K-best Parsing: Algorithms and Applications

Liang Huang 黄亮
University of Pennsylvania

joint work with David Chiang 蔣偉
(USC Information Sciences Institute)

Hong Kong University of Science & Technology
July 30, 2007
$k$-best Parsing
$k$-best Parsing

I saw a boy with a telescope.
I saw a boy with a telescope.
I saw a boy with a telescope.
Not a trivial task…
Not a trivial task…

Aravind Joshi
Not a trivial task...

I saw her duck.

Aravind Joshi
Not a trivial task…

I saw her duck.

Aravind Joshi
Not a trivial task…

I saw her duck.

Aravind Joshi
Not a trivial task…

I eat sushi with tuna.

Aravind Joshi
Not a trivial task…

I eat sushi with tuna.

Aravind Joshi
Not a trivial task…

I eat sushi with tuna.

Aravind Joshi
Not a trivial task…

I eat sushi with tuna.

Aravind Joshi

… you’d improve your $k$-best parser
Why $k$-best?

- Postpone disambiguation in a pipeline
  - 1-best is not always optimal in the future
  - Propagate $k$-best lists instead of 1-best
  - E.g.: semantic role labeler uses $k$-best parses
- Approximate the set of all possible interpretations
  - Reranking (Collins, 2000)
  - Minimum error training (Och, 2003)
  - Online training (McDonald et al., 2005)
In this talk...

• Formulations
  – parsing as deduction; the CKY algorithm
  – directed monotonic hypergraphs
• Algorithms
  – Algorithm 0 thru Algorithm 3
• Experiments
• Applications in Machine Translation
Parsing as Deduction

• Parsing with **context-free grammars** (CFGs)
  – Dynamic Programming (CKY algorithm)

\[
\begin{align*}
(B, i, j) & \quad (C, j+1, k) \quad \frac{A \rightarrow BC}{(A, i, k)} \\
(NP, 1, 3) & \quad (VP, 4, 6) \quad \frac{S \rightarrow NP VP}{(S, 1, 6)}
\end{align*}
\]
Parsing as Deduction

- Parsing with **context-free grammars** (CFGs)
  - Dynamic Programming (CKY algorithm)

\[
\begin{align*}
(B, i, j) &\quad (C, j+1, k) \\
\hline
(A, i, k) &\quad A \rightarrow BC
\end{align*}
\]

\[
\begin{align*}
(NP, 1, 3) &\quad (VP, 4, 6) \\
\hline
(S, 1, 6) &\quad S \rightarrow NP\ VP
\end{align*}
\]
Parsing as Deduction

• Parsing with context-free grammars (CFGs)
  – Dynamic Programming (CKY algorithm)

\[
(B, i, j) \ (C, j+1, k) \quad \frac{(A, i, k)}{A \rightarrow B C}
\]

\[
(NP, 1, 3) \ (VP, 4, 6) \quad \frac{(S, 1, 6)}{S \rightarrow NP \ VP}
\]
Parsing as Deduction

- Parsing with context-free grammars (CFGs)
  - Dynamic Programming (CKY algorithm)

\[ (B, i, j) (C, j+1, k) \]
\[ A \rightarrow BC \]
\[ (A, i, k) \]

\[ (NP, 1, 3) (VP, 4, 6) \]
\[ S \rightarrow NP VP \]

\[ \text{computational complexity: } O(n^3 |P|) \]
\[ P \text{ is the set of productions (rules)} \]
Deduction $\Rightarrow$ Hypergraph

- hypergraph is a generalization of graph
  - each hyperedge connects several vertices to one vertex

\[
\begin{array}{c}
\text{(NP, 1, 3)} & \text{(VP, 4, 6)} \\
\text{(S, 1, 6)}
\end{array}
\begin{array}{c}
\text{(VB, 3, 3)} & \text{(PP, 4, 6)} \\
\text{(NP, 1, 3)} & \text{(VP, 4, 6)} \\
\text{(S, 1, 6)}
\end{array}
\]
Deduction => Hypergraph

- hypergraph is a generalization of graph
  - each hyperedge connects several vertices to one vertex

\[
\begin{align*}
\text{(VB, 3, 3)} & \quad \text{(PP, 4, 6)} \\
\hline
\text{(NP, 1, 3)} & \quad \text{(VP, 4, 6)} \\
\hline
\text{(S, 1, 6)} & \quad \text{(S, 1, 6)}
\end{align*}
\]
hypergraph is a generalization of graph

– each hyperedge connects several vertices to one vertex

\[(VB, 3, 3) \ (PP, 4, 6)\]
\[(NP, 1, 3) \ (VP, 4, 6)\]
\[(S, 1, 6)\]
I saw a boy with a telescope

packed forest

a compact representation of all parse trees
I saw a boy with a telescope

packed forest

a compact representation of all parse trees
I saw a boy with a telescope

S

I

v

v

I

saw

a boy

with

a telescope

NP

VP

NP

PP

NP

packed forest

a compact representation of all parse trees
Weighted Deduction/Hypergraph

\[ (B, i, j): \ p \quad (C, j+1, k): \ q \]

\[ \frac{(A, i, k): \ f \ (p, q)}{A \rightarrow B C} \]

- \( f \) is the weight function

\text{e.g.: in Probabilistic Context-Free Grammars:}

\[ f \ (p, q) = p \cdot q \cdot \text{Pr} \ (A \rightarrow B C) \]
Monotonic Weight Functions

- all weight functions must be *monotonic* on each of their arguments
- optimal sub-problem property in dynamic programming

CKY example:

\[ A = (S, 1, 5) \]
\[ B = (NP, 1, 2), \quad C = (VP, 3, 5) \]
\[ f(b, c) = b \cdot c \cdot \Pr(S \rightarrow NP \ VP) \]
Monotonic Weight Functions

