Dynamic Programming for Linear Time Incremental Parsing

Liang Huang
Information Sciences Institute
University of Southern California

Kenji Sagae
Institute for Creative Technologies
University of Southern California

ACL 2010, Uppsala, Sweden, July 2010 (slightly expanded)
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
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)
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>This work:</td>
<td>Fast shift-reduce parsing with dynamic programming</td>
<td></td>
</tr>
<tr>
<td>Fast (linear-time)</td>
<td></td>
<td>Slow (cubic-time)</td>
</tr>
</tbody>
</table>

DP for Incremental Parsing (Huang and Sagae)
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 *exponentially* many trees (and outputs forest)

![Graph showing parsing time vs. sentence length对比图]
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)

![Graph showing parsing time vs sentence length for various methods including DP (exponential), non-DP beam search, and this work (Berkeley and Charniak).]
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>

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>
<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 shift</td>
<td>I</td>
<td>feed cats nearby ...</td>
</tr>
<tr>
<td>2 shift</td>
<td>I feed</td>
<td>cats nearby in ...</td>
</tr>
<tr>
<td>3 l-reduce</td>
<td>feed</td>
<td>cats nearby in ...</td>
</tr>
</tbody>
</table>
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>
<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.
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>
</tbody>
</table>
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>
<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>
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 \( f \) and weights \( w \)
- features are drawn from a local window
- abstraction (or signature) of a state -- this inspires DP!
- weights trained by structured perceptron (Collins 02)

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

- each state => 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)

DP for Incremental Parsing (Huang and Sagae)
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

“graph-structured stack” (Tomita, 1988)

DP for Incremental Parsing (Huang and Sagae)
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)

DP for Incremental Parsing (Huang and Sagae)
Dynamic Programming

- Each state leads to three new states (shift, left-reduce, right-reduce).
- Key idea of DP: share common subproblems.
- Merge equivalent states to achieve polynomial space.

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

“Graph-structured stack” (Tomita, 1988)

DP for Incremental Parsing (Huang and Sagae)
Merging Equivalent States

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

- shift-reduce conflict:

```
... S2 S1 S0
```

```
... stack queue ...
```

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

```
... sh ... feed re feed sh ... cats
```

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

```
... cats re feed ...
```

DP for Incremental Parsing (Huang and Sagae)
Merging Equivalent States

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

Shift-reduce conflict:
- Assure features only look at root of $s_0$. 

Example: feed cats 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:

- 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:

- \text{feed cats} \hspace{1cm} \text{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
  • feed nearby in the garden

DP for Incremental Parsing (Huang and Sagae)
Merging Equivalent States

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

Shift-reduce conflict:

- \text{feed, cats}
- \text{nearby in the garden}
- \text{in the garden}
- \text{feed, nearby}
Merging Equivalent States

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

shift-reduce conflict:

- **feed**

  "feed in the garden"

- **feed**

  "feed 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
    in the garden
  • feed
    in the garden

  ... cats    nearby
  ... feed    ... cats    ... feed
  ... nearby  ... feed    ... nearby
  ... cats    ... feed    ... feed

  left stack  queue right
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
  sh ... nearby
  re ... cats
  re

  ... feed
  sh ... nearby
  re

- feed

  in the garden

  ... cats
  re

  ... feed
  re

  ... nearby
  re

DP for Incremental Parsing (Huang and Sagae)
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

(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
```

```
... S2 S1 S0
q0 q1 ...
← stack queue →
```

```
... nearby re ... cats re ... feed
... feed sh ... nearby re ...
```

```
... cats sh ... nearby re ...
```

```
... in
```

```
... s2 s1 s0
q0 q1 ...
← stack queue →
```

```
... nearby re ... cats re ... feed
... feed sh ... nearby re ...
```

```
... cats sh ... nearby re ...
```

```
... in
```

```
... s2 s1 s0
q0 q1 ...
← stack queue →
```

```
... nearby re ... cats re ... feed
... feed sh ... nearby re ...
```

```
... cats sh ... nearby re ...
```

```
... in
```

```
... s2 s1 s0
q0 q1 ...
← stack queue →
```

```
... nearby re ... cats re ... feed
... feed sh ... nearby re ...
```

```
... cats sh ... nearby re ...
```

```
... in
```
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

graph-structured stack

DP for Incremental Parsing (Huang and Sagae)
• this DP is exact and polynomial-time if features are:
  
  • a) bounded -- for polynomial time
    
    ● features can only look at a local window
  
  • b) 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) 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”

![Diagram showing DP for Incremental Parsing (Huang and Sagae)]
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
  - general idea: compile CFG parse chart to FSAs (e.g. our beam)
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
  - general idea: compile CFG parse chart to FSAs (e.g. our beam)
- 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
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)
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

DP for Incremental Parsing (Huang and Sagae)
Better Search => Better Learning

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

- greedy search: update at first error (Collins/Roark 04)
- beam search: update when gold is pruned (Zhang/Clark 08)
- 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>
Pars 域 Time vs. Sentence Length

- parsing speed (scatter plot) compared to other parsers
Parsig Time vs. Sentence Length

- Parsing speed (scatter plot) compared to other parsers

![Graph showing the relationship between parsing time and sentence length for different parsers.](image-url)
• parsing speed (scatter plot) compared to other parsers
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>
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></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>92.4</td>
<td></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>Word</th>
<th>Non-root</th>
<th>Root</th>
</tr>
</thead>
<tbody>
<tr>
<td>Duan et al. 2007</td>
<td></td>
<td></td>
<td>83.9</td>
</tr>
<tr>
<td></td>
<td>73.7</td>
<td></td>
<td>84.4</td>
</tr>
<tr>
<td>Zhang &amp; Clark 08 (single)</td>
<td></td>
<td></td>
<td>84.3</td>
</tr>
<tr>
<td></td>
<td>76.7</td>
<td></td>
<td>84.7</td>
</tr>
<tr>
<td>this work</td>
<td></td>
<td></td>
<td>85.2</td>
</tr>
<tr>
<td></td>
<td>78.3</td>
<td></td>
<td>85.5</td>
</tr>
</tbody>
</table>

DP for Incremental Parsing (Huang and Sagae)
## 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>
</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>
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](http://www.isi.edu/~lhuang)
  - [http://www.ict.usc.edu/~sagae](http://www.ict.usc.edu/~sagae)
- future work
  - adapt to constituency parsing (straightforward)
  - other grammar formalisms like CCG and TAG
  - integrate POS tagging into the parser