Pipeline Iteration

Kristy Hollingshead and **Brian Roark**
Center for Spoken Language Understanding, OGI School of Science & Engineering
Oregon Health & Science University, Beaverton, Oregon, 97006 USA
{hollingk,roark}@cslu.ogi.edu

Abstract

This paper presents pipeline iteration, an approach that uses output from later stages of a pipeline to constrain earlier stages of the same pipeline. We demonstrate significant improvements in a state-of-the-art PCFG parsing pipeline using base-phrase constraints, derived either from later stages of the parsing pipeline or from a finite-state shallow parser. The best performance is achieved by reranking the union of unconstrained parses and relatively heavily-constrained parses.

1 Introduction

A "pipeline" system consists of a sequence of processing stages such that the output from one stage provides the input to the next. Each stage in such a pipeline identifies a subset of the possible solutions, and later stages are constrained to find solutions within that subset. For example, a part-of-speech tagger could constrain a "base phrase" chunker (Ratnaparkhi, 1999), or the n-best output of a parser could constrain a reranker (Charniak and Johnson, 2005). A pipeline is typically used to reduce search complexity for rich models used in later stages, usually at the risk that the best solutions may be pruned in early stages.

Pipeline systems are ubiquitous in natural language processing, used not only in parsing (Ratnaparkhi, 1999; Charniak, 2000), but also machine translation (Och and Ney, 2003) and speech recognition (Fiscus, 1997; Goel et al., 2000), among others. Despite the widespread use of pipelines, they have been understudied, with very little work on general techniques for designing and improving pipeline systems (although *cf.* Finkel et al. (2006)). This paper presents one such general technique, here applied to stochastic parsing, whereby output from later stages of a pipeline is used to constrain earlier stages of the same pipeline. To our knowledge, this is the first time such a pipeline architecture has been proposed.

It may seem surprising that later stages of a pipeline, already constrained to be consistent with the output of earlier stages, can profitably inform the earlier stages in a second pass. However, the richer models used in later stages of a pipeline provide a better distribution over the subset of possible solutions produced by the early stages, effectively resolving some of the ambiguities that account for much of the original variation. If an earlier stage is then constrained in a second pass not to vary with respect to these resolved ambiguities, it will be forced to find other variations, which may include better solutions than were originally provided.

To give a rough illustration, consider the Venn diagram in Fig. 1(i). Set A represents the original subset of possible solutions passed along by the earlier stage, and the dark shaded region represents high-probability solutions according to later stages. If some constraints are then extracted from these high-probability solutions, defining a subset of solutions (S) that rule out some of A, the early stage will be forced to produce a different set (B). Constraints derived from later stages of the pipeline focus the search in an area believed to contain high-quality candidates.

Another scenario is to use a different model altogether to constrain the pipeline. In this scenario,

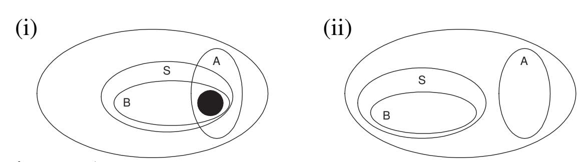

Figure 1: Two Venn diagrams, representing (i) constraints derived from later stages of an iterated pipelined system; and (ii) constraints derived from a different model.

Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics, pages 952–959,
Prague, Czech Republic, June 2007.

represented in Fig. 1(ii), the other model constrains the early stage to be consistent with some subset of solutions (S), which may be largely or completely disjoint from the original set A. Again, a different set (B) results, which may include better results than A. Whereas when iterating we are guaranteed that the new subset S will overlap at least partially with the original subset A, that is not the case when making use of constraints from a separately trained model.

In this paper, we investigate pipeline iteration within the context of the Charniak and Johnson (2005) parsing pipeline, by constraining parses to be consistent with a base-phrase tree. We derive these base-phrase constraints from three sources: the reranking stage of the parsing pipeline; a finite-state shallow parser (Hollingshead et al., 2005); and a combination of the output from these two sources. We compare the relative performance of these three sources and find the best performance improvements using constraints derived from a weighted combination of shallow parser output and reranker output.

The Charniak parsing pipeline has been extensively studied over the past decade, with a number of papers focused on improving early stages of the pipeline (Charniak et al., 1998; Caraballo and Charniak, 1998; Blaheta and Charniak, 1999; Hall and Johnson, 2004; Charniak et al., 2006) as well as many focused on optimizing final parse accuracy (Charniak, 2000; Charniak and Johnson, 2005; McClosky et al., 2006). This focus on optimization has made system improvements very difficult to achieve; yet our relatively simple architecture yields statistically significant improvements, making pipeline iteration a promising approach for other tasks.