- all weight functions must be *monotonic* on each of their arguments
- optimal sub-problem property in dynamic programming

CKY example:

- A: \( f(b', c) \leq f(b, c) \)
- B: \( b' \leq b \)
- C: \( c \)

\[
A = (S, 1, 5) \\
B = (NP, 1, 2), \ C = (VP, 3, 5) \\
f(b, c) = b \cdot c \cdot \Pr(S \rightarrow NP \ VP)
\]
$k$-best Problem in Hypergraphs

- **1-best problem**
  - find the best derivation of the target vertex $t$

- **$k$-best problem**
  - find the top $k$ derivations of the target vertex $t$

- **assumption**
  - acyclic: so that we can use topological order

In CKY, $t = (S, 1, n)$
Outline

• Formulations

• Algorithms
  – Generic 1-best Viterbi Algorithm
  – Algorithm 0: naïve
  – Algorithm 1: hyperedge-level
  – Algorithm 2: vertex (item)-level
  – Algorithm 3: lazy algorithm

• Experiments

• Applications to Machine Translation
Generic 1-best Viterbi Algorithm

- traverse the hypergraph in topological order
  - for each vertex
    - for each incoming hyperedge
      - compute the result of the $f$ function along the hyperedge
      - update the 1-best value for the current vertex if possible

\[
\begin{array}{c}
u: a \\
w: b \\
\end{array}
\xrightarrow{f_1}
\begin{array}{c}
v : f_1(a, b)
\end{array}
\]
Generic 1-best Viterbi Algorithm

- traverse the hypergraph in topological order ("bottom-up")
  - for each vertex
    - for each incoming hyperedge
      - compute the result of the $f$ function along the hyperedge
      - update the 1-best value for the current vertex if possible

\[
\begin{align*}
(VP, 2, 4) & \quad u: a \\
(PP, 4, 7) & \quad w: b \\
\hline
V & : f_1(a, b) \\
(VP, 2, 7) & \quad \text{(VP, 2, 7)}
\end{align*}
\]
Generic 1-best Viterbi Algorithm

- traverse the hypergraph in topological order
  - for each vertex
    - for each incoming hyperedge
      - compute the result of the $f$ function along the hyperedge
      - update the 1-best value for the current vertex if possible

```
u: a
w: b
v: better ($f_1(a, b)$, $f_2(c, d)$)
u': c
w': d
```

Liang Huang (Penn)
Generic 1-best Viterbi Algorithm

- traverse the hypergraph in topological order
  - for each incoming hyperedge
    - compute the result of the $f$ function along the hyperedge
    - update the 1-best value for the current vertex if possible

```
\begin{align*}
\text{u: } a & \quad \quad f_1 \\
\text{w: } b & \quad \quad f_2 \\
\text{u': } c & \quad \quad \text{v} \\
\text{w': } d & \quad \quad \text{\vdots}
\end{align*}
```

\[
\text{v: better} \left( \text{better} \left( f_1(a, b), f_2(c, d) \right), \ldots \right)
\]
Generic 1-best Viterbi Algorithm

- traverse the hypergraph in topological order
  - for each incoming hyperedge
    - compute the result of the $f$ function along the hyperedge
    - update the 1-best value for the current vertex if possible

\[
\begin{align*}
  \text{overall time complexity: } & O(|E|) \\
  \text{better( better( } & f_1(a, b), f_2(c, d), \ldots) \\
\end{align*}
\]
Generic 1-best Viterbi Algorithm

- traverse the hypergraph in topological order
  - for each incoming hyperedge
    - compute the result of the $f$ function along the hyperedge
    - update the 1-best value for the current vertex if possible

\[
\begin{align*}
&\text{overall time complexity: } O(|E|) \\
&\text{in CKY: } |E| = O(n^3|P|)
\end{align*}
\]
Dynamic Programming: 1950’s

Richard Bellman

Andrew Viterbi
Dynamic Programming: 1950’s

Richard Bellman

Andrew Viterbi

We knew everything so far in your talk 40 years ago
$k$-best Viterbi algorithm 0: naïve

- straightforward $k$-best extension:
  - a vector of length $k$ instead of a single value
  - vector components maintain *sorted*
  - now what’s $f(a, b)$?
    - $k^2$ values -- Cartesian Product $f(a_i, b_j)$
    - just need top $k$ out of the $k^2$ values

$$
\begin{align*}
\text{u: } & a \\
\text{w: } & b \\
\end{align*}
\quad \begin{align*}
\mult_k (f_1, a, b) = \top_k \{ f(a_i, b_j) \}
\end{align*}
$$
\( k \)-best Viterbi algorithm 0: naïve

- straightforward \( k \)-best extension:
  - a vector of length \( k \) instead of a single value
  - vector components maintain sorted
  - now what’s \( f(a, b) \)?
    - \( k^2 \) values -- Cartesian Product \( f(a_i, b_j) \)
    - just need top \( k \) out of the \( k^2 \) values

\[
\begin{array}{c}
\text{u: a} \\
\text{w: b}
\end{array}
\quad f_1
\quad \begin{array}{c}
\text{v}
\end{array}
\]

\[
mult_k(f_1, a, b) = \text{top}_k \{ f(a_i, b_j) \}
\]

\[
mult_k(f, a, b) = \text{top}_k \{ f(a_i, b_j) \}
\]
Algorithm 0: naïve

- **straightforward** $k$-best extension:
  - a vector of length $k$ instead of a single value
  - and how to update?
    - from two $k$-lengthed vectors ($2k$ elements)
    - select the top $k$ elements: $O(k)$

$$
\begin{align*}
  u: & \ a \\
  w: & \ b \\
  u': & \ c \\
  w': & \ d \\
  f_1 & \rightarrow v \quad : \text{merge}_k(\text{mult}_k(f_1, a, b), \text{mult}_k(f_2, c, d))
\end{align*}
$$
Algorithm 0: naïve

