be addressed by future research.

First, the top-down nature of the independence constraints does not make a natural fit with the bottom-up CKY algorithm. In particular, the presence of incomplete rules in the grammar combined with the bottom-up search means that the parser still ends up doing some computation to create spans which violate the constraints, even though it is prevented from completing such a span.

Second, the pipelined nature of the classifier means that it only has access to POS tags and in particular is not able to make use of information generated as the parser processes lower-level spans. Tighter integration of the classifier into the parser may be beneficial to both.

Third, the current classifier combines instances from different syntactic structures into a single model. It is possible that training multiple models on different types of sentences would result in a better classifier.

References

Srinivas Bangalore and Aravind K Joshi. 1999. Supertagging: An Approach to Almost Parsing. *Computational Linguistics*, 25:237–265.

Nathan Bodenstab, Aaron Dunlop, Keith Hall, and Brian Roark. 2011. Beam-Width Prediction for Efficient Context-Free Parsing. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics*, pages 440–449, Portland, Oregon. Association for Computational Linguistics.

Eugene Charniak and Mark Johnson. 2005. Coarse-to-fine n-best parsing and maxent discriminative reranking. In *ACL 2005*.

Jason Eisner and Giorgio Satta. 1999. Efficient parsing for bilexical context-free grammars and head automaton grammars. In *ACL 1999*.

Samar Husain, Phani Gadde, Joakim Nivre, and Rajeev Sangal. 2011. Clausal parsing helps data-driven dependency parsing: Experiments with hindi. In *IJCNLP 2011*, pages 1279–1287. The Association for Computer Linguistics.

Hideki Isozaki and Hideto Kazawa. 2002. Efficient support vector classifiers for named entity recognition. In *COLING 2002*.

Mi-Young Kim and Jong-Hyeok Lee. 2004. Syntactic analysis of long sentences based on s-clauses. In Keh-Yih Su, Jun'ichi Tsujii, Jong-Hyeok Lee, and Oi Yee Kwong, editors, *IJCNLP 2004*, volume 3248 of *Lecture Notes in Computer Science*, pages 518–526. Springer.

Dan Klein and Christopher D Manning. 2003. Accurate Unlexicalized Parsing. In *Proceedings of the 41st Annual Meeting of the Association for Computational Linguistics*, pages 423–430, Sapporo, Japan, July. Association for Computational Linguistics.

Taku Kudo and Yuji Matsumoto. 2003. Fast methods for kernel-based text analysis. In *ACL 2003*, pages 24–31.

Tomohiro Ohno, Shigeki Matsubara, Hideki Kashioka, Takehiko Maruyama, and Yasuyoshi Inagaki. 2006. Dependency parsing of japanese spoken monologue based on clause boundaries. In *ACL 2006*. The Association for Computer Linguistics.

Slav Petrov and Dan Klein. 2007. Improved inference for unlexicalized parsing. In *Human Language Technology Conference of the North American Chapter of the Association of Computational Linguistics, Proceedings, April 22-27, 2007, Rochester, New York, USA*, pages 404–411.

Brian Roark, Kristy Hollingshead, and Nathan Bodenstab. 2012. Finite-State Chart Constraints for Reduced Complexity Context-Free Parsing Pipelines. *Computational Linguistics*, 38(4):702–753.

Kentaro Torisawa, Kenji Nishida, Yusuke Miyao, and Jun'ichi Tsujii. 2000. An HPSG parser with CFG filtering. *Natural Language Engineering*, 6(1):63–80.

Mahsa Yarmohammadi, Aaron Dunlop, and Brian Roark. 2014. Transforming trees into hedges and parsing with "hedgebank" grammars. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics, ACL 2014, June 22-27, 2014, Baltimore, MD, USA*, pages 797–802. The Association for Computer Linguistics.

Naoki Yoshinaga and Masaru Kitsuregawa. 2010. Kernel Slicing: Scalable Online Training with Conjunctive Features. In Chu-Ren Huang and Dan Jurafsky, editors, *Proceedings of the 23rd International Conference on Computational Linguistics COLING 2010*, pages 1245–1253, Beijing. Tsinghua University Press.

