Hierarchical Search for Parsing

Adam Pauls and Dan Klein
Motivation

- Modern parsers users very large grammars (millions of rules!)

- Coarse-to-Fine has proven successful (Charniak and Caraballo 1998)

- Multi-level or Hierarchical Coarse-to-Fine works even better (Charniak and Johnson 2005, Petrov and Klein 2007)

- In this talk, we explore an optimal hierarchical search method: *Hierarchical A*
Hierarchical Setting

fast parsing is fun.
fast parsing is fun.
Hierarchical Setting

fast parsing is fun .

Fine Grammar

S
  /  
NP    VP
  /  
JJ    NN VBZ JJ .
Hierarchical Setting

fast parsing is fun.
Hierarchical Setting

Fast parsing is fun.

Coarser Grammar

Coarse Grammar

Fine Grammar

Fast parsing is fun.
Hierarchical Setting

Coarse Grammar

Fine Grammar

fast parsing is fun.
Hierarchical Setting

Coarse Grammar

Coarser Grammar

Fine Grammar

fast parsing is fun .
Agenda-Based Search
Agenda-Based Search

“edges”

VP[2,4]
S[0,2]

Agenda
Agenda-Based Search

“edges”

VP[2,4]
S[0,2]

Agenda

Chart
Agenda-Based Search

“edges”

S[0,2]
.
.
.

“Agenda”

VP[2,4]

NP

VP

S

S

Chart

VB

JJ

NP

VB

NN

VB

NN

VB

NN


Agenda-Based Search

“edges”

S[0,2]

NP[0,2]  VP[2,4]
Agenda-Based Search

“edges”

Agenda

Chart

S[0,2]

NP[0,2] VP[2,4]
Agenda-Based Search

“edges”

S[0,4]

S

NP VP

NP[0,2] VP[2,4]

Agenda

Chart
Agenda-Based Search

“edges”

S[0,2]
S[0,4]

... 

Agenda

Chart
Agenda-Based Search

```
“edges”
```

```
S[0,2]
S[0,4]
```

```
Agenda
```

```
Chart
```

```
VB
JJ
NP
NN
VP
VB
NP
```

```
S
S
```

```
VP
VP
```

```
```

Building Edges

[Diagram showing two triangles, one labeled NP with vertices 1 and 3, and the other labeled VP with vertices 3 and 5.]
Building Edges

\[ \beta_L \]
\[ \beta_R \]
Building Edges

NP 1 3  \( \beta_L \)

S

NP 3 5  \( \beta_R \)

VP

w

NP

VP
Building Edges

S

NP

NP

$\beta_L$

1 3

VP

VP

$\beta_R$

3 5

$\mathcal{W}$
Building Edges

\[
\begin{array}{c}
S \\
NP \quad VP \\
\beta_L \quad \beta_R
\end{array}
\]

\[
\begin{array}{c}
1 \\
3 \\
3 \\
5 \\
1 \\
5
\end{array}
\]
Building Edges

\[ \beta = \beta_L + \beta_R + w \]
Building Edges

priority:

\[ \beta = \beta_L + \beta_R + w \]
Building Edges

priority:

$$\beta = \beta_L + \beta_R + w$$

Uniform Cost Search
Building Edges

priority:

$$\beta + h(S[1,5])$$

$$A^*$$

$$= \beta_L + \beta_R + w$$
Heuristics

- $h$ is a heuristic which lower bounds the Viterbi outside cost $\alpha$
Heuristics

- $h$ is a heuristic which lower bounds the Viterbi outside cost $\alpha$

$$h(\text{VP}[2,4]) \leq \alpha(\text{VP}[2,4])$$
Heuristics

- $h$ is a heuristic which lower bounds the Viterbi outside cost $\alpha$

$$h(\text{VP}[2,4]) \leq \alpha(\text{VP}[2,4])$$

fast parsing is fun.
• $h$ is a heuristic which lower bounds the Viterbi outside cost $\alpha$

$$h(\text{VP}[2,4]) \leq \alpha(\text{VP}[2,4])$$
Heuristics

- \( h \) is a heuristic which lower bounds the Viterbi outside cost \( \alpha \)

\[
h(\text{VP}[2,4]) \leq \alpha(\text{VP}[2,4])
\]

\text{fast parsing} \quad \text{is} \quad \text{fun}
• $h$ is a heuristic which lower bounds the Viterbi outside cost $\alpha$

$$h(\text{VP}[2,4]) \leq \alpha(\text{VP}[2,4])$$
Heuristics

- $h$ is a heuristic which lower bounds the Viterbi outside cost $\alpha$

$$h(\text{VP}[2,4]) \leq \alpha(\text{VP}[2,4])$$
Heuristics

• $h$ is a heuristic which lower bounds the Viterbi outside cost $\alpha$

$$h(\text{VP}[2,4]) \leq \alpha(\text{VP}[2,4])$$
Heuristics

- $h$ is a heuristic which lower bounds the Viterbi outside cost $\alpha$

\[
h(\text{VP}[2,4]) \leq \alpha(\text{VP}[2,4])
\]
Outside Scores

- We can get lower bounds on $\alpha$ from coarse grammars
Outside Scores

- We can get lower bounds on $\alpha$ from coarse grammars
• We can get lower bounds on $\alpha$ from coarse grammars
Outside Scores

- We can get lower bounds on $\alpha$ from coarse grammars.