- **straightforward** $k$-best extension:
  - a vector of length $k$ instead of a single value
  - and how to update?
    - from two $k$-lengthed vectors ($2k$ elements)
    - select the top $k$ elements: $O(k)$

$$
\begin{align*}
\text{u: a} & \quad f_1 \\
\text{w: b} & \quad \text{v} \\
\text{u': c} & \quad f_2 \\
\text{w': d} & \quad \text{merge}_k(\text{mult}_k(f_1, a, b), \text{mult}_k(f_2, c, d))
\end{align*}
$$

overall time complexity: $O(k^2 |E|)$
Algorithm 1: speedup $\text{mult}_k$

$$\text{mult}_k(f, a, b) = \text{top}_k\{f(a_i, b_j)\}$$
Algorithm 1: speedup $\text{mult}_k$

$\text{mult}_k(f, a, b) = \text{top}_k\{f(a_i, b_j)\}$

- only interested in top $k$, why enumerate all $k^2$?

![Diagram showing a grid with values]
Algorithm 1: speedup \textit{mult}_k

\begin{equation}
\text{mult}_k(f, a, b) = \text{top}_k\{f(a_i, b_j)\}
\end{equation}

- only interested in top $k$, why enumerate all $k^2$?
- $a$ and $b$ are sorted!

\begin{tabular}{c|c|c|c|c|c}
 & .1 & & & & \\
\hline
.3 & & & & & \\
\hline
.4 & & & & & \\
\hline
.5 & & & & & \\
\hline
\hline
.6 & .4 & .3 & .3 & &
\end{tabular}
Algorithm 1: speedup \( \text{mult}_k \)

\[
\text{mult}_k(f, a, b) = \text{top}_k \{ f(a_i, b_j) \}
\]

- only interested in top \( k \), why enumerate all \( k^2 \)?
- \( a \) and \( b \) are sorted!
- \( f \) is monotonic!
Algorithm 1: speedup $\text{mult}_k$

\[
\text{mult}_k(f, a, b) = \text{top}_k\{f(a_i, b_j)\}
\]

- only interested in top $k$, why enumerate all $k^2$?
- $a$ and $b$ are sorted!
- $f$ is monotonic!
- $f(a_1, b_1)$ must be the 1-best
Algorithm 1: speedup $\text{mult}_k$

\[ \text{mult}_k(f, a, b) = \text{top}_k\{f(a_i, b_j)\} \]

- only interested in top $k$, why enumerate all $k^2$?
- $a$ and $b$ are sorted!
- $f$ is monotonic!
- $f(a_1, b_1)$ must be the 1-best
- the 2nd-best must be…
  - either $f(a_2, b_1)$ or $f(a_1, b_2)$
Algorithm 1: speedup $\text{mult}_k$

$\text{mult}_k(f, a, b) = \text{top}_k \{ f(a_i, b_j) \}$

- only interested in top $k$, why enumerate all $k^2$?
- $a$ and $b$ are sorted!
- $f$ is monotonic!
- $f(a_1, b_1)$ must be the 1-best
- the 2nd-best must be...
  - either $f(a_2, b_1)$ or $f(a_1, b_2)$
- what about the 3rd-best?
Algorithm 1 (Demo)

\[ f(a, b) = ab \]
Algorithm 1 (Demo)

\[ f(a, b) = ab \]
Algorithm 1 (Demo)

\[ f(a, b) = ab \]
Algorithm 1 (Demo)

\[ f(a, b) = ab \]
Algorithm 1 (Demo)

\[ f(a, b) = ab \]
Algorithm 1 (Demo)

\[ f(a, b) = ab \]
Algorithm 1 (Demo)

\[ f(a, b) = ab \]

\[ \begin{array}{cccc}
  & .1 & .3 & .4 \\
.1 & & & .18 \\
.3 & & & \ \\
.4 & .18 & .24 & \ \\
.5 & .30 & .20 & .6 \\
\end{array} \]
Algorithm 1 (Demo)

use a priority queue (heap) to store the candidates \( \text{(frontier)} \)

\[
\begin{array}{c|c|c|c|c}
\text{b}_j & \text{.1} & \text{.3} & \text{.4} & \text{.5} \\
\hline
\text{.1} & \text{.18} & \text{.24} & \text{.16} & \text{.30} \\
\text{.3} & \text{.20} & \text{.4} & \text{.3} & \text{.3} \\
\end{array}
\]

\( a_i \)
Algorithm 1 (Demo)

use a priority queue (heap) to store the candidates \((\text{frontier})\)

<table>
<thead>
<tr>
<th></th>
<th>(a_i)</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>.1</td>
<td>.3</td>
<td>.4</td>
<td>.5</td>
</tr>
<tr>
<td></td>
<td>.18</td>
<td>.24</td>
<td>.16</td>
<td>.30</td>
</tr>
<tr>
<td>(b_j)</td>
<td>.4</td>
<td>.6</td>
<td>.4</td>
<td>.3</td>
</tr>
</tbody>
</table>

\(k\)-best parsing
Algorithm 1 (Demo)

use a priority queue (heap) to store the candidates (frontier)

\[b_j\]
\[\begin{array}{cccc}
.1 & & & \\
.3 & .18 & & \\
.4 & .24 & .16 & \\
.5 & .30 & .20 & .15 \\
.6 & .4 & .3 & .3 \\
\end{array}\]
use a priority queue (heap) to store the candidates \((\textit{frontier})\)

### Algorithm 1 (Demo)

use a priority queue (heap) to store the candidates \((\textit{frontier})\)

in each iteration:

1. extract-max from the heap
2. push the two “shoulders” into the heap

\(k\) iterations.

\[O(k \log k \ |E|)\] overall time
Algorithm 2: speedup merge $k$

- Algorithm 1 works on each hyperedge sequentially

- can we process them simultaneously?