2 Approach

Our approach uses the Charniak state-of-the-art parsing pipeline. The well-known Charniak (2000) coarse-to-fine parser is a two-stage parsing pipeline, in which the first stage uses a vanilla PCFG to populate a chart of parse constituents. The second stage, constrained to only those items in the first-stage chart, uses a refined grammar to generate an n-best list of parse candidates. Charniak and Johnson (2005) extended this pipeline with a discriminative maximum entropy model to rerank the n-best parse candidates, deriving a significant benefit from the richer model employed by the reranker.

For our experiments, we modified the parser[1] to allow us to optionally provide base-phrase trees to constrain the first stage of parsing.

[1] ftp://ftp.cs.brown.edu/pub/nlparser/

Parser		Base Phrases	Shallow Phrases
Charniak	parser-best	91.9	94.4
	reranker-best	92.8	94.8
Finite-state shallow parser		91.7	94.3

Table 1: F-scores on WSJ section 24 of output from two parsers on the similar tasks of base-phrase parsing and shallow-phrase parsing. For evaluation, base and shallow phrases are extracted from the Charniak/Johnson full-parse output.

2.1 Base Phrases

Following Ratnaparkhi (1999), we define a *base phrase* as any parse node with only preterminal children. Unlike the shallow phrases defined for the CoNLL-2000 Shared Task (Tjong Kim Sang and Buchholz, 2000), base phrases correspond directly to constituents that appear in full parses, and hence can provide a straightforward constraint on edges within a chart parser. In contrast, shallow phrases collapse certain non-constituents—such as auxiliary chains—into a single phrase, and hence are not directly applicable as constraints on a chart parser.

We have two methods for deriving base-phrase annotations for a string. First, we trained a finite-state shallow parser on base phrases extracted from the Penn Wall St. Journal (WSJ) Treebank (Marcus et al., 1993). The treebank trees are pre-processed identically to the procedure for training the Charniak parser, e.g., empty nodes and function tags are removed. The shallow parser is trained using the perceptron algorithm, with a feature set nearly identical to that from Sha and Pereira (2003), and achieves comparable performance to that paper. See Hollingshead et al. (2005) for more details. Second, base phrases can be extracted from the full-parse output of the Charniak and Johnson (2005) reranker, via a simple script to extract nodes with only preterminal children.

Table 1 shows these systems' bracketing accuracy on both the base-phrase and shallow parsing tasks for WSJ section 24; each system was trained on WSJ sections 02-21. From this table we can see that base phrases are substantially more difficult than shallow phrases to annotate. Output from the finite-state shallow parser is roughly as accurate as output extracted from the Charniak parser-best trees, though a fair amount below output extracted from the reranker-best trees.

In addition to using base phrase constraints from these two sources independently, we also looked at combining the predictions of both to obtain more reliable constraints. We next present a method of combining output from multiple parsers based on combined precision and recall optimization.

2.2 Combining Parser n-best Lists

In order to select high-likelihood constraints for the pipeline, we may want to extract annotations with high levels of agreement ("consensus hypotheses") between candidates. In addition, we may want to favor precision over recall to avoid erroneous constraints within the pipeline as much as possible. Here we discuss how a technique presented in Goodman's thesis (1998) can be applied to do this.

We will first present this within a general chart parsing approach, then move to how we use it for n-best lists. Let $\mathcal{T}$ be the set of trees for a particular input, and let a parse $T \in \mathcal{T}$ be considered as a set of labeled spans. Then, for all labeled spans $X \in T$, we can calculate the posterior probability $\gamma(X)$ as follows:

$$\gamma(X) = \sum_{T \in \mathcal{T}} \frac{\mathrm{P}(T)[\![X \in T]\!]}{\sum_{T' \in \mathcal{T}} \mathrm{P}(T')} \quad (1)$$

where $[\![X \in T]\!] = \begin{cases} 1 & \text{if } X \in T \\ 0 & \text{otherwise.} \end{cases}$

Goodman (1996; 1998) presents a method for using the posterior probability of constituents to maximize the expected labeled recall of binary branching trees, as follows:

$$\widehat{T} = \operatorname*{argmax}_{T \in \mathcal{T}} \sum_{X \in T} \gamma(X) \quad (2)$$

Essentially, find the tree with the maximum sum of the posterior probabilities of its constituents. This is done by computing the posterior probabilities of constituents in a chart, typically via the Inside-Outside algorithm (Baker, 1979; Lari and Young, 1990), followed by a final CYK-like pass to find the tree maximizing the sum.