- How do we compute these outside scores?
Building Outside Edges

\[ \alpha \]

\[ VP \]

0 3 5 n
Building Outside Edges

\[ \alpha \]

\[ VP \]

0 \hspace{1cm} 3 \hspace{1cm} 5 \hspace{1cm} n
Building Outside Edges

\[ V \alpha \]

\[ \alpha_0 \]

\[ S \]

\[ 0 \quad 1 \quad 5 \quad n \]

\[ \alpha \]

\[ VP \]

\[ 0 \quad 3 \quad 5 \quad n \]
Building Outside Edges

\[
\alpha_0 \quad S \quad 5 \quad n
\]

\[
\alpha \\
\text{VP}
\]

\[
0 \quad 3 \quad 5 \quad n
\]
Building Outside Edges

\[ \alpha \]

\[ \beta \]

\[ \omega \]

\[ \text{NP} \]

\[ \text{S} \]

\[ \text{VP} \]

\[ \text{NP}[1,3] \]
Building Outside Edges

\[ \alpha = \alpha_O + w + \beta_L \]
Hierarchical A* (Felzenswalb and McAllester 2007)

- Basic Idea:
  - build both inside and outside edges as needed using same agenda
  - use coarse outside scores as heuristics for fine inside edges
Hierarchically Building Inside Edges

\[ \beta + h(S[1,5]) = \beta_L + \beta_R + w \]
Hierarchically Building Inside Edges

\[ \beta + h(S[1,5]) \]
Hierarchically Building Inside Edges

\[ \beta + h(S[1,5]) \]

\[ \beta = \beta_L + \beta_R + w \]
Hierarchically Building Inside Edges

\[ S \]

\[ \alpha' \]

\[ NP \] \[ VP \]

\[ \beta_L \] \[ \beta_R \]

\[ \beta = \beta_L + \beta_R + w \]
HA*
HA*

inside

Agenda

Charts
HA*

inside

outside

Agenda

Charts
HA*

inside

outside

coarser

Agenda

Charts
HA*

inside

outside

coarser

Agenda

Charts
HA*
HA*
Coarse-to-Fine

- Prune edges in fine grammar based on posteriors from coarse grammar
Coarse-to-Fine

• Prune edges in fine grammar based on posteriors from coarse grammar

• We use Viterbi posteriors for pruning (Petrov and Klein 2007)
Coarse-to-Fine

- Prune edges in fine grammar based on posteriors from coarse grammar

- We use Viterbi posteriors for pruning (Petrov and Klein 2007)

\[ \beta'(e) + \alpha'(e) \leq \text{threshold} \]
Agenda-Based CTF

- (Hierarchical) CTF can also be thought of as an instance of agenda-based parsing with

\[
\text{priority}(e) = \begin{cases} 
\beta(e) & \beta'(e) + \alpha'(e) \leq \text{threshold} \\
\infty & \text{otherwise}
\end{cases}
\]
Agenda-Based CTF

- (Hierarchical) CTF can also be thought of as an instance of agenda-based parsing with

\[
priority(e) = \begin{cases} 
\beta(e) & \beta'(e) + \alpha'(e) \leq \text{threshold} \\
\infty & \text{otherwise}
\end{cases}
\]

- This reformulation makes architectures directly comparable
HA* vs. HCTF Qualitatively

HA*  HCTF
HA* vs. HCTF Qualitatively

- **HA***  
  - optimal

- **HCTF**  
  - makes search errors
<table>
<thead>
<tr>
<th></th>
<th>HA*</th>
<th>HCTF</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><strong>optimal</strong></td>
<td><strong>makes search errors</strong></td>
</tr>
<tr>
<td></td>
<td><strong>uses coarse grammars</strong></td>
<td><strong>uses coarse grammars</strong></td>
</tr>
<tr>
<td></td>
<td>to prioritize search</td>
<td>to prune search</td>
</tr>
<tr>
<td><strong>HA</strong>*</td>
<td><strong>HCTF</strong></td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td></td>
</tr>
<tr>
<td>‣ optimal</td>
<td>‣ makes search errors</td>
<td></td>
</tr>
<tr>
<td>‣ uses coarse grammars to prioritize search</td>
<td>‣ uses coarse grammars to prune search</td>
<td></td>
</tr>
<tr>
<td>‣ speed determined by tightness of heuristic</td>
<td>‣ speed determined by threshold</td>
<td></td>
</tr>
<tr>
<td>HA*</td>
<td>HCTF</td>
<td></td>
</tr>
<tr>
<td>--------------</td>
<td>----------------</td>
<td></td>
</tr>
<tr>
<td>optimal</td>
<td>makes search errors</td>
<td></td>
</tr>
<tr>
<td>uses coarse grammars to prioritize search</td>
<td>uses coarse grammars to prune search</td>
<td></td>
</tr>
<tr>
<td>speed determined by tightness of heuristic</td>
<td>speed determined by threshold</td>
<td></td>
</tr>
<tr>
<td>min over rules</td>
<td>average over rules</td>
<td></td>
</tr>
</tbody>
</table>
Experimental Setup #1

