Today's Papers

- Example Selection for Bootstrapping Statistical Parsers (2003)
  - Mark Steedman, Rebecca Hwa, Stephen Clark, Miles Osborne, Anoop Sarkar, Julia Hockenmaier, Paul Ruhlen, Steven Baker, Jeremiah Crim
- Parsing with Treebank Grammars: Empirical Bounds, Theoretical Models, and the Structure of the Penn Treebank (2001)
  - Dan Klein and Christopher D. Manning

Co-training

- Idea: multiple classifiers (parsers) train each other
- Assumes parsers use independent models
- During each co-training iteration:
  - Select a small cache of unlabeled sentences
  - Run parsers A and B on the cache
  - Score each parse output from each parser
  - Select some of A's parses to add to the training set of B, select some of B's parses to add to the training set of A
  - Retrain both A and B
- Corrected co-training: human checks & corrects parser output before it is added to training set

Co-training (2)

- Problem 1: How do we score output of parser?
- Problem 2: How do we do sample selection?
  - Intuitively, we should use only <u>accurate</u> output
  - But also choose examples with <u>high training utility</u>
- This paper: Sample selection for co-training
  - Opposing goals: want training samples to have both <u>high training utility</u> and <u>high accuracy</u>

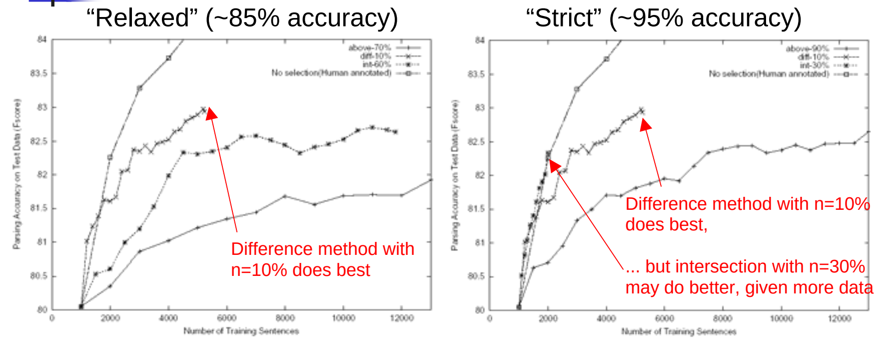
Scoring & selecting training examples

- Parse scoring
  - Optimal: comparison to human-labeled ground truth
  - Practical: likelihood of parse given model
- Training example selection
  - Above-n: (select high-quality samples)
    - (score of teacher's parse) $\geq$ n
  - Difference: (select high-utility samples)
    - (score of teacher's parse) – (score of student's parse) $\geq$ n
  - Intersection: (high-quality, high-utility samples)
    - (score of teacher's parse in highest n percentile) <u>and</u> (score of student's parse in lowest n percentile)

Experimental protocol

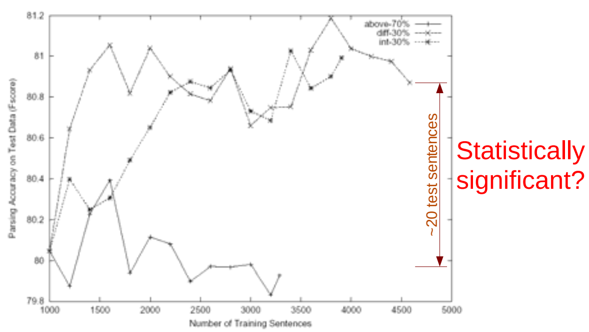
- Two parsers
  - Lexicalized context free grammar parser [Collins99]
  - Lexicalized tree adjoining grammar parser [Sarkar02]
- Seed (labelled) training data: 1,000 sentences
- Unlabelled training data: ~38,000 sentences
- Cache size: 500 sentences
- Test data: independent, ~2,400 sentences

