Dynamic Programming for Linear-Time Incremental Parsing

Liang Huang
Information Sciences Institute
University of Southern California

(Joint work with Kenji Sagae, USC/ICT)

JHU CLSP Seminar    September 14, 2010
Prof. Jelinek hosted my visit and this talk on his last day.

He was very supportive of this work, which is related to his work on structured language models, and I dedicate my work to his memory.
Ambiguity and Incrementality

• NLP is (almost) all about ambiguity resolution
• human-beings resolve ambiguity incrementally
Ambiguity and Incrementality

- NLP is (almost) all about ambiguity resolution
- Human-beings resolve ambiguity **incrementally**

One morning in Africa,
I shot an elephant in my pajamas;
Ambiguity and Incrementality

- NLP is (almost) all about ambiguity resolution
- Human-beings resolve ambiguity incrementally

One morning in Africa,
I shot an elephant in my pajamas;
how he got into my pajamas I’ll never know.
Ambiguity and Incrementality

- NLP is (almost) all about ambiguity resolution
- human-beings resolve ambiguity incrementally

One morning in Africa,
I shot an elephant in my pajamas;
how he got into my pajamas I’ll never know.
Ambiguity and Incrementality

• NLP is (almost) all about ambiguity resolution
• human-beings resolve ambiguity incrementally

One morning in Africa,
I shot an elephant in my pajamas;
how he got into my pajamas I’ll never know.
Ambiguities in Translation
Ambiguities in Translation

Slip carefully

BE CAREFUL OF LANDSLIDE
Ambiguities in Translation

Google translate: carefully slide
Ambiguities in Translation

Google translate: carefully slide
Ambiguities in Translation

Google translate: carefully slide
If you are stolen...

If you are stolen, call the police at once.

Once lost, report to the police, do not hesitate.
If you are stolen...
or even...
or even...

clear evidence that NLP is used in real life!
Ambiguities in Parsing

- let's focus on dependency structures for simplicity
- ambiguous attachments of *nearby* and *in*
- ambiguity explodes *exponentially* with sentence length
- must design efficient (polynomial) search algorithm
  - typically using dynamic programming (DP); e.g. CKY
Ambiguities in Parsing

- let’s focus on dependency structures for simplicity
- ambiguous attachments of nearby and in
- ambiguity explodes exponentially with sentence length
- must design efficient (polynomial) search algorithm
  - typically using dynamic programming (DP); e.g. CKY
Ambiguities in Parsing

- Let's focus on dependency structures for simplicity.
- Ambiguous attachments of *nearby* and *in*.
- Ambiguity explodes *exponentially* with sentence length.
- Must design efficient (polynomial) search algorithm.
  - Typically using dynamic programming (DP); e.g. CKY.
Ambiguities in Parsing

- let’s focus on dependency structures for simplicity
- ambiguous attachments of *nearby* and *in*
- ambiguity explodes *exponentially* with sentence length
- must design efficient (polynomial) search algorithm
  - typically using dynamic programming (DP); e.g. CKY
I feed cats nearby in the garden...

- full DP (like CKY) is too slow (cubic-time)
- while human parsing is fast & incremental (linear-time)
But full DP is too slow...

I feed cats nearby in the garden ...

• full DP (like CKY) is too slow (cubic-time)

• while human parsing is fast & incremental (linear-time)

• how about incremental parsing then?

  • yes, but only with greedy search (accuracy suffers)

  • explores tiny fraction of trees (even w/ beam search)
But full DP is too slow...

I feed cats nearby in the garden ...