![Diagram showing hyperedges and nodes labeled with $u: a$, $w: b$, $v$, $f_1$, $f_i$, $f_d$, $p: x$, and $q: y$.]
Algorithm 2 (Demo)

starts with an initial heap of the 1-best derivations from each hyperedge

<p>| | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B₁ x C₁</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.4</td>
<td>0.7</td>
<td>0.1</td>
<td>0.6</td>
<td>0.42</td>
<td>0.7</td>
<td>0.1</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B₂ x C₂</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.3</td>
<td>0.4</td>
<td>0.5</td>
<td>0.9</td>
<td>0.36</td>
<td>0.4</td>
<td>0.9</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B₃ x C₃</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.7</td>
<td>0.8</td>
<td>0.1</td>
<td>0.4</td>
<td>0.32</td>
<td>0.4</td>
<td>0.4</td>
<td></td>
</tr>
</tbody>
</table>

item-level heap

\[ k = 2, \ d = 3 \]
Algorithm 2 (Demo)

pop the best (.42) and …

\[
\begin{bmatrix}
B_1 \times C_1 & 0.1 & 0.6 & 0.4 & 0.7 \\
\end{bmatrix}
\begin{bmatrix}
B_2 \times C_2 & 0.5 & 0.9 & 0.3 & 0.4 \\
\end{bmatrix}
\begin{bmatrix}
B_3 \times C_3 & 0.1 & 0.4 & 0.7 & 0.8 \\
\end{bmatrix}
\]

item-level heap

\[v, k = 2, d = 3\]
Algorithm 2 (Demo)

pop the best (.42) and …  

push the two successors (.07 and .24)

\[
\begin{array}{ccc}
B_1 \times C_1 & B_2 \times C_2 & B_3 \times C_3 \\
0.4 & 0.7 & 0.1 \\
0.24 & 0.42 & 0.6 \\
0.3 & 0.4 & 0.9 \\
0.7 & 0.8 & 0.4 \\
\end{array}
\]

item-level heap

\[
k = 2, \ d = 3
\]

output

\[
0.42
\]
Algorithm 2 (Demo)

pop the 2\textsuperscript{nd}-best (.36)

\begin{align*}
B_1 \times C_1 & \begin{array}{c}
.07 \\
.24 \\
0.4
\end{array}
\begin{array}{c}
0.1 \\
.42 \\
0.7
\end{array} \\
B_2 \times C_2 & \begin{array}{c}
0.5 \\
0.3
\end{array}
\begin{array}{c}
.36 \\
0.4
\end{array} \\
B_3 \times C_3 & \begin{array}{c}
0.1 \\
0.7
\end{array}
\begin{array}{c}
.32 \\
0.8
\end{array}
\end{align*}

item-level heap

\begin{align*}
V & \begin{array}{c}
.42 \\
.36
\end{array} \\
k = 2, \ d = 3
\end{align*}
Algorithm 3: Offline (lazy)

• from Algorithm 0 to Algorithm 2:
  – delaying the calculations until needed -- lazier
  – larger locality

• even lazier… (one step further)
  – we are interested in the $k$-best derivations of the final item only!
Algorithm 3: Offline (lazy)

- **forward phase**
  - do a normal 1-best search till the final item
  - *but* construct the hypergraph (*forest*) along the way

- **recursive backward phase**
  - ask the final item: what’s your 2nd-best?
  - final item will propagate this question till the leaves
  - then ask the final item: what’s your 3rd-best?
after the “forward” step (1-best parsing):

forest = 1-best derivations from each hyperedge
Algorithm 3 demo

now the backward step

what’s your 2nd-best?

NP (1, 2) VP (3, 7)

NP (1, 3) VP (4, 7)

VP (1, 5) NP (6, 7)

S (1, 7)

k=2
I'm not sure... let me ask my parents...
Algorithm 3 demo

well, it must be either … or …

NP (1, 2)  VP (3, 7)

NP (1, 3)  VP (4, 7)

NP (6, 7)

VP (1, 5)

S (1, 7)

k=2

NP (1, 3)

VP (4, 7)

NP (6, 7)

k=2
but wait a minute... did you already know the ?’s ?
Algorithm 3 demo

but wait a minute… did you already know the ?’s ?

oops… forgot to ask more questions recursively …
Algorithm 3 demo

what's your 2nd-best?

NP (1, 2) VP (3, 7) VP (1, 5) NP (6, 7) 

S (1, 7) 

k=2

NP (1, 3) VP (4, 7) 

k=2

what's your 2nd-best?
Algorithm 3 demo

recursion goes on to the leaf nodes

NP (1, 2) VP (3, 7) NP (1, 3) VP (4, 7) VP (1, 5) NP (6, 7) S (1, 7) k=2

Liang Huang (Penn)
Algorithm 3 demo

and reports back the numbers...

NP (1, 2)        VP (3, 7)
    0.4  0.5
?  0.20  ?