For non-binary branching trees, where precision and recall may differ, Goodman (1998, Ch.3) proposes the following combined metric for balancing precision and recall:

$$\widehat{T} = \operatorname*{argmax}_{T \in \mathcal{T}} \sum_{X \in T} (\gamma(X) - \lambda) \quad (3)$$

where λ ranges from 0 to 1. Setting λ=0 is equivalent to Eq. 2 and thus optimizes recall, and setting λ=1 optimizes precision; Appendix 5 at the end of this paper presents brief derivations of these metrics.[2] Thus, λ functions as a mixing factor to balance recall and precision.

This approach also gives us a straightforward way to combine n-best outputs of multiple systems. To do this, we construct a chart of the constituents in the trees from the n-best lists, and allow any combination of constituents that results in a tree – even one with no internal structure. In such a way, we can produce trees that only include a small number of high-certainty constituents, and leave the remainder of the string unconstrained, even if such trees were not candidates in the original n-best lists.

For simplicity, we will here discuss the combination of two n-best lists, though it generalizes in the obvious way to an arbitrary number of lists. Let $\mathcal{T}$ be the union of the two n-best lists. For all trees $T \in \mathcal{T}$, let $\mathrm{P}_1(T)$ be the probability of T in the first n-best list, and $\mathrm{P}_2(T)$ the probability of T in the second n-best list. Then, we define $\mathrm{P}(T)$ as follows:

$$\mathrm{P}(T) = \alpha \frac{\mathrm{P}_1(T)}{\sum_{T' \in \mathcal{T}} \mathrm{P}_1(T')} + \frac{\mathrm{P}_2(T)}{\sum_{T' \in \mathcal{T}} \mathrm{P}_2(T')} \quad (4)$$

where the parameter α dictates the relative weight of P_1 versus P_2 in the combination.[3]

For this paper, we combined two n-best lists of base-phrase trees. Although there is no hierarchical structure in base-phrase annotations, they can be represented as flat trees, as shown in Fig. 2(a). We constructed a chart from the two lists being combined, using Eq. 4 to define $\mathrm{P}(T)$ in Eq. 1. We wish to consider every possible combination of the base phrases, so for the final CYK-like pass to find the argmax tree, we included rules for attaching each preterminal directly to the root of the tree, in addition to rules permitting any combination of hypothesized base phrases.

Consider the trees in Fig. 2. Figure 2(a) is a shallow parse with three NP base phrases; Figure 2(b) is the same parse where the ROOT production has been binarized for the final CYK-like pass, which requires binary productions. If we include productions of the form 'ROOT $\rightarrow$ X ROOT' and 'ROOT $\rightarrow$ X Y' for all non-terminals X and Y (including POS tags), then any tree-structured combination of base phrases hypothesized in either n-

[2] Our notation differs slightly from that in Goodman (1998), though the approaches are formally equivalent.

[3] Note that P_1 and P_2 are normalized in eq. 4, and thus are not required to be true probabilities. In turn, P is normalized when used in eq. 1, such that the posterior probability γ is a true probability. Hence P need not be normalized in eq. 4.

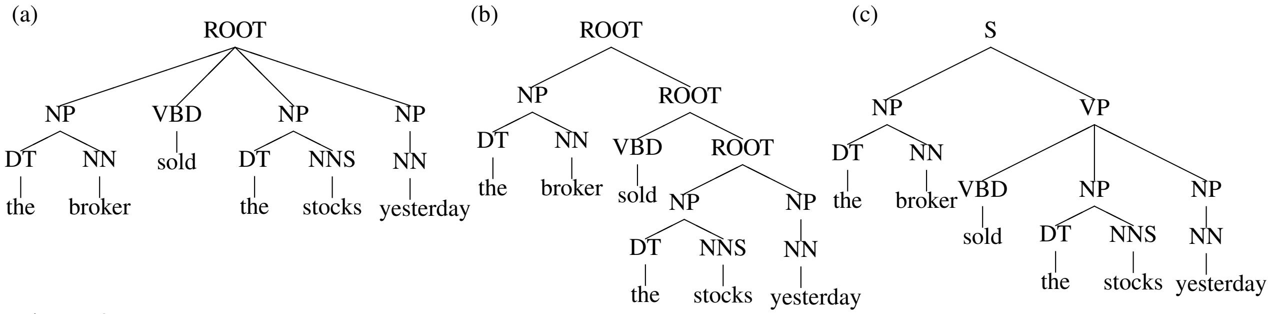

Figure 2: Base-phrase trees (a) as produced for an n-best list and (b) after root-binarization for n-best list combination. Full-parse tree (c) consistent with constraining base-phrase tree (a).