- full DP (like CKY) is too slow (cubic-time)
- while human parsing is fast & incremental (linear-time)
- how about incremental parsing then?
  - yes, but only with greedy search (accuracy suffers)
  - explores tiny fraction of trees (even w/ beam search)
- can we combine the merits of both approaches?
  - a fast, incremental parser with dynamic programming?
  - explores exponentially many trees in linear-time?
## Linear-Time Incremental DP

<table>
<thead>
<tr>
<th>Greedy Search</th>
<th>Incremental Parsing (e.g. shift-reduce) (Nivre 04; Collins/Roark 04; ...)</th>
<th>Full DP (e.g. CKY) (Eisner 96; Collins 99; ...)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Principled Search</td>
<td><strong>This work:</strong> fast shift-reduce parsing with dynamic programming</td>
<td><strong>Fast</strong> (linear-time)</td>
</tr>
<tr>
<td></td>
<td>natural languages</td>
<td>programming languages</td>
</tr>
<tr>
<td>----------------</td>
<td>-------------------</td>
<td>-----------------------</td>
</tr>
<tr>
<td>human</td>
<td>☑️ psycholinguistics ☑️</td>
<td>☐ compiler theory (LR, LALR, ...)</td>
</tr>
<tr>
<td>computer</td>
<td>☐ NLP ☐</td>
<td>☑️</td>
</tr>
</tbody>
</table>

DP for Incremental Parsing
## Big Picture

<table>
<thead>
<tr>
<th></th>
<th>natural languages</th>
<th>programming languages</th>
</tr>
</thead>
<tbody>
<tr>
<td>human</td>
<td>☑</td>
<td>☕</td>
</tr>
<tr>
<td>psycholinguistics</td>
<td></td>
<td></td>
</tr>
<tr>
<td>computer</td>
<td>☕</td>
<td>☑ compiler theory (LR, LALR, ...)</td>
</tr>
<tr>
<td>NLP</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Preview of the Results

- very fast linear-time dynamic programming parser
- best reported dependency accuracy on PTB/CTB
- explores *exponentially* many trees (and outputs forest)
Preview of the Results

- very fast linear-time dynamic programming parser
- best reported dependency accuracy on PTB/CTB
- explores \textit{exponentially} many trees (and outputs forest)
Preview of the Results

- very fast linear-time dynamic programming parser
- best reported dependency accuracy on PTB/CTB
- explores \textit{exponentially} many trees (and outputs forest)
Outline

• Motivation

• Incremental (Shift-Reduce) Parsing

• Dynamic Programming for Incremental Parsing

• Experiments
### Shift-Reduce Parsing

I feed cats nearby in the garden.

<table>
<thead>
<tr>
<th>action</th>
<th>stack</th>
<th>queue</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>-</td>
<td>I feed cats ...</td>
</tr>
</tbody>
</table>
I feed cats nearby in the garden.

<table>
<thead>
<tr>
<th>action</th>
<th>stack</th>
<th>queue</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>-</td>
<td>I feed cats ...</td>
</tr>
<tr>
<td>1</td>
<td>shift</td>
<td>feed cats nearby ...</td>
</tr>
</tbody>
</table>
I feed cats nearby in the garden.

<table>
<thead>
<tr>
<th>action</th>
<th>stack</th>
<th>queue</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>-</td>
<td>I feed cats ...</td>
</tr>
<tr>
<td>1</td>
<td>shift</td>
<td>feed cats nearby ...</td>
</tr>
<tr>
<td>2</td>
<td>shift</td>
<td>cats nearby in ...</td>
</tr>
</tbody>
</table>
I feed cats nearby in the garden.

<table>
<thead>
<tr>
<th>action</th>
<th>stack</th>
<th>queue</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>-</td>
<td>I feed cats ...</td>
</tr>
<tr>
<td>1</td>
<td>shift</td>
<td>feed cats nearby ...</td>
</tr>
<tr>
<td>2</td>
<td>shift</td>
<td>cats nearby in ...</td>
</tr>
<tr>
<td>3</td>
<td>l-reduce</td>
<td>cats nearby in ...</td>
</tr>
</tbody>
</table>
I feed cats nearby in the garden.