NP (1, 3)        VP (4, 7)
    0.5  0.42  0.3
?  ?  0.7  0.6

VP (1, 5)        NP (6, 7)
    0.7  0.4
?  0.28

S (1, 7)  k=2

Liang Huang (Penn)
Algorithm 3 demo

push .30 and .21 to the candidate heap (priority queue)
Algorithm 3 demo

pop the root of the heap (.30)

now I know my 2nd-best
Interesting Properties

- 1-best is best everywhere (all decisions optimal)
- 2nd-best is optimal everywhere except one decision
  - and that decision must be 2nd-best
  - and it’s the best of all 2nd-best decisions
- so what about the 3rd-best?
- kth-best is...

$$\sum_{\delta \in \Delta} (\text{rank}(\delta) - 1) \leq k - 1$$

Local picture:

```
   .1  .06  .04  .03  .03
   .3  .18  .12  .09  .09
   .4  .24  .16  .12  .12
   .5  .30  .20  .15  .15
   .6  .4   .3   .3   .3
```

$$(i - 1) + (j - 1) = 3 - 1$$
## Summary of Algorithms

<table>
<thead>
<tr>
<th>Algorithms</th>
<th>Time Complexity</th>
<th>Locality</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-best</td>
<td>$O(</td>
<td>E</td>
</tr>
<tr>
<td>alg. 0: naïve</td>
<td>$O(k^a</td>
<td>E</td>
</tr>
<tr>
<td>alg. 1</td>
<td>$O(k \log k</td>
<td>E</td>
</tr>
<tr>
<td>alg. 2</td>
<td>$O(</td>
<td>E</td>
</tr>
<tr>
<td>alg. 3: lazy</td>
<td>$O(</td>
<td>E</td>
</tr>
</tbody>
</table>

For CKY: $a=2$, $|E|=O(n^3 |P|)$, $|V|=O(n^2 |N|)$, $|D|=O(n)$

$a$ is the arity of the grammar
Outline

• Formulations

• Algorithms: Alg.0 thru Alg. 3

• **Experiments**
  – Collins/Bikel Parser
  – both efficiency and accuracy

• Applications in Machine Translation
Background: Statistical Parsing

• **Probabilistic Grammar**
  – induced from a treebank (Penn Treebank)

• **State-of-the-art Parsers**
  – Collins (1999), Bikel (2004), Charniak (2000), etc.

• **Evaluation of Accuracy**
  – PARSEVAL: tree-similarity (English treebank: ~90%)

• **Previous work on k-best Parsing:**
  – Collins (2000): turn off dynamic programming
  – Charniak/Johnson (2005): coarse-to-fine, still too slow
Efficiency

Implemented Algorithms 0, 1, 3 on top of Collins/Bikel Parser

Average (wall-clock) time on Penn Treebank (per sentence):

\[ O(|E| + |D| k \log k) \]

\( k \)-best parsing
Oracle Reranking

given $k$ parses of a sentence

- **oracle reranking**: pick the best parse according to the gold-standard
- **real reranking**: pick the best parse according to the score function

<table>
<thead>
<tr>
<th>metric</th>
<th>gold standard</th>
<th>accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;correct&quot; parse</td>
<td>100%</td>
<td></td>
</tr>
<tr>
<td>1-best</td>
<td>89%</td>
<td></td>
</tr>
</tbody>
</table>
given $k$ parses of a sentence

- **oracle reranking**: pick the best parse according to the gold-standard

- **real reranking**: pick the best parse according to the score function

<table>
<thead>
<tr>
<th>k-best parses</th>
<th>accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>gold standard</td>
<td>100%</td>
</tr>
<tr>
<td>“correct” parse</td>
<td>96%</td>
</tr>
<tr>
<td>1-best</td>
<td>89%</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

78%
given $k$ parses of a sentence

- **oracle reranking**: pick the best parse according to the gold-standard

- **real reranking**: pick the best parse according to the score function

<table>
<thead>
<tr>
<th>$k$-best parses</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>oracle</td>
<td>96%</td>
</tr>
<tr>
<td>1-best</td>
<td>91%</td>
</tr>
<tr>
<td>$k$-best parses</td>
<td>89%</td>
</tr>
<tr>
<td>gold standard</td>
<td>100%</td>
</tr>
<tr>
<td>“correct” parse</td>
<td>78%</td>
</tr>
</tbody>
</table>
Quality of the $k$-best lists

This work on top of Collins parser

Collins (2000)

This work with beam width $10^{-4}$
(Collins, 2000)
(Ratnaparkhi, 1997)
Quality of the $k$-best lists

This work on top of Collins parser

Collins (2000)

Oracle F-score

This work with beam width $10^{-4}$
(Collins, 2000)
(Ratnaparkhi, 1997)
Why are our $k$-best lists better?

Collins (2000): turn down dynamic programming theoretically exponential time complexity; aggressive beam pruning to make it tractable in practice.
Why are our $k$-best lists better?

Collins (2000): turn down dynamic programming theoretically exponential time complexity; aggressive beam pruning to make it tractable in practice.
Implemented in ...

- state-of-the-art statistical parsers
  - Charniak parser (2005); Berkeley parser (2006)
  - McDonald et al. dependency parser (2005)
  - Microsoft Research (Redmond) dependency parser (2006)
- generic dynamic programming languages/packages
  - Dyna (Eisner et al., 2005) and Tiburon (May and Knight, 2006)
- state-of-the-art syntax-based translation systems
  - Hiero (Chiang, 2005)
  - ISI syntax-based system (2005)
  - CMU syntax-based system (2006)
  - BBN syntax-based system (2007)
Applications in Machine Translation
Syntax-based Translation

- synchronous context-free grammars (SCFGs)
- generating pairs of strings/trees simultaneously
- co-indexed nonterminal further rewritten as a unit

\[
\begin{align*}
S & \rightarrow \text{NP}^{(1)} \text{VP}^{(2)}, \quad \text{NP}^{(1)} \text{VP}^{(2)} \\
\text{VP} & \rightarrow \text{PP}^{(1)} \text{VP}^{(2)}, \quad \text{VP}^{(2)} \text{PP}^{(1)} \\
\text{NP} & \rightarrow \text{Baoweier}, \quad \text{Powell} \\
\end{align*}
\]

...
Translation as Parsing

- translation ("decoding") => monolingual parsing
- parse the source input with the source projection
  - build the corresponding target sub-strings in parallel
Translation as Parsing

- translation ("decoding") => monolingual parsing
- parse the source input with the source projection
  - build the corresponding target sub-strings in parallel

S → NP(1) VP(2), NP(1) VP(2)
VP → PP(1) VP(2), VP(2) PP(1)
NP → Baoweier, Powell
...