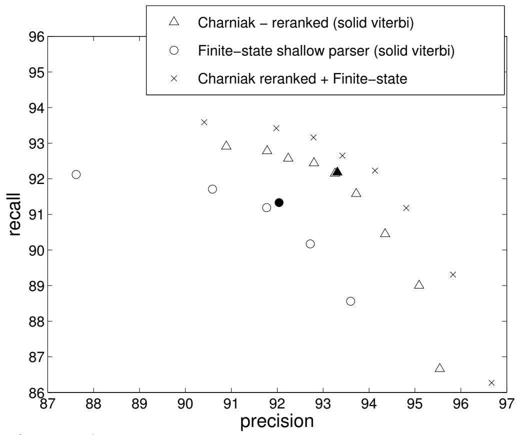

Figure 3: The tradeoff between recall and precision using a range of λ values (Eq. 3) to select high-probability annotations from an n-best list. Results are shown on 50-best lists of base-phrase parses from two parsers, and on the combination of the two lists.

best list is allowed, including the one with no base phrases at all. Note that, for the purpose of finding the argmax tree in Eq. 3, we only sum the posterior probabilities of base-phrase constituents, and not the ROOT symbol or POS tags.

Figure 3 shows the results of performing this combined precision/recall optimization on three separate n-best lists: the 50-best list of base-phrase trees extracted from the full-parse output of the Charniak and Johnson (2005) reranker; the 50-best list output by the Hollingshead et al. (2005) finite-state shallow parser; and the weighted combination of the two lists at various values of λ in Eq. 3. For the combination, we set α=2 in Eq. 4, with the Charniak and Johnson (2005) reranker providing P_1, effectively giving the reranker twice the weight of the shallow parser in determining the posteriors. The shallow parser has perceptron scores as weights, and the distribution of these scores after a softmax normalization was too peaked to be of utility, so we used the normalized reciprocal rank of each candidate as P_2 in Eq. 4.

We point out several details in these results. First, using this method does not result in an F-measure improvement over the Viterbi-best base-phrase parses (shown as solid symbols in the graph) for either the reranker or shallow parser. Also, using this model effects a greater improvement in precision than in recall, which is unsurprising with these non-hierarchical annotations; unlike full parsing (where long sequences of unary productions can improve recall arbitrarily), in base-phrase parsing, any given span can have only one non-terminal. Finally, we see that the combination of the two n-best lists improves over either list in isolation.

3 Experimental Setup

For our experiments we constructed a simple parsing pipeline, shown in Fig. 4. At the core of the pipeline is the Charniak and Johnson (2005) coarse-to-fine parser and MaxEnt reranker, described in Sec. 2. The parser constitutes the first and second stages of our pipeline, and the reranker the final stage. Following Charniak and Johnson (2005), we set the parser to output 50-best parses for all experiments described here. We constrain only the first stage of the parser: during chart construction, we disallow any constituents that conflict with the constraints, as described in detail in the next section.

3.1 Parser Constraints

We use base phrases, as defined in Sec. 2.1, to constrain the first stage of our parsing pipeline. Under these constraints, full parses must be consistent with the base-phrase tree provided as input to the parser, i.e., any valid parse must contain all of the base-phrase constituents in the constraining tree. The full-parse tree in Fig. 2(c), for example, is consistent with the base-phrase tree in Fig. 2(a).

Implementing these constraints in a parser is straightforward, one of the advantages of using base phrases as constraints. Since the internal structure of base phrases is, by definition, limited to preterminal children, we can constrain the entire parse by constraining the parents of the appropriate preterminal nodes. For any preterminal that occurs within the span of a constraining base phrase, the only valid parent is a node matching both the *span* (start and end points) and the *label* of the provided base

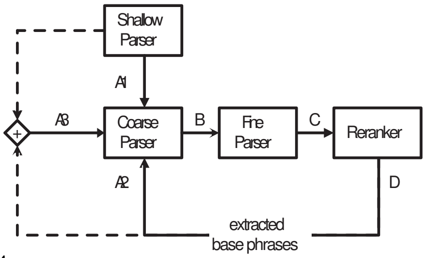

Figure 4: The iterated parsing pipeline. In the first iteration, the coarse parser may be either unconstrained, or constrained by base phrases from the shallow parser (A1). In the second iteration, base phrase constraints may be extracted either from reranker output (A2) or from a weighted combination of shallow parser output and reranker output (A3). Multiple sets of n-best parses, as output by the coarse-to-fine parser under different constraint conditions, may be joined in a set union (C).

phrase. All other proposed parent-nodes are rejected. In such a way, for any parse to cover the entire string, it would have to be consistent with the constraining base-phrase tree.

Words that fall outside of any base-phrase constraint are unconstrained in how they attach within the parse; hence, a base-phrase tree with few words covered by base-phrase constraints will result in a larger search space than one with many words covered by base phrases. We also put no restrictions on the preterminal labels, even within the base phrases. We normalized for punctuation. If the parser fails to find a valid parse with the constraints, then we lift the constraints and allow any parse constituent originally proposed by the first-stage parser.