- Use the state-split grammars of Petrov et al. 2006
- Train on WSJ Sections 2-21, and use 6 split iterations, which creates 7 grammars
Experimental Setup #1

- Use the state-split grammars of Petrov et al. 2006
- Train on WSJ Sections 2-21, and use 6 split iterations, which creates 7 grammars
Experimental Setup #1

- Use the state-split grammars of Petrov et al. 2006
- Train on WSJ Sections 2-21, and use 6 split iterations, which creates 7 grammars
Experimental Setup #1

- Use the state-split grammars of Petrov et al. 2006
- Train on WSJ Sections 2-21, and use 6 split iterations, which creates 7 grammars
Experimental Setup #1

- Use the state-split grammars of Petrov et al. 2006

- Train on WSJ Sections 2-21, and use 6 split iterations, which creates 7 grammars
State-Split Projections

S-26

NP-21  VP-14

6-split
State-Split Projections

S-14

NP-12  VP-6

S-26

NP-21  VP-14

5-split

6-split
State-Split Projections

4-split

5-split

6-split

S-7

NP-5  VP-6

S-14

NP-12  VP-6

S-26

NP-21  VP-14
State-Split Projections

4-split

5-split

6-split
One-Level CTF vs. A*

- Only one coarse grammar (the 3-split)
- CTF is faster than A*, but makes search errors

Exhaustive: 424.00
A*: 86.60
CTF: 8.83

Makes 2% search errors
Hierarchies

- How do HCTF and HA* scale with size of hierarchy?
Hierarchies

- How do HCTF and HA* scale with size of hierarchy?

<table>
<thead>
<tr>
<th></th>
<th>3,6-split</th>
<th>3-6 split</th>
<th>0-6 split</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>8.83</td>
<td>7.12</td>
<td>1.98</td>
</tr>
<tr>
<td>1</td>
<td>9.00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Hierarchies

- How do HCTF and HA* scale with size of hierarchy?
Why A*?

- CTF is faster, and extends to hierarchies nicely, so why A*?

1. If you really don’t want to make search errors
Cost of Optimality: State-Split Grammars

HCTF Speed vs. Search Errors

Edges pushed (billions)

Fraction of sentences without search errors
Cost of Optimality: State-Split Grammars

HCTF Speed vs. Search Errors

Edges pushed (billions)

Fraction of sentences without search errors
Cost of Optimality: State-Split Grammars

HCTF Speed vs. Search Errors

Edges pushed (billions)

Fraction of sentences without search errors

HA*

99%

99%
Why A*?

• CTF is faster, and extends to hierarchies nicely, so why A*?

1. If you really don’t want to make search errors

2. For some problems, we can find very efficient, tight heuristics

• In this case, A* is very fast
Experimental Setup #2

- We use the factored lexicalized grammar of Klein and Manning (2003)

- They construct a lexicalized grammar as the cross-product of a dependency grammar and PCFG
Experimental Setup #2

- We use the factored lexicalized grammar of Klein and Manning (2003)

- They construct a lexicalized grammar as the cross-product of a dependency grammar and PCFG
Experimental Setup #2

• We use the factored lexicalized grammar of Klein and Manning (2003)

• They construct a lexicalized grammar as the cross-product of a dependency grammar and PCFG
Experimental Setup #2

- We use the factored lexicalized grammar of Klein and Manning (2003)
- They construct a lexicalized grammar as the cross-product of a dependency grammar and PCFG

\[ S \]  
\[ W_p \]  
\[ \text{NP} \]  
\[ \text{VP} \]  
\[ \text{PCFG} \]  
\[ \times \]  
\[ \text{is-VB} \]  
\[ \text{is-VB} \]  
\[ \text{NP-parsing-NN} \]  
\[ \text{VP-is-VB} \]  
\[ S\text{-is-VB} \]  
\[ W_p + W_d \]  
\[ \text{Dependency Grammar} \]  
\[ \text{Lexicalized Grammar} \]
Cost of Optimality: Lexicalized Grammar

CTF Speed vs. Optimality

Edges pushed (billions)

Fraction of sentences without search errors
Conclusions

• Coarse-to-Fine is much faster for reasonable number of search errors

• Hierarchical Coarse-to-Fine effectively exploits multilevel hierarchies, Hierarchical A* does not

• Hierarchical A* is the right choice if
  • optimality is desired
  • heuristics are very tight
Thank you