Results, ideal scoring function



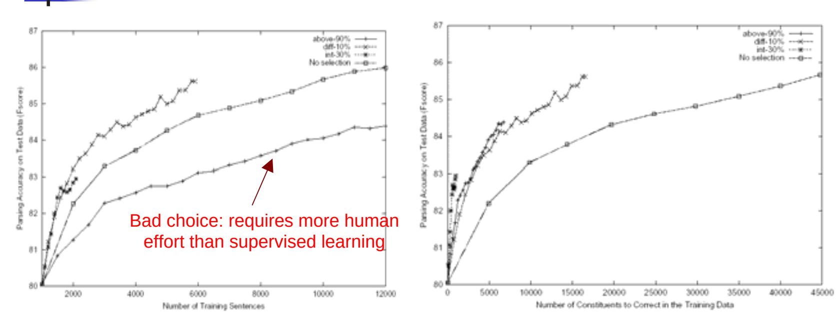
- Conclusion: utility is more important than accuracy
- But:
  - Statistical significance (e.g. error bars)?
  - What about other values of n?

Results, practical scoring function



- ~1 percentage point gain using diff-30% or int-30%
- Again, utility more important than accuracy

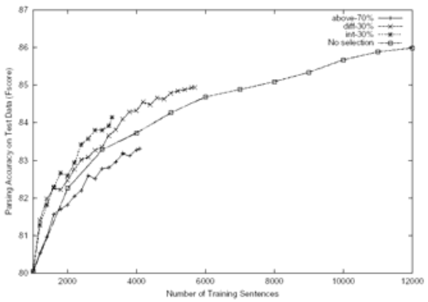
Corrected co-training, ideal scoring



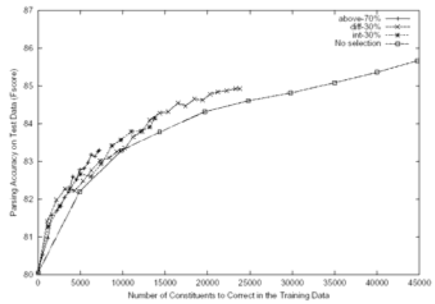
(i.e. # of sentences that must be checked by human) (i.e. # of constituents that must be corrected by human)

- Intersection (n=30%) may be best choice based on growth rate (need more data to confirm)

Corrected co-training, practical scoring



(i.e. # of sentences that must be checked by human)



(i.e. # of constituents that must be corrected by human)

- Corrected co-training still saves human effort
- Again, utility is more important than accuracy
  (assuming results are statistically significant)

Conclusions

- Selection methods emphasizing high training utility do best, even at the expense of lower accuracy
- Quality of scoring function important
  - For ideal scoring function, co-training significantly improved parser performance (2-3 percentage points)
  - For practical scoring function, co-training improved performance only marginally (< 1 percentage point)
  - (so better scoring functions are needed...)
- Corrected co-training with high training utility selection further increases performance, with less human effort than a supervised method
- Future work: try other pairs (sets) of parsers

Today's Papers

- Example Selection for Bootstrapping Statistical Parsers (2003)
  - Mark Steedman, Rebecca Hwa, Stephen Clark, Miles Osborne, Anoop Sarkar, Julia Hockenmaier, Paul Ruhlen, Steven Baker, Jeremiah Crim
- Parsing with Treebank Grammars: Empirical Bounds, Theoretical Models, and the Structure of the Penn Treebank (2001)
  - Dan Klein and Christopher D. Manning

Paper by Klein & Manning

- Idea: build a grammar by observing the hand-labelled parses of the Penn treebank
- But: this can create huge grammars
  - [Charniak96] found 10,605 rules on the Penn treebank, and less than 40% of those occurred more than once
- How fast (slow) are chart parsers that use these grammars?
- Which parameters affect parsing speed and memory use? How can we model these effects?