3.2 Experimental Conditions

Our experiments will demonstrate the effects of constraining the Charniak parser under several different conditions. The baseline system places no constraints on the parser. The remaining experimental conditions each consider one of three possible sources of the base phrase constraints: (1) the base phrases output by the finite-state shallow parser; (2) the base phrases extracted from output of the reranker; and (3) a combination of the output from the shallow parser and the reranker, which is produced using the techniques outlined in Sec. 2.2. Constraints are enforced as described in Sec. 3.1.

Unconstrained For our baseline system, we run the Charniak and Johnson (2005) parser and reranker with default parameters. The parser is provided with treebank-tokenized text and, as mentioned previously, outputs 50-best parse candidates to the reranker.

FS-constrained The FS-constrained condition provides a comparison point of non-iterated constraints. Under this condition, the one-best base-phrase tree output by the finite-state shallow parser is input as a constraint to the Charniak parser. We run the parser and reranker as before, under constraints from the shallow parser. The accuracy of the constraints used under this condition is shown in the first row of Table 2. Note that this condition is *not* an instance of pipeline iteration, but is included to show the performance levels that can be achieved without iteration.

System	LR	LP	F
Finite-state shallow parser	91.3	92.0	91.7
Charniak reranker-best	92.2	93.3	92.8
Combination (λ=0.5)	92.2	94.1	93.2
Combination (λ=0.9)	81.0	97.4	88.4

Table 2: Labeled recall (LR), precision (LP), and F-scores on WSJ section 24 of base-phrase trees produced by the three possible sources of constraints.

Reranker-constrained We will use the reranker-constrained condition to examine the effects of pipeline iteration, with no input from other models outside the pipeline. We take the reranker-best full-parse output under the condition of unconstrained search, and extract the corresponding base-phrase tree. We run the parser and reranker as before, now with constraints from the reranker. The accuracy of the constraints used under this condition is shown in the second row of Table 2.

Combo-constrained The combo-constrained conditions are designed to compare the effects of generating constraints with different combination parameterizations, i.e., different λ parameters in Eq. 3. For this experimental condition, we extract base-phrase trees from the n-best full-parse trees output by the reranker. We combine this list with the n-best list output by the finite-state shallow parser, exactly as described in Sec. 2.2, again with the reranker providing P_1 and α=2 in Eq. 4. We examined a range of operating points from λ=0.4 to λ=0.9, and report two points here (λ=0.5 and λ=0.9), which represent the highest overall accuracy and the highest precision, respectively, as shown in Table 2.

Constrained and Unconstrained Union When iterating this pipeline, the original n-best list of full parses output from the unconstrained parser is available at no additional cost, and our final set of experimental conditions investigate taking the union of constrained and unconstrained n-best lists. The imposed constraints can result in candidate sets that are largely (or completely) disjoint from the unconstrained sets, and it may be that the unconstrained set is in many cases superior to the constrained set.

Constraints	Parser-best	Reranker-best	Oracle-best	# Candidates
Baseline (Unconstrained, 50-best)	88.92	90.24	95.95	47.9
FS-constrained	88.44	89.50	94.10	46.2
Reranker-constrained	89.60	90.46	95.07	46.9
Combo-constrained (λ=0.5)	89.81	90.74	95.41	46.3
Combo-constrained (λ=0.9)	89.34	90.43	95.91	47.5

Table 3: Full-parse F-scores on WSJ section 24. The unconstrained search (first row) provides a baseline comparison for the effects of constraining the search space. The last four rows demonstrate the effect of various constraint conditions.

Even our high-precision constraints did not reach 100% precision, attesting to the fact that there was some error in all constrained conditions. By constructing the union of the two n-best lists, we can take advantage of the new constrained candidate set without running the risk that the constraints have resulted in a worse n-best list. Note that the parser probabilities are produced from the same model in both passes, and are hence directly comparable.

The output of the second pass of the pipeline could be used to constrain a third pass, for multiple pipeline iterations. However, we found that further iterations provided no additional improvements.

3.3 Data

Unless stated otherwise, all reported results will be F-scores on WSJ section 24 of the Penn WSJ Treebank, which was our development set. Training data was WSJ sections 02-21, with section 00 as held-out data. Crossfold validation (20-fold with 2,000 sentences per fold) was used to train the reranker for every condition. Evaluation was performed using `evalb` under standard parameterizations. WSJ section 23 was used only for final testing.

4 Results & Discussion

We evaluate the one-best parse candidates before and after reranking (*parser-best* and *reranker-best*, respectively). We additionally provide the best-possible F-score in the n-best list (*oracle-best*) and the number of unique candidates in the list.