Appendix A. Derivation of equations in section 3.1

The amount of computation done in lines 4-10 of Algorithm 1 can be calculated as follows:

$$\begin{aligned}
&\sum_{j=2}^{n}\sum_{i=1}^{n-j+1}\sum_{k=i+1}^{i+j-1}|G| \\
=&|G|\sum_{j=2}^{n}(n-j+1)(j-1) \\
=&|G|\sum_{i=1}^{n-1}(n-i)(i) \\
=&|G|\sum_{i=1}^{n-1}(ni-i^2) \\
=&|G|(n\sum_{i=1}^{n-1}i-\sum_{i=1}^{n-1}i^2) \\
=&|G|(n\frac{(n-1)n}{2}-(\frac{n^3}{3}-\frac{n^2}{2}+\frac{n}{6})) \\
=&|G|(\frac{1}{2}n^3-\frac{1}{2}n^2-\frac{1}{3}n^3+\frac{1}{2}n^2-\frac{1}{6}n) \\
=&|G|(\frac{1}{6}n^3-\frac{1}{6}n)
\end{aligned}$$

This is the number of binary edges evaluated by the CKY algorithm. Using independence constraints, the algorithm avoids doing any computation for complete edges in spans which violate the constraints. The work saved is thus the number of complete binary edges in the entire chart minus the number of complete edges that are actually processed in cells that satisfy the constraints. For a single independent boundary at $\frac{n}{K}$, we get:

$$\begin{aligned}
&|G_{comp}|[\frac{1}{6}n^3-\frac{1}{6}n] \\
&-|G_{comp}|[\frac{1}{6}(\frac{n}{K})^3-\frac{1}{6}\frac{n}{K}] \\
&-|G_{comp}|[\frac{1}{6}(\frac{(K-1)n}{K})^3-\frac{1}{6}\frac{(K-1)n}{K}] \\
=&|G_{comp}|[\frac{1}{6}n^3-\frac{1}{6}n-\frac{1}{6}\frac{(K-1)^3+1}{K^3}n^3+\frac{1}{6}n] \\
=&|G_{comp}|[\frac{1}{6}\frac{K^3}{K^3}n^3-\frac{1}{6}\frac{K^3-3K^2+3K}{K^3}n^3] \\
=&|G_{comp}|\frac{3K^2-3K}{6K^3}n^3
\end{aligned}$$

The proportion of work saved relative to the original algorithm is then

$$\frac{|G_{comp}|\frac{3K^2-3K}{6K^3}n^3}{|G|(\frac{1}{6}n^3-\frac{1}{6}n)}$$

which depends on n as well as K; however, we can approximate this as the limit as n goes to infinity:

$$\begin{aligned}
&\lim_{n\to\infty}\frac{|G_{comp}|\frac{3K^2-3K}{6K^3}n^3}{|G|(\frac{1}{6}n^3-\frac{1}{6}n)} \\
=&\frac{|G_{comp}|}{|G|}[\frac{3}{K}-\frac{3}{K^2}]
\end{aligned}$$

Similarly, the work saved with K evenly-spaced boundaries is

$$\begin{aligned}
&|G_{comp}|[\frac{1}{6}n^3-\frac{1}{6}n] \\
&-K|G_{comp}|[\frac{1}{6}(\frac{n}{K})^3-\frac{1}{6}\frac{n}{K}] \\
=&|G_{comp}|[\frac{1}{6}n^3-\frac{1}{6}n-\frac{1}{6}\frac{1}{K^2}n^3+\frac{1}{6}\frac{n}{K}] \\
=&|G_{comp}|\frac{1}{6}\frac{K^2-1}{K^2}n^3
\end{aligned}$$

and the proportion of the original work saved is approximately

$$\begin{aligned}
&\lim_{n\to\infty}\frac{|G_{comp}|\frac{1}{6}\frac{K^2-1}{K^2}n^3}{|G|(\frac{1}{6}n^3-\frac{1}{6}n)} \\
=&\frac{|G_{comp}|}{|G|}\frac{K^2-1}{K^2}
\end{aligned}$$

Appendix B. Detailed parse results

We experimented with a post-processing step to adjust the recall and precision of the classifier, as well as adding a threshold on the minimum length of a sentence to apply constraints to in the parser (on the hypothesis that longer sentences are likely to gain a proportionally larger advantage). We show the detailed results from the parser in Table 8, using both the linear and polynomial classifiers. Sentences shorter than `MinSentLen` were parsed without constraints.