Baoweier yu Shalong juxing le huitan

Liang Huang (Penn)
Translation as Parsing

- translation ("decoding") => monolingual parsing
- parse the source input with the source projection
  - build the corresponding target sub-strings in parallel

\[
\begin{align*}
S & \rightarrow \text{NP}^{(1)} \text{ VP}^{(2)}, \\
\text{VP} & \rightarrow \text{PP}^{(1)} \text{ VP}^{(2)}, \\
\text{NP} & \rightarrow \text{Baoweier}, \\
\ldots
\end{align*}
\]
Translation as Parsing

- translation ("decoding") => monolingual parsing
- parse the source input with the source projection
  - build the corresponding target sub-strings in parallel

\[
S \rightarrow \text{NP}^{(1)} \text{VP}^{(2)}, \\
\text{VP} \rightarrow \text{PP}^{(1)} \text{VP}^{(2)}, \\
\text{NP} \rightarrow \text{Baoweier}, \\
\ldots
\]

NP

Baoweier

PP

yu Shalong

VP

juxing le huitan

Liang Huang (Penn)
Translation as Parsing

- translation ("decoding") => monolingual parsing
- parse the source input with the source projection
- build the corresponding target sub-strings in parallel

\[
\begin{align*}
S & \rightarrow \text{NP}^{(1)} \text{ VP}^{(2)}, \\
\text{VP} & \rightarrow \text{PP}^{(1)} \text{ VP}^{(2)}, \\
\text{NP} & \rightarrow \text{Baoweier}, \\
& \ldots
\end{align*}
\]
Translation as Parsing

- translation (“decoding”) => monolingual parsing
- parse the source input with the source projection
  - build the corresponding target sub-strings in parallel

\[
S \rightarrow \text{NP}^{(1)} \text{ VP}^{(2)}, \\
\text{VP} \rightarrow \text{PP}^{(1)} \text{ VP}^{(2)}, \\
\text{NP} \rightarrow \text{Baoweier}, \\
\ldots 
\]
Translation as Parsing

- translation ("decoding") => monolingual parsing
- parse the source input with the source projection
- build the corresponding target sub-strings in parallel

S → NP(1) VP(2), NP(1) VP(2)
VP → PP(1) VP(2), VP(2) PP(1)
NP → Baoweier, Powell

Liang Huang (Penn)
Translation as Parsing

- translation ("decoding") => monolingual parsing
- parse the source input with the source projection
- build the corresponding target sub-strings in parallel

S → NP(1) VP(2), NP(1) VP(2)
VP → PP(1) VP(2), VP(2) PP(1)
NP → Baoweier, Powell

...
Translation as Parsing

- translation ("decoding") => monolingual parsing
- parse the source input with the source projection
- build the corresponding target sub-strings in parallel

S → NP(1) VP(2), NP(1) VP(2)
VP → PP(1) VP(2), VP(2) PP(1)
NP → Baoweier, Powell

\[ S \rightarrow NP^{(1)} \ VP^{(2)}, \ NP^{(1)} \ VP^{(2)} \]
\[ VP \rightarrow PP^{(1)} \ VP^{(2)}, \ VP^{(2)} \ PP^{(1)} \]
\[ NP \rightarrow Baoweier, \ Powell \]

held a talk with Sharon

Liang Huang (Penn)
Language Model: Rescoring

Spanish/English Bilingual Text

Statistical Analysis

translation model (TM)
competency

Spanish

Broken English

English

language model (LM)
fluency

Que hambre tengo yo

What hunger have I
Hungry I am so
Have I that hunger
I am so hungry
How hunger have I
...

I am so hungry
Language Model: Rescoring

Statistical Analysis

Spanish/English Bilingual Text

Que hambre tengo yo

What hunger have I

Hungry I am so

Have I that hunger

I am so hungry

How hunger have I

...

n-gram LM

English

Broken English

English Text

Statistical Analysis

synchronous CFG

Spanish

I am so hungry

Text

Text

Statistical Analysis

English

Broken

Text

Text

Que hambre tengo yo

Spanish

Broken

English

Spanish

Liang Huang (Penn)
Language Model: Rescoring

Spanish/English Bilingual Text

Statistical Analysis

Spanish

synchronous CFG

Broken English

n-gram LM

English

Statistical Analysis

$k$-best rescoring

Que hambre tengo yo

What hunger have I

Hungry I am so

Have I that hunger

I am so hungry

I am so hungry

How hunger have I

...
Language Model: Rescoring

Spanish/English Bilingual Text

Statistical Analysis

Spanish

synchronous CFG

Broken English

English Text

Statistical Analysis

$n$-gram LM

$k$-best rescoring

Que hambre tengo yo

What hunger have I

Hungry I am so

Have I that hunger

I am so hungry

How hunger have I

I am so hungry

Liang Huang (Penn)
Language Model: Rescoring

Spanish/English Bilingual Text

Statistical Analysis

Spanish

synchronous CFG

Broken

English

k-best rescoring

Que hambre tengo yo

What hunger have I

Hungry I am so

I have I that hunger

I am so hungry

How hunger have I

...
$k$-best rescoring results

- The ISI syntax-based translation system
  - currently the best performing system on Chinese to English task in NIST evaluations
  - based on synchronous grammars
  - translation model (TM) only: BLEU score 24.45
  - rescoring with trigram LM on 25000-best list: 34.58
Language Model: Integration