<table>
<thead>
<tr>
<th>action</th>
<th>stack</th>
<th>queue</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>-</td>
<td>I feed cats ...</td>
</tr>
<tr>
<td>1</td>
<td>shift</td>
<td>feed cats nearby ...</td>
</tr>
<tr>
<td>2</td>
<td>shift</td>
<td>cats nearby in ...</td>
</tr>
<tr>
<td>3</td>
<td>l-reduce</td>
<td>cats nearby in ...</td>
</tr>
<tr>
<td>4</td>
<td>shift</td>
<td>nearby in the ...</td>
</tr>
</tbody>
</table>
I feed cats nearby in the garden.

<table>
<thead>
<tr>
<th>action</th>
<th>stack</th>
<th>queue</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>-</td>
<td>I feed cats ...</td>
</tr>
<tr>
<td>1</td>
<td>shift</td>
<td>feed cats nearby ...</td>
</tr>
<tr>
<td>2</td>
<td>shift</td>
<td>cats nearby in ...</td>
</tr>
<tr>
<td>3</td>
<td>l-reduce</td>
<td>feed</td>
</tr>
<tr>
<td>4</td>
<td>shift</td>
<td>cats nearby in ...</td>
</tr>
<tr>
<td>5a</td>
<td>r-reduce</td>
<td>feed</td>
</tr>
<tr>
<td></td>
<td></td>
<td>nearby in the ...</td>
</tr>
</tbody>
</table>
I feed cats nearby in the garden.

<table>
<thead>
<tr>
<th>action</th>
<th>stack</th>
<th>queue</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>-</td>
<td>I feed cats ...</td>
</tr>
<tr>
<td>1</td>
<td>shift</td>
<td>feed cats nearby ...</td>
</tr>
<tr>
<td>2</td>
<td>shift</td>
<td>cats nearby in ...</td>
</tr>
<tr>
<td>3</td>
<td>l-reduce</td>
<td>cats nearby in ...</td>
</tr>
<tr>
<td>4</td>
<td>shift</td>
<td>nearby in the ...</td>
</tr>
<tr>
<td>5a</td>
<td>r-reduce</td>
<td>nearby in the ...</td>
</tr>
<tr>
<td>5b</td>
<td>shift</td>
<td>in the garden ...</td>
</tr>
</tbody>
</table>

DP for Incremental Parsing
I feed cats nearby in the garden.

<table>
<thead>
<tr>
<th>action</th>
<th>stack</th>
<th>queue</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>-</td>
<td>I feed cats ...</td>
</tr>
<tr>
<td>1</td>
<td>shift</td>
<td>feed cats nearby ...</td>
</tr>
<tr>
<td>2</td>
<td>shift</td>
<td>cats nearby in ...</td>
</tr>
<tr>
<td>3</td>
<td>l-reduce</td>
<td>cats nearby in ...</td>
</tr>
<tr>
<td>4</td>
<td>shift</td>
<td>nearby in the ...</td>
</tr>
<tr>
<td>5a</td>
<td>r-reduce</td>
<td>nearby in the ...</td>
</tr>
<tr>
<td>5b</td>
<td>shift</td>
<td>in the garden ...</td>
</tr>
</tbody>
</table>

**shift-reduce conflict**
Choosing Parser Actions

• score each action using features $\mathbf{f}$ and weights $\mathbf{w}$
  • features are drawn from a local window
  • abstraction (or signature) of a state -- this inspires DP!
• weights trained by structured perceptron (Collins 02)

features:
$$(s_0.w, s_0.rc, q_0, ...) = (\text{cats, nearby, in, ...})$$
Greedy Search

- each state => three new states (shift, l-reduce, r-reduce)
- search space should be exponential
- greedy search: always pick the best next state
Greedy Search

- each state => three new states (shift, l-reduce, r-reduce)
- search space *should* be exponential
- greedy search: always pick the best next state
each state $\Rightarrow$ three new states (shift, l-reduce, r-reduce)
- search space *should* be exponential
- beam search: always keep top-$b$ states
Dynamic Programming