Table 3 presents trials showing the effect of constraining the parser under various conditions. Constraining the parser to the base phrases produced by the finite-state shallow parser (FS-constrained) hurts performance by half a point. Constraining the parser to the base phrases produced by the reranker, however, provides a 0.7 percent improvement in the parser-best accuracy, and a 0.2 percent improvement after reranking. Combining the two base-phrase n-best lists to derive the constraints provides further improvements when λ=0.5, to a total improvement of 0.9 and 0.5 percent over parser-best and reranker-best accuracy, respectively. Performance degrades at λ=0.9 relative to λ=0.5, indicating that, even at a lower precision, more constraints are beneficial.

The oracle rate decreases under all of the constrained conditions as compared to the baseline, demonstrating that the parser was prevented from finding some of the best solutions that were originally found. However, the improvement in F-score shows that the constraints assisted the parser in achieving high-quality solutions despite this degraded oracle accuracy of the lists.

Table 4 shows the results when taking the union of the constrained and unconstrained lists prior to reranking. Several interesting points can be noted in this table. First, despite the fact that the FS-constrained condition hurts performance in Table 3, the union provides a 0.5 percent improvement over the baseline in the parser-best performance. This indicates that, in some cases, the Charniak parser is scoring parses in the constrained set higher than in the unconstrained set, which is evidence of search errors in the unconstrained condition. One can see from the number of candidates that the FS-constrained condition provides the set of candidates most disjoint from the original unconstrained parser, leading to the largest number of candidates in the union. Surprisingly, even though this set provided the highest parser-best F-score of all of the union sets, it did not lead to significant overall improvements after reranking.

In all other conditions, taking the union decreases the parser-best accuracy when compared to the corresponding constrained output, but improves the reranker-best accuracy in all but the combo-constrained λ=0.9 condition. One explanation for the lower performance at λ=0.9 versus λ=0.5 is seen in the number of candidates, about 7.5 fewer than in the λ=0.5 condition. There are fewer constraints in the high-precision condition, so the resulting n-best lists do not diverge as much from the original lists, leading to less diversity in their union.

The gains in performance should not be attributed to increasing the number of candidates nor to allow-

Constraints	Parser-best	Reranker-best	Oracle-best	# Candidates
Baseline (Unconstrained, 50-best)	88.92	90.24	95.95	47.9
Unconstrained $\cup$ FS-constrained	89.39	90.27	96.61	74.9
Unconstrained $\cup$ Reranker-constrained	89.23	90.59	96.48	70.3
Unconstrained $\cup$ Combo (λ=0.5)	89.28	90.78	96.53	69.7
Unconstrained $\cup$ Combo (λ=0.9)	89.03	90.44	96.40	62.1
Unconstrained (100-best)	88.82	90.13	96.38	95.2
Unconstrained (50-best, beam$\times$2)	89.01	90.45	96.13	48.1

Table 4: Full-parse F-scores on WSJ section 24 after taking the set union of unconstrained and constrained parser output under the 4 different constraint conditions. Also, F-score for 100-best parses, and 50-best parses with an increased beam threshold, output by the Charniak parser under the unconstrained condition.

Constraints	F-score
Baseline (Unconstrained, 50-best)	91.06
Unconstrained $\cup$ Combo (λ=0.5)	91.48

Table 5: Full-parse F-scores on WSJ section 23 for our best-performing system on WSJ section 24. The 0.4 percent F-score improvement is significant at $p < 0.001$.

ing the parser more time to generate the parses. The penultimate row in Table 4 shows the results with 100-best lists output in the unconstrained condition, which does not improve upon the 50-best performance, despite an improved oracle F-score. Since the second iteration through the parsing pipeline clearly increases the overall processing time by a factor of two, we also compare against output obtained by doubling the coarse-parser's beam threshold. The last row in Table 4 shows that the increased threshold yields an insignificant improvement over the baseline, despite a very large processing burden.

We applied our best-performing model (Unconstrained $\cup$ Combo, λ=0.5) to the test set, WSJ section 23, for comparison against the baseline system. Table 5 shows a 0.4 percent F-score improvement over the baseline for that section, which is statistically significant at $p < 0.001$, using the stratified shuffling test (Yeh, 2000).

5 Conclusion & Future Work

In summary, we have demonstrated that pipeline iteration can be useful in improving system performance, by constraining early stages of the pipeline with output derived from later stages. While the current work made use of a particular kind of constraint—base phrases—many others could be extracted as well. Preliminary results extending the work presented in this paper show parser accuracy improvements from pipeline iteration when using constraints based on an unlabeled partial bracketing of the string. Higher-level phrase segmentations or fully specified trees over portions of the string might also prove to be effective constraints. The techniques shown here are by no means limited to parsing pipelines, and could easily be applied to other tasks making use of pipeline architectures.