Spanish/English Bilingual Text

Statistical Analysis

synchronous CFG

Broken English

n-gram LM

English

(Liang Huang, Penn)
Language Model: Integration

Spanish/English Bilingual Text

English Text

Statistical Analysis

synchronous CFG

Broken English

n-gram LM

Que hambre tengo yo

integrated decoder

I am so hungry

computationally challenging! 😞
Integrated Decoding Results

- The ISI syntax-based translation system
  - currently the best performing system on Chinese to English task in NIST evaluations
  - based on synchronous grammars
  - translation model (TM) only: BLEU score 24.45
  - rescoring with trigram LM on 25000-best list: 34.58
  - trigram integrated decoding: 38.44
Integrated Decoding Results

- The ISI syntax-based translation system
  - currently the best performing system on Chinese to English task in NIST evaluations
  - based on synchronous grammars
    - translation model (TM) only: BLEU score 24.45
    - rescoring with trigram LM on 25000-best list: 34.58
    - trigram integrated decoding: 38.44

but very slow!
Rescoring vs. Integration

quality

poor
good

speed

slow

fast

rescoring

Liang Huang (Penn)
Rescoring vs. Integration

- Rescoring (poor quality, fast speed)
- Integration (good quality, slow speed)
Rescoring vs. Integration

- Quality: good vs. poor
- Speed: slow vs. fast

- Rescoring: slow speed, poor quality
- Integration: fast speed, good quality

Any compromise? (on-the-fly rescoring?)

Liang Huang (Penn)
Rescoring vs. Integration

Yes, forest-rescoring -- almost as fast as rescoring, and almost as good as integration

any compromise? (on-the-fly rescoring?)
Why Integration is Slow?

- split each node into +LM items (w/ boundary words)
- beam search: only keep top-k +LM items at each node
- but there are many ways to derive each node
- can we avoid enumerating all combinations?

Liang Huang (Penn)
Why Integration is Slow?

- split each node into +LM items (w/ boundary words)
- beam search: only keep top-$k$ +LM items at each node
- but there are many ways to derive each node
- can we avoid enumerating all combinations?
Forest Rescoring

$k$-best parsing Algorithm 2

with LM cost, we can only do $k$-best \textit{approximately}.

$k$-best parsing

Algorithm 2

with LM cost, we can only do $k$-best \textit{approximately}.

process all hyperedges \textit{simultaneously}!

significant savings of computation

Liang Huang (Penn)
Forest Rescoring Results

- on the Hiero system (Chiang, 2005)
  - ~10 fold speed-up at the same level of BLEU

- on my syntax-directed system (Huang et al., 2006)
  - ~10 fold speed-up at the same level of search-error

- on a typical phrase-based system (Pharaoh)
  - ~30 fold speed-up at the same level of search-error
  - ~100 fold speed-up at the same level of BLEU

- also used in ISI, CMU, and BBN syntax-based systems

see my ACL ’07 paper for details
Conclusions

• monotonic hypergraph formulation
  • the $k$-best derivations problem

• $k$-best Algorithms
  • Algorithm 0 (naïve) to Algorithm 3 (lazy)

• experimental results
  • efficiency
  • accuracy (effectively searching over larger space)

• applications in machine translation
  • $k$-best rescoring and forest rescoring
Thank you!
谢谢!
Questions?
Comments?
Thank you!
谢谢!

Questions?
Comments?
Thank you!

谢谢!

Questions?

Comments?


Quality of the $k$-best lists

![Graph showing quality of $k$-best lists]

- Collins (2000)
- Charniak and Johnson, 2005
- This work with beam width $10^{-4}$
- (Collins, 2000)
- (Ratnaparkhi, 1997)
Syntax-based Experiments
Tree-to-String System

- syntax-directed, English to Chinese (Huang, Knight, Joshi, 2006)
- the reverse direction is found in (Liu et al., 2006)

Synchronous tree-substitution grammars (STSG)

(Galley et al., 2004; Eisner, 2003)
related to
STAG (Shieber/Schabes, 90)

tested on 140 sentences slightly better BLEU scores than Pharaoh
Speed vs. Search Quality

average model cost

full-integration
cube pruning
cube growing

average number of +LM items explored per sentence
Speed vs. Translation Accuracy

The diagram shows the relationship between the average number of +LM items explored per sentence and the BLEU score. Three methods are compared:

- full-integration
- cube pruning
- cube growing

The graph indicates that as the average number of +LM items explored per sentence increases, the BLEU score also increases, with the full-integration method consistently achieving the highest scores.
Cube Pruning

<table>
<thead>
<tr>
<th></th>
<th>1.0</th>
<th>3.0</th>
<th>8.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>(VP _3,6 held * meeting)</td>
<td>1.0</td>
<td>2.0</td>
<td>4.0</td>
</tr>
<tr>
<td>(VP _3,6 held * talk)</td>
<td>1.1</td>
<td>2.1</td>
<td>4.1</td>
</tr>
<tr>
<td>(VP _3,6 hold * conference)</td>
<td>3.5</td>
<td>4.5</td>
<td>6.5</td>
</tr>
</tbody>
</table>

monotonic grid?
Cube Pruning

non-monotonic grid due to LM combo costs

\[(VP_{3,6} \text{ held } \star \text{ meeting})\]
\[(VP_{3,6} \text{ held } \star \text{ talk})\]
\[(VP_{3,6} \text{ hold } \star \text{ conference})\]
Cube Pruning