- each state => three new states (shift, l-reduce, r-reduce)
- key idea of DP: share common subproblems
  - merge equivalent states => polynomial space
Dynamic Programming

- each state => three new states (shift, l-reduce, r-reduce)
- key idea of DP: share common subproblems
  - merge equivalent states => polynomial space

“graph-structured stack” (Tomita, 1988)
Dynamic Programming

- each state => three new states (shift, l-reduce, r-reduce)
- key idea of DP: share common subproblems
  - merge equivalent states => polynomial space

“graph-structured stack” (Tomita, 1988)
Dynamic Programming

- each state => three new states (shift, l-reduce, r-reduce)
- key idea of DP: share common subproblems
  - merge equivalent states => polynomial space

Each DP state corresponds to exponentially many non-DP states

“graph-structured stack” (Tomita, 1988)
Dynamic Programming

- each state $\Rightarrow$ three new states (shift, l-reduce, r-reduce)
- key idea of DP: share common subproblems
  - merge equivalent states $\Rightarrow$ polynomial space

```
Each DP state corresponds to exponentially many non-DP states
```

“graph-structured stack” (Tomita, 1988)
two states are equivalent if they agree on features

because same features guarantee same cost

shift-reduce conflict:

... feed cats nearby in the garden

... feed re feed sh ... cats
Merging Equivalent States

- Two states are equivalent if they agree on features.
- Because same features guarantee same cost.

Shift-reduce conflict:

- Assume features only look at root of $s_0$
Merging Equivalent States

- two states are equivalent if they agree on features
- because same features guarantee same cost

shift-reduce conflict:

 assumes features only look at root of $s_0$

two states are equivalent if they agree on root of $s_0$
Merging Equivalent States

- two states are equivalent if they agree on features
  - because same features guarantee same cost

- shift-reduce conflict:

```
feed cats nearby in the garden
```

- assume features only
  - look at root of \( s_0 \)

- two states are equivalent if they agree on root of \( s_0 \)
Merging Equivalent States

- two states are equivalent if they agree on features
- because same features guarantee same cost

shift-reduce conflict:

- **feed cats nearby** in the garden
- **feed** nearby in the garden
Merging Equivalent States

- two states are equivalent if they agree on features
  - because same features guarantee same cost

- shift-reduce conflict:
  - **feed cats nearby** in the garden
  - **feed** nearby in the garden
Merging Equivalent States

- two states are equivalent if they agree on features
- because same features guarantee same cost

shift-reduce conflict:

- `feed cats` in the garden
- `feed` `nearby` in the garden

... `nearby` ... → `... cats`

... `... feed` ... → `... nearby`

... `... cats` ... → `... nearby`
Merging Equivalent States

- two states are equivalent if they agree on features
- because same features guarantee same cost

- shift-reduce conflict:

  - feed
    - in the garden
      - ... cats
  - feed
    - in the garden
      - ... feed
Merging Equivalent States

- two states are equivalent if they agree on features
  - because same features guarantee same cost

shift-reduce conflict:

- \( \text{feed in the garden} \)
- \( \text{cats} \)
- \( \text{nearby} \)

- \( \text{feed in the garden} \)
- \( \text{cats} \)
- \( \text{nearby} \)
- \( \text{feed} \)

\( \text{stack} \rightarrow \text{queue} \)

\( q_0 \) \( q_1 \) \( \ldots \)
Merging Equivalent States

- two states are equivalent if they agree on features
  - because same features guarantee same cost

- shift-reduce conflict:

  - feed
    - cats
    - nearby

  - feed
    - cats
    - nearby

in the garden

in the garden

DP for Incremental Parsing
Merging Equivalent States

- two states are equivalent if they agree on features
- because same features guarantee same cost

shift-reduce conflict:

- feed

in the garden

- feed

in the garden

(local) ambiguity-packing!
Merging Equivalent States

- two states are equivalent if they agree on features
- because same features guarantee same cost

- shift-reduce conflict:

```
I           cats
nearby
feed in the garden
```

```
I          cats nearby
feed in the garden
```

```
... s2 s1 s0
q0 q1 ...
```

→ stack  queue ←
Merging Equivalent States

- two states are equivalent if they agree on features
- because same features guarantee same cost

- shift-reduce conflict:

```
I           cats
nearby

→ stack  queue ←

... S2  S1  S0  q0  q1 ...

graph-structured stack
```
Theory: Polynomial-Time DP

- this DP is exact and \textit{polynomial-time} if features are:
  
  - a) \textbf{bounded} -- for polynomial time  
    - features can only look at a \textit{local window}
  
  - b) \textbf{monotonic} -- for correctness (optimal substructure) 
    - features should draw no more info from trees farther away from stack top than from trees closer to top

  - both are intuitive: a) always true; b) \textit{almost} always true
Theory: Monotonic History

- related: grammar refinement by annotation (Johnson, 1998)
- annotate vertical context history (e.g., parent)
- monotonicity: can’t annotate grand-parent without annotating the parent (otherwise DP would fail)
- our features: left-context history instead of vertical-context
  - similarly, can’t annotate $s_2$ without annotating $s_1$
  - but we can always design “minimum monotonic superset”
Related Work

- Graph-Structured Stack (Tomita 88): Generalized LR
  - GSS is just a chart viewed from left to right (e.g. Earley 70)
  - this line of work started w/ Lang (1974); stuck since 1990
  - b/c explicit LR table is impossible with modern grammars
- Jelinek (2004) independently rediscovered GSS
Related Work

- Graph-Structured Stack (Tomita 88): Generalized LR
  - GSS is just a chart viewed from left to right (e.g. Earley 70)
  - this line of work started w/ Lang (1974); stuck since 1990
  - b/c explicit LR table is impossible with modern grammars
  - Jelinek (2004) independently rediscovered GSS

- We revived and advanced this line of work in two aspects
  - theoretical: implicit LR table based on features
    - merge and split on-the-fly; no pre-compilation needed
    - monotonic feature functions guarantee correctness (new)
  - practical: achieved linear-time performance with pruning
STOCHASTIC ANALYSIS OF STRUCTURED LANGUAGE MODELING

FREDERICK JELINEK*

Abstract. As previously introduced, the Structured Language Model (SLM) operated with the help of a stack from which less probable sub-parse entries were purged before further words were generated. In this article we generalize the CKY algorithm to obtain a chart which allows the direct computation of language model probabilities thus rendering the stacks unnecessary. An analysis of the behavior of the SLM leads to a generalization of the Inside – Outside algorithm and thus to rigorous EM type re-estimation of the SLM parameters. The derived algorithms are computationally expensive but their demands can be mitigated by use of appropriate thresholding.

1. Introduction. The structured language model (SLM) was developed to allow a speech recognizer to assign a priori probabilities to words and do so based on a wider past context than is available to the state-of-the-art trigram language model. It is then not surprising that the use of the SLM results in lower perplexities and lower error probabilities [1, 2].

* In: M. Johnson, S. Khudanpur, M. Ostendorf, and R. Rosenfeld (eds.): Mathematical Foundations of Speech and Language Processing, 2004
STOCHASTIC ANALYSIS OF STRUCTURED LANGUAGE MODELING
FREDERICK JELINEK*

Abstract. As previously introduced, the Structured Language Model (SLM) operated with the help of a stack from which less probable sub-parse entries were purged before further words were generated. In this article we generalize the CKY algorithm to obtain a chart which allows the direct computation of language model probabilities thus rendering the stacks unnecessary. An analysis of the behavior of the SLM leads to a generalization of the Inside–Outside algorithm and thus to rigorous EM type re-estimation of the SLM parameters. The derived algorithms are computationally expensive but their demands can be mitigated by use of appropriate thresholding.