Parameters (1)

- Tree transforms
  - None, NoEmpties, NoUnaries

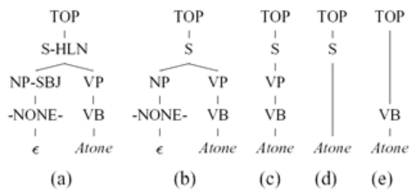


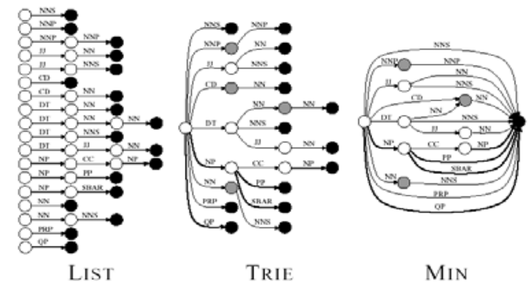
Figure 1: Tree Transforms: (a) The raw tree, (b) NoTransform, (c) NoEmpties, (d) NoUnariesHigh (e) NoUnariesLow

Parameters (2)

- Grammar encoding
  - List, trie, or minimized DFA
  - All encodings equivalent (do not affect parser output)

e.g., rules for noun phrase:

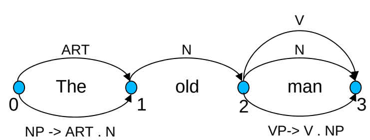
(gray and white states are accepting)



- Rule ordering
  - Top-down or Bottom-up

Background: chart parsers (1)

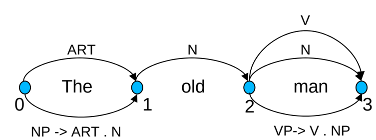
- *C* categories, *S* states in the grammar DFA
- Chart: nodes and edges
- Node: placed between each word of sentence
  - (so *n+1* nodes for a sentence with *n* words)
- Span: range of words (e.g. [0,2] refers to *The old*)
  - (there are $O(n^2)$ possible spans)



S -> NP VP
NP -> ART N
VP -> V NP | V
V -> man
N -> man | old
ART -> the

Background: chart parsers (2)

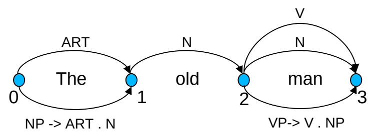
- Edge: associates a category with a span (e.g. N:[1,2])
  - Passive edge: e.g. ART:[0,1]
    - Means the span belongs to the category
    - There are $O(Cn^2)$ possible passive edges (~2% of edges)
  - Active edge: e.g. NP -> ART . N:[0,1]
    - Means that if we find some node k such that [1,k] is a noun, then [0,k] is a noun phrase
    - There are $O(Sn^2)$ possible active edges (~98% of edges)



S -> NP VP
NP -> ART N
VP -> V NP | V
V -> man
N -> man | old
ART -> the

Background: chart parsers (3)

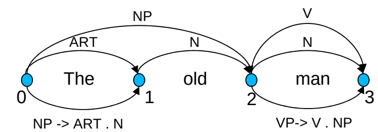
- Saturation of a span: # of edges over that span
- Traversal: combining an active edge and a passive edge to form a new edge
  - e.g. (NP -> ART . N:[0,1] + Noun:[1,2]) => NP:[0,2]
  - # of traversals bounded by $O(SCn^3)$
- Computation cost proportional to number of traversals
- Memory use proportional to number of active edges [$O(Sn^2)$]



S -> NP VP
NP -> ART N
VP -> V NP | V
V -> man
N -> man | old
ART -> the

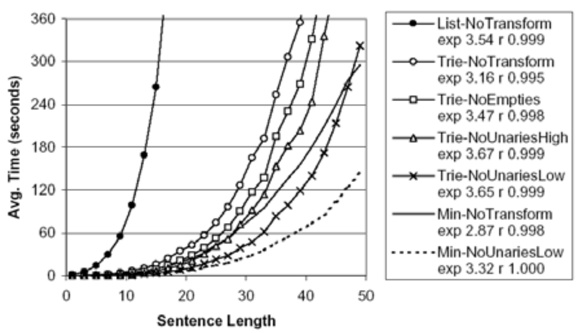
Background: chart parsers (3)

- Saturation of a span: # of edges over that span
- Traversal: combining an active edge and a passive edge to form a new edge
  - e.g. (NP -> ART . N:[0,1] + Noun:[1,2]) => NP:[0,2]
  - # of traversals bounded by $O(SCn^3)$
- Computation cost proportional to number of traversals
- Memory use proportional to number of active edges [$O(Sn^2)$]