The results are largely as expected. Sentences less than 20 words do not affect the results much. The `recall` threshold predictably results in a large loss in classifier precision and thus parsing accuracy. We note the results in boldface: with a high precision threshold, the polynomial classifier is able to reduce the computation in the CKY loop by 42% while losing less than half a point in F_1 score.

Classifier	MinSentLen	Constraints	Time (s)	# Edges	F_1	Parse Failures
-	-	baseline	1558	1.75×10^{10} (100%)	85.85	0
linear	0	default	1283	1.08×10^{10} (62%)	83.71 (-2.14)	15
linear	0	precision	1143	1.13×10^{10} (65%)	84.05 (-1.80)	7
linear	0	max precision	1384	1.42×10^{10} (81%)	85.55 (-0.30)	2
linear	0	recall	1024	7.80×10^{09} (45%)	78.74 (-7.11)	136
linear	20	default	1126	1.12×10^{10} (64%)	84.17 (-1.68)	9
linear	20	precision	1313	1.16×10^{10} (66%)	84.43 (-1.42)	4
linear	20	max precision	1338	1.44×10^{10} (82%)	85.59 (-0.26)	2
linear	20	recall	1121	8.24×10^{09} (47%)	80.38 (-5.47)	103
linear	30	default	1312	1.28×10^{10} (73%)	84.82 (-1.03)	3
linear	30	precision	1279	1.31×10^{10} (75%)	85.01 (-0.84)	1
linear	30	max precision	1485	1.53×10^{10} (87%)	85.63 (-0.22)	1
linear	30	recall	1140	1.02×10^{10} (58%)	82.79 (-3.06)	57
linear	40	default	1476	1.51×10^{10} (86%)	85.56 (-0.29)	1
linear	40	precision	1390	1.52×10^{10} (87%)	85.59 (-0.26)	0
linear	40	max precision	1513	1.65×10^{10} (94%)	85.75 (-0.10)	0
linear	40	recall	1403	1.33×10^{10} (76%)	84.65 (-1.20)	14
poly	0	default	1106	9.74×10^{09} (56%)	84.85 (-1.00)	6
poly	0	precision	1118	9.84×10^{09} (56%)	85.12 (-0.73)	4
poly	0	max precision	1137	**1.02×10^{10} (58%)**	**85.42 (-0.43)**	2
poly	0	recall	1050	9.25×10^{09} (53%)	84.05 (-1.80)	33
poly	20	default	1070	1.02×10^{10} (58%)	85.08 (-0.77)	5
poly	20	precision	1172	1.03×10^{10} (59%)	85.25 (-0.60)	3
poly	20	max precision	1092	1.06×10^{10} (61%)	85.41 (-0.44)	2
poly	20	recall	1088	9.68×10^{09} (55%)	84.75 (-1.10)	7
poly	30	default	1222	1.20×10^{10} (69%)	85.57 (-0.28)	1
poly	30	precision	1267	1.20×10^{10} (69%)	85.62 (-0.23)	1
poly	30	max precision	1238	1.23×10^{10} (70%)	85.65 (-0.20)	1
poly	30	recall	1238	1.16×10^{10} (66%)	85.44 (-0.41)	2
poly	40	default	1465	1.49×10^{10} (85%)	85.72 (-0.13)	0
poly	40	precision	1353	1.49×10^{10} (85%)	85.75 (-0.10)	0
poly	40	max precision	1570	1.50×10^{10} (86%)	85.78 (-0.07)	0
poly	40	recall	1489	1.47×10^{10} (84%)	85.69 (-0.16)	1

Table 8: Results from parsing section 22 using constraints from both linear and polynomial classifiers, varying minimum sentence length and classifier probability threshold.