1. Introduction. The structured language model (SLM) was developed to allow a speech recognizer to assign a priori probabilities to words and do so based on a wider past context than is available to the state-of-the-art trigram language model. It is then not surprising that the use of the SLM results in lower perplexities and lower error probabilities [1, 2].

In: M. Johnson, S. Khudanpur, M. Ostendorf, and R. Rosenfeld (eds.): Mathematical Foundations of Speech and Language Processing, 2004
Abstract. As previously introduced, the Structured Language Model (SLM) operated with the help of a stack from which less probable sub-parse entries were purged before further words were generated. In this article we generalize the CKY algorithm to obtain a chart which allows the direct computation of language model probabilities thus rendering the stacks unnecessary. An analysis of the behavior of the SLM leads to a generalization of the Inside - Outside algorithm and thus to rigorous EM type re-estimation of the SLM parameters. The derived algorithms are computationally expensive but their demands can be mitigated by use of appropriate thresholding.

1. Introduction. The structured language model (SLM) was developed to allow a speech recognizer to assign a priori probabilities to words and do so based on a wider past context than is available to the state-of-the-art trigram language model. It is then not surprising that the use of the SLM results in lower perplexities and lower error probabilities [1, 2].

In: M. Johnson, S. Khudanpur, M. Ostendorf, and R. Rosenfeld (eds.): Mathematical Foundations of Speech and Language Processing, 2004
Jelinek (2004)

• structured language model as graph-structured stack

see also (Chelba and Jelinek, 98; 00; Xu, Chelba, Jelinek, 02)
Jelinek (2004)

structured language model as graph-structured stack

$P_{SLM}(a | \text{has, show})$

$P_{3gram}(a | \text{its, host})$

see also (Chelba and Jelinek, 98; 00; Xu, Chelba, Jelinek, 02)
Experiments
**Speed Comparison**

- 5 times faster with the same parsing accuracy
Correlation of Search and Parsing

- better search quality $\iff$ better parsing accuracy
Search Space: Exponential

DP: exponential
non-DP: fixed (beam-width)

number of trees explored

sentence length

DP for Incremental Parsing
N-Best / Forest Oracles

(b) oracle precision on dev

- DP forest oracle (98.15)
- DP $k$-best in forest
- non-DP $k$-best in beam
Better Search => Better Learning

- DP leads to faster and better learning w/ perceptron
Learning Details: Early Updates

- greedy search: update at first error
- beam search: update when gold is pruned (Collins/Roark 04)
- DP search: also update when gold is “merged” (new!)
  - b/c we know gold can’t make to the top again

<table>
<thead>
<tr>
<th>it</th>
<th>updates</th>
<th>early%</th>
<th>time</th>
<th>updates</th>
<th>early%</th>
<th>time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>31943</td>
<td>98.9</td>
<td>22</td>
<td>31189</td>
<td>87.7</td>
<td>29</td>
</tr>
<tr>
<td>2</td>
<td>27311</td>
<td>98.8</td>
<td>29</td>
<td>26324</td>
<td>80.9</td>
<td>37</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>5</td>
<td>20236</td>
<td>98.3</td>
<td>38</td>
<td>19027</td>
<td>70.3</td>
<td>47</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>17</td>
<td>8683</td>
<td>97.1</td>
<td>48</td>
<td>7434</td>
<td>49.5</td>
<td>60</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>25</td>
<td>5715</td>
<td>97.2</td>
<td>51</td>
<td>4676</td>
<td>41.2</td>
<td>65</td>
</tr>
</tbody>
</table>
Parsing Time vs. Sentence Length

- parsing speed (scatter plot) compared to other parsers

![Graph showing parsing time vs. sentence length]
Parsimg Time vs. Sentence Length

- parsing speed (scatter plot) compared to other parsers
• parsing speed (scatter plot) compared to other parsers

Parsing Time vs. Sentence Length

- Parsing time (secs) compared to other parsers.