Acknowledgments

Thanks to Martin Jansche for useful discussions on topics related to this paper. The first author of this paper was supported under an NSF Graduate Research Fellowship. In addition, this research was supported in part by NSF Grant #IIS-0447214. Any opinions, findings, conclusions or recommendations expressed in this publication are those of the authors and do not necessarily reflect the views of the NSF.

References

J.K. Baker. 1979. Trainable grammars for speech recognition. In *Speech Communication papers for the 97th Meeting of the Acoustical Society of America*.

D. Blaheta and E. Charniak. 1999. Automatic compensation for parser figure-of-merit flaws. In *Proceedings of the 37th Annual Meeting of ACL*, pages 513–518.

S. Caraballo and E. Charniak. 1998. New figures of merit for best-first probabilistic chart parsing. *Computational Linguistics*, 24(2):275–298.

E. Charniak and M. Johnson. 2005. Coarse-to-fine n-best parsing and MaxEnt discriminative reranking. In *Proceedings of the 43rd Annual Meeting of ACL*, pages 173–180.

E. Charniak, S. Goldwater, and M. Johnson. 1998. Edge-based best-first chart parsing. In *Proceedings of the 6th Workshop for Very Large Corpora*, pages 127–133.

E. Charniak, M. Johnson, M. Elsner, J.L. Austerweil, D. Ellis, S.R. Iyangar, J. Moore, M.T. Pozar, C. Hill, T.Q. Vu, and I. Haxton. 2006. Multi-level course-to-fine PCFG parsing. In *Proceedings of the HLT-NAACL Annual Meeting*, pages 168–175.

E. Charniak. 2000. A Maximum-Entropy-inspired parser. In *Proceedings of the 1st Annual Meeting of NAACL and 6th Conference on ANLP*, pages 132–139.

J.R. Finkel, C.D. Manning, and A.Y. Ng. 2006. Solving the problem of cascading errors: Approximate Bayesian inference for linguistic annotation pipelines. In *Proceedings of EMNLP*, pages 618–626.

J. Fiscus. 1997. A post-processing system to yield reduced word error rates: Recognizer output voting error reduction (ROVER). In *Proceedings of the IEEE Workshop on Automatic Speech Recognition and Understanding*.

V. Goel, S. Kumar, and W. Byrne. 2000. Segmental minimum Bayes-risk ASR voting strategies. In *Proceedings of ICSLP*, pages 139–142.

J. Goodman. 1996. Parsing algorithms and metrics. In *Proceedings of the 34th Annual Meeting of ACL*, pages 177–183.

J. Goodman. 1998. *Parsing inside-out*. Ph.D. thesis, Harvard University.

K. Hall and M. Johnson. 2004. Attention shifting for parsing speech. In *Proceedings of the 42nd Annual Meeting of ACL*, pages 40–46.

K. Hollingshead, S. Fisher, and B. Roark. 2005. Comparing and combining finite-state and context-free parsers. In *Proceedings of HLT-EMNLP*, pages 787–794.

K. Lari and S.J. Young. 1990. The estimation of stochastic context-free grammars using the inside-outside algorithm. *Computer Speech and Language*, 4(1):35–56.

M.P. Marcus, M.A. Marcinkiewicz, and B. Santorini. 1993. Building a large annotated corpus of English: The Penn treebank. *Computational Linguistics*, 19:314–330.

D. McClosky, E. Charniak, and M. Johnson. 2006. Reranking and self-training for parser adaptation. In *Proceedings of COLING-ACL*, pages 337–344.

F.J. Och and H. Ney. 2003. A systematic comparison of various statistical alignment models. *Computational Linguistics*, 29.

A. Ratnaparkhi. 1999. Learning to parse natural language with maximum entropy models. *Machine Learning*, 34(1-3):151–175.

F. Sha and F. Pereira. 2003. Shallow parsing with conditional random fields. In *Proceedings of the HLT-NAACL Annual Meeting*, pages 134–141.

E.F. Tjong Kim Sang and S. Buchholz. 2000. Introduction to the CoNLL-2000 shared task: Chunking. In *Proceedings of CoNLL*, pages 127–132.

A. Yeh. 2000. More accurate tests for the statistical significance of result differences. In *Proceedings of the 18th International COLING*, pages 947–953.

Appendix A Combined Precision/Recall Decoding

Recall that $\mathcal{T}$ is the set of trees for a particular input, and each $T \in \mathcal{T}$ is considered as a set of labeled spans. For all labeled spans $X \in T$, we can calculate the posterior probability $\gamma(X)$ as follows:

$$\gamma(X) = \sum_{T \in \mathcal{T}} \frac{\mathrm{P}(T)[\![X \in T]\!]}{\sum_{T' \in \mathcal{T}} \mathrm{P}(T')}$$

where $[\![X \in T]\!] = \begin{cases} 1 & \text{if } X \in T \\ 0 & \text{otherwise.} \end{cases}$

If τ is the reference tree, the labeled precision (LP) and labeled recall (LR) of a T relative to τ are defined as

$$\mathrm{LP} = \frac{|T \cap \tau|}{|T|} \qquad \mathrm{LR} = \frac{|T \cap \tau|}{|\tau|}$$

where $|T|$ denotes the size of the set T.