<table>
<thead>
<tr>
<th></th>
<th>(VP held ⋆ meeting)</th>
<th>(VP held ⋆ talk)</th>
<th>(VP hold ⋆ conference)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>VP₁, 6</strong></td>
<td>1.0</td>
<td>1.1</td>
<td>3.5</td>
</tr>
<tr>
<td><strong>PP₁, 3</strong></td>
<td>2.0 + 0.5</td>
<td>2.1 + 0.3</td>
<td>4.5 + 0.6</td>
</tr>
<tr>
<td><strong>VP₃, 6</strong></td>
<td>4.0 + 5.0</td>
<td>4.1 + 5.4</td>
<td>6.5 + 10.5</td>
</tr>
<tr>
<td><strong>Sharon</strong></td>
<td>9.0 + 0.5</td>
<td>9.1 + 0.3</td>
<td>11.5 + 0.6</td>
</tr>
<tr>
<td><strong>Shalong</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

non-monotonic grid due to LM combo costs

bigram (**meeting**, **with**)
Cube Pruning

<table>
<thead>
<tr>
<th></th>
<th>1.0</th>
<th>3.0</th>
<th>8.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>(VP _held * meeting)_{3,6}</td>
<td>1.0</td>
<td>2.5</td>
<td>9.0</td>
</tr>
<tr>
<td>(VP _held * talk)_{3,6}</td>
<td>1.1</td>
<td>2.4</td>
<td>9.5</td>
</tr>
<tr>
<td>(VP _hold * conference)_{3,6}</td>
<td>3.5</td>
<td>5.1</td>
<td>17.0</td>
</tr>
</tbody>
</table>

non-monotonic grid
due to LM combo costs

Liang Huang (Penn)
Cube Pruning

**k-best parsing**

**Algorithm 1**

- a priority queue of candidates
- extract the best candidate

<table>
<thead>
<tr>
<th></th>
<th>PP with Sharon</th>
<th>PP along Sharon</th>
<th>PP with Shalong</th>
</tr>
</thead>
<tbody>
<tr>
<td>(VP held * meeting)</td>
<td>1.0</td>
<td>3.0</td>
<td>8.0</td>
</tr>
<tr>
<td>(VP held * talk)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(VP hold * conference)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(VP with * Sharon 1,3)</td>
<td>1.0</td>
<td>2.5</td>
<td>9.5</td>
</tr>
<tr>
<td>(VP with * Shalong 1,3)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(VP with * Sharon 1,3)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(VP with * Shalong 1,3)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Liang Huang (Penn)
### Cube Pruning

#### k-best parsing

**Algorithm 1**

- a priority queue of candidates
- extract the best candidate
- push the two successors

<table>
<thead>
<tr>
<th></th>
<th>(PP with * Sharon)</th>
<th>(PP along * Sharon)</th>
<th>(PP with * Shalong)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(VP held * meeting)</td>
<td>1.0</td>
<td>3.0</td>
<td>8.0</td>
</tr>
<tr>
<td>(VP held * talk)</td>
<td>1.1</td>
<td>2.4</td>
<td>9.5</td>
</tr>
<tr>
<td>(VP hold * conference)</td>
<td>3.5</td>
<td>5.1</td>
<td>17.0</td>
</tr>
</tbody>
</table>

Liang Huang (Penn)
**Cube Pruning**

### k-best parsing

**Algorithm 1**

- a priority queue of candidates
- extract the best candidate
- push the two successors

<table>
<thead>
<tr>
<th></th>
<th>PP with * Sharon</th>
<th>PP along * Sharon</th>
<th>PP with * Shalong</th>
</tr>
</thead>
<tbody>
<tr>
<td>(VP held * meeting)_{3,6}</td>
<td>1.0</td>
<td>2.5</td>
<td>9.0</td>
</tr>
<tr>
<td>(VP held * talk)_{3,6}</td>
<td>1.1</td>
<td>2.4</td>
<td>9.5</td>
</tr>
<tr>
<td>(VP hold * conference)_{3,6}</td>
<td>3.5</td>
<td>5.1</td>
<td>17.0</td>
</tr>
</tbody>
</table>
### Cube Pruning

**items are popped out-of-order**

**solution:** keep a buffer of pop-ups

<table>
<thead>
<tr>
<th></th>
<th>(PP with * Sharon)</th>
<th>(PP along * Sharon)</th>
<th>(PP with * Shalong)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(VP ( \text{held} * \text{meeting} )) ( \begin{array}{c} 3,6 \end{array} )</td>
<td>1.0</td>
<td>3.0</td>
<td>8.0</td>
</tr>
<tr>
<td></td>
<td>1.0</td>
<td>2.5</td>
<td>9.0</td>
</tr>
<tr>
<td>(VP ( \text{held} * \text{talk} )) ( \begin{array}{c} 3,6 \end{array} )</td>
<td>1.1</td>
<td>2.4</td>
<td>9.5</td>
</tr>
<tr>
<td></td>
<td>3.5</td>
<td>5.1</td>
<td>17.0</td>
</tr>
</tbody>
</table>
Cube Pruning

Items are popped out-of-order.

**Solution**: keep a buffer of pop-ups

2.5 2.4 5.1

Finally re-sort the buffer and return in order:

2.4 2.5 5.1

<table>
<thead>
<tr>
<th></th>
<th>1.0</th>
<th>3.0</th>
<th>8.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>(VP held * meeting)</td>
<td>1.0</td>
<td>2.5</td>
<td>9.0</td>
</tr>
<tr>
<td>(VP held * talk)</td>
<td>1.1</td>
<td>2.4</td>
<td>9.5</td>
</tr>
<tr>
<td>(VP hold * conference)</td>
<td>3.5</td>
<td>5.1</td>
<td>17.0</td>
</tr>
</tbody>
</table>

Liang Huang (Penn)