Charniak: $O(n^2)$
Berkeley: $O(n^{2.4})$
MST: $O(n^{2.5})$
This work: $O(n)$
Final Results

- much faster than major parsers (even with Python!)
- first linear-time incremental dynamic programming parser
- best reported dependency accuracy on Penn Treebank

<table>
<thead>
<tr>
<th>Time</th>
<th>Complexity</th>
<th>Trees Searched</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.12</td>
<td>$O(n^2)$</td>
<td>exponential</td>
</tr>
<tr>
<td>-</td>
<td>$O(n^4)$</td>
<td>exponential</td>
</tr>
<tr>
<td>0.11</td>
<td>$O(n)$</td>
<td>constant</td>
</tr>
<tr>
<td>0.04</td>
<td>$O(n)$</td>
<td>exponential</td>
</tr>
<tr>
<td>0.49</td>
<td>$O(n^{2.5})$</td>
<td>exponential</td>
</tr>
<tr>
<td>0.21</td>
<td>$O(n^{2.4})$</td>
<td>exponential</td>
</tr>
</tbody>
</table>

McDonald et al 05 - MST: 90.2
Koo et al 08 baseline*: 92.0
Zhang & Clark 08 single: 91.4
this work: 92.1
Charriak 00: 92.5
Petrov & Klein 07: 92.4
Final Results

- much faster than major parsers (even with Python!)
- first linear-time incremental dynamic programming parser
- best reported dependency accuracy on Penn Treebank

<table>
<thead>
<tr>
<th></th>
<th>89</th>
<th>91</th>
<th>93</th>
</tr>
</thead>
<tbody>
<tr>
<td>McDonald et al 05 - MST</td>
<td>90.2</td>
<td>92.0</td>
<td></td>
</tr>
<tr>
<td>Koo et al 08 baseline*</td>
<td></td>
<td>92.0</td>
<td></td>
</tr>
<tr>
<td>Zhang &amp; Clark 08 single</td>
<td>91.4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>this work</td>
<td></td>
<td>92.1</td>
<td></td>
</tr>
<tr>
<td>Charniak 00</td>
<td></td>
<td>92.5</td>
<td></td>
</tr>
<tr>
<td>Petrov &amp; Klein 07</td>
<td></td>
<td>92.4</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>time</th>
<th>complexity</th>
<th>trees searched</th>
</tr>
</thead>
<tbody>
<tr>
<td>McDonald et al 05 - MST</td>
<td>0.12</td>
<td>$O(n^2)$</td>
<td>exponential</td>
</tr>
<tr>
<td>Koo et al 08 baseline*</td>
<td>-</td>
<td>$O(n^4)$</td>
<td>exponential</td>
</tr>
<tr>
<td>Zhang &amp; Clark 08 single</td>
<td>0.11</td>
<td>$O(n)$</td>
<td>constant</td>
</tr>
<tr>
<td>this work</td>
<td>0.04</td>
<td>$O(n)$</td>
<td>exponential</td>
</tr>
<tr>
<td>Charniak 00</td>
<td>0.49</td>
<td>$O(n^{2.5})$</td>
<td>exponential</td>
</tr>
<tr>
<td>Petrov &amp; Klein 07</td>
<td>0.21</td>
<td>$O(n^{2.4})$</td>
<td>exponential</td>
</tr>
</tbody>
</table>
Final Results

- much faster than major parsers (even with Python!)
- first linear-time incremental dynamic programming parser
- best reported dependency accuracy on Penn Treebank