A metric very close to LR is $|T \cap \tau|$, the number of nodes in common between the tree and the reference tree. To maximize the expected value ($\mathcal{E}$) of this metric, we want to find the tree $\widehat{T}$ as follows:

$$\begin{aligned} \widehat{T} &= \operatorname*{argmax}_{T \in \mathcal{T}} \mathcal{E}\left[|T \bigcap \tau|\right] \\ &= \operatorname*{argmax}_{T \in \mathcal{T}} \sum_{T' \in \mathcal{T}} \frac{\mathrm{P}(T')\left[|T \bigcap T'|\right]}{\sum_{T'' \in \mathcal{T}} \mathrm{P}(T'')} \\ &= \operatorname*{argmax}_{T \in \mathcal{T}} \sum_{T' \in \mathcal{T}} \frac{\mathrm{P}(T') \sum_{X \in T} [\![X \in T']\!]}{\sum_{T'' \in \mathcal{T}} \mathrm{P}(T'')} \\ &= \operatorname*{argmax}_{T \in \mathcal{T}} \sum_{X \in T} \sum_{T' \in \mathcal{T}} \frac{\mathrm{P}(T')[\![X \in T']\!]}{\sum_{T'' \in \mathcal{T}} \mathrm{P}(T'')} \\ &= \operatorname*{argmax}_{T \in \mathcal{T}} \sum_{X \in T} \gamma(X) \qquad (5) \end{aligned}$$

This exactly maximizes the expected LR in the case of binary branching trees, and is closely related to LR for non-binary branching trees. Similar to maximizing the expected number of matching nodes, we can minimize the expected number of non-matching nodes, for a metric related to LP:

$$\begin{aligned} \widehat{T} &= \operatorname*{argmin}_{T \in \mathcal{T}} \mathcal{E}\left[|T| - |T \bigcap \tau|\right] \\ &= \operatorname*{argmax}_{T \in \mathcal{T}} \mathcal{E}\left[|T \bigcap \tau| - |T|\right] \\ &= \operatorname*{argmax}_{T \in \mathcal{T}} \sum_{T' \in \mathcal{T}} \frac{\mathrm{P}(T')\left[|T \bigcap T'| - |T|\right]}{\sum_{T'' \in \mathcal{T}} \mathrm{P}(T'')} \\ &= \operatorname*{argmax}_{T \in \mathcal{T}} \sum_{T' \in \mathcal{T}} \frac{\mathrm{P}(T') \sum_{X \in T}([\![X \in T']\!] - 1)}{\sum_{T'' \in \mathcal{T}} \mathrm{P}(T'')} \\ &= \operatorname*{argmax}_{T \in \mathcal{T}} \sum_{X \in T} \sum_{T' \in \mathcal{T}} \frac{\mathrm{P}(T')([\![X \in T']\!] - 1)}{\sum_{T'' \in \mathcal{T}} \mathrm{P}(T'')} \\ &= \operatorname*{argmax}_{T \in \mathcal{T}} \sum_{X \in T} (\gamma(X) - 1) \qquad (6) \end{aligned}$$

Finally, we can combine these two metrics in a linear combination. Let λ be a mixing factor from 0 to 1. Then we can optimize the weighted sum:

$$\begin{aligned} \widehat{T} &= \operatorname*{argmax}_{T \in \mathcal{T}} \mathcal{E}\left[(1-\lambda)|T \bigcap \tau| + \lambda(|T \bigcap \tau| - |T|)\right] \\ &= \operatorname*{argmax}_{T \in \mathcal{T}} (1-\lambda)\mathcal{E}\left[|T \bigcap \tau|\right] + \lambda\mathcal{E}\left[|T \bigcap \tau| - |T|\right] \\ &= \operatorname*{argmax}_{T \in \mathcal{T}} \left[(1-\lambda) \sum_{X \in T} \gamma(X)\right] + \left[\lambda \sum_{X \in T} (\gamma(X) - 1)\right] \\ &= \operatorname*{argmax}_{T \in \mathcal{T}} \sum_{X \in T} (\gamma(X) - \lambda) \qquad (7) \end{aligned}$$

The result is a combined metric for balancing precision and recall. Note that, if λ=0, Eq. 7 is the same as Eq. 5 and thus maximizes the LR metric; and if λ=1, Eq. 7 is the same as Eq. 6 and thus maximizes the LP metric.