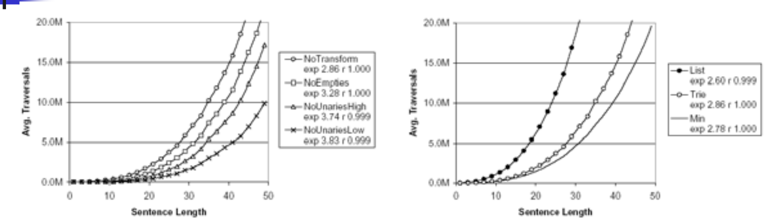
S -> NP VP
NP -> ART N
VP -> V NP | V
V -> man
N -> man | old
ART -> the

Results: parsing time



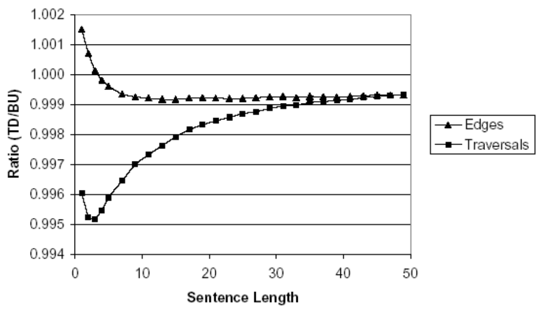
- Fit power law model $(y=Ax^B)$ to data
- Some exponents > 3 (asymptotic worst case)
- Execution time affected by Java garbage collection
  - Traversal count a better measure of execution time

Results: traversal counts



- Simpler grammars (more aggressive tree transforms) produce faster parsers
  - But affect utility of parses
- Fewer states (more aggressive state reduction) in grammar encoding produce faster parsers

Results: top-down vs. bottom-up



- Top-down parsing slightly more efficient

Modeling passive edge count

$$ptot(n) = \sum_{i=0}^{n} (n+1-i) psat_i$$

Avg # of passive edges for a sentence of length n

# of possible spans of length i

Average passive saturation of a span of length i

- Find empirically that $psat_i$ for $i \geq 2$ is relatively constant, so this sum can be approximated as (for NoTransform and NoEmpties):

$$ptot(n) = \frac{(n-1)n}{2} psat_2 + (n) psat_1 + (n+1) psat_0$$

Modeling active edge count

- Assume a random tag matches a random word with some fixed probability *p*
- We can characterize an active state by the number of tags *t* and categories *c* that must be matched
  - For an active state *a*, *sigma(a)=(t,c)* is its signature
- Approximate active edge count by summing over signatures:

$$asat(n) = \sum_{\sigma} count(\sigma) E_{a \in \sigma}[P(match(a,n)]$$

Avg # of active edges for a span of length *n*

# of active states having signature *sigma*

Expected probability that a random rule of signature *sigma* will match a random span of length *n*

- *count* and *p* are parameters estimated from treebank

Modeling traversal count

- Predict traversals from passive and active edge models
- Assume that a given active edge and a given passive edge can be combined into a traversal with the following fixed probability:

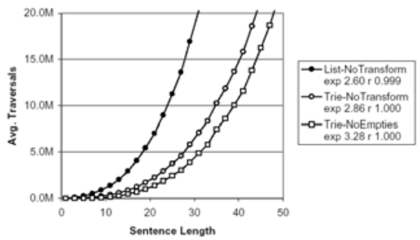
$$\frac{\text{Avg outgoing degree of FSA}}{\text{\# of labels (categories or POS tags)}}$$

1 for lists
~3.7 for tries
~4.2 for min. FSAs

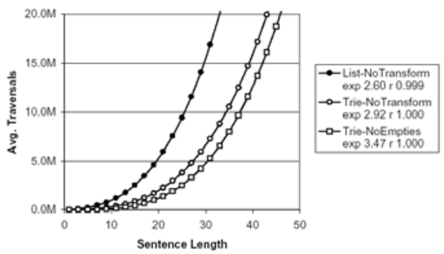
73

Traversal counts: observed vs. modeled

Observed



Predicted



Conclusions

- Simple models can be built to predict parse time, memory requirements
- Ordering (top-down or bottom-up) has little effect on performance
- Choice of grammar encoding has greatest effect on parser
  - This is good, since choice of tree transform is highly application-sensitive