<table>
<thead>
<tr>
<th></th>
<th>time</th>
<th>complexity</th>
<th>trees searched</th>
</tr>
</thead>
<tbody>
<tr>
<td>McDonald et al 05 - MST</td>
<td>0.12</td>
<td>$O(n^2)$</td>
<td>exponential</td>
</tr>
<tr>
<td>Koo et al 08 baseline*</td>
<td>-</td>
<td>$O(n^4)$</td>
<td>exponential</td>
</tr>
<tr>
<td>Zhang &amp; Clark 08 single</td>
<td>0.11</td>
<td>$O(n)$</td>
<td>constant</td>
</tr>
<tr>
<td>this work</td>
<td>0.04</td>
<td>$O(n)$</td>
<td>exponential</td>
</tr>
<tr>
<td>Charniak 00</td>
<td>0.49</td>
<td>$O(n^{2.5})$</td>
<td>exponential</td>
</tr>
<tr>
<td>Petrov &amp; Klein 07</td>
<td>0.21</td>
<td>$O(n^{2.4})$</td>
<td>exponential</td>
</tr>
</tbody>
</table>

*at this ACL: Koo & Collins 10: 93.0 with $O(n^4)$
Final Results on Chinese

- also the best parsing accuracy on Chinese
- Penn Chinese Treebank (CTB 5)
- all numbers below use gold-standard POS tags

```
<table>
<thead>
<tr>
<th></th>
<th>Duan et al. 2007</th>
<th>Zhang &amp; Clark 08 (single)</th>
<th>this work</th>
</tr>
</thead>
<tbody>
<tr>
<td>word</td>
<td>73.7</td>
<td>76.7</td>
<td>78.3</td>
</tr>
<tr>
<td>non-root</td>
<td>84.4</td>
<td>84.7</td>
<td>85.2</td>
</tr>
<tr>
<td>root</td>
<td>83.9</td>
<td>84.3</td>
<td>85.5</td>
</tr>
</tbody>
</table>
```

DP for Incremental Parsing
## Conclusion

<table>
<thead>
<tr>
<th>Greedy Search</th>
<th>Incremental Parsing (e.g. shift-reduce)</th>
<th>Full Dynamic Programming (e.g. CKY)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Principled Search</td>
<td>☺ ✓</td>
<td>Fast (linear-time)</td>
</tr>
<tr>
<td>Fast (linear-time)</td>
<td></td>
<td>Slow (cubic-time)</td>
</tr>
</tbody>
</table>
## Conclusion

<table>
<thead>
<tr>
<th>Greedy Search</th>
<th>Incremental Parsing (e.g. shift-reduce)</th>
<th>Full Dynamic Programming (e.g. CKY)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Principled Search</td>
<td>Linear-time shift-reduce parsing w/ dynamic programming</td>
<td>Fast (linear-time)</td>
</tr>
</tbody>
</table>
### Zoom out to Big Picture...

<table>
<thead>
<tr>
<th>Human</th>
<th>Natural Languages</th>
<th>Programming Languages</th>
</tr>
</thead>
<tbody>
<tr>
<td>Psycholinguistics</td>
<td>☺</td>
<td>☹</td>
</tr>
<tr>
<td>NLP</td>
<td>☼</td>
<td>☻</td>
</tr>
</tbody>
</table>

Still a long way to go...
Thank You

- a general theory of DP for shift-reduce parsing
  - as long as features are bounded and monotonic
- fast, accurate DP parser release coming soon:
  - http://www.isi.edu/~lhuang

- future work
  - adapt to constituency parsing (straightforward)
  - other grammar formalisms like CCG and TAG
  - integrate POS tagging into the parser
  - integrate semantic interpretation
How I was invited to give this talk

- Fred attended ACL 2010 in Sweden
- Mark Johnson mentioned to him about this work
- Fred saw my co-author Kenji Sagae giving the talk
- but didn’t realize it was Kenji; he thought it was me
- he emailed me (but mis-spelled my name in the address)
- not getting a reply, he asked Kevin Knight to “forward it to Liang Haung or his student Sagae.”
- Fred complained that my paper is very hard to read “As you can see, I am completely confused!” And he was right.
- finally he said “come here to give a talk and explain it.”