K-best Parsing Algorithms

Liang Huang 黃亮
University of Pennsylvania

joint work with David Chiang 蔣偉
(USC Information Sciences Institute)
$k$-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
I saw her duck.
I saw her duck.

how about

“I saw her duck with a telescope.”
I saw her duck with a telescope.
I saw her duck with a telescope.
I saw her duck with a telescope.
Another Example

I eat sushi with tuna.

Aravind Joshi
I eat sushi with tuna.
Another Example

I eat sushi with tuna.

Aravind Joshi
Or even...

I eat sushi with tuna.

Aravind Joshi
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)
**k-Best Viterbi Algorithm?**

1. **topological sort**

2. visit each vertex v in sorted order and do updates
   - for each incoming edge \((u, v)\) in E
   - use \(d(u)\) to update \(d(v)\): 
     \[
     d(v) \oplus = d(u) \otimes w(u, v)
     \]
   - key observation: \(d(u)\) is fixed to optimal at this time

3. time complexity: \(O(V + E)\)
**k-Best Viterbi Algorithm?**

1. topological sort

2. visit each vertex \( v \) in sorted order and do updates
   - for each **incoming** edge \((u, v)\) in \( E \)
   - use \( d(u) \) to update \( d(v) \):
     
     \[
     d(v) \oplus = d(u) \otimes w(u, v)
     \]
   - key observation: \( d(u) \) is fixed to optimal at this time

3. time complexity: \( O(V + E) \)
**k-Best Viterbi Algorithm?**

1. topological sort

2. visit each vertex $v$ in sorted order and do updates
   - for each incoming edge $(u, v)$ in $E$
   - use $d(u)$ to update $d(v)$: $d(v) \oplus = d(u) \otimes w(u, v)$
   - key observation: $d(u)$ is fixed to optimal at this time

3. time complexity: $O(V + E)$
**k-Best Viterbi Algorithm?**

1. topological sort

2. visit each vertex v in sorted order and do updates
   - for each incoming edge \((u, v)\) in E
   - use \(d(u)\) to update \(d(v)\):
     \[
     d(v) \oplus = d(u) \otimes w(u, v)
     \]
   - key observation: \(d(u)\) is fixed to optimal at this time

3. time complexity: \(O(V + E)\)

naive k-best: \(O(kE)\)
**k-Best Viterbi Algorithm?**

1. **topological sort**

2. visit each vertex \( v \) in sorted order and do updates
   - for each *incoming* edge \((u, v)\) in \( E \)
   - use \( d(u) \) to update \( d(v) \):
     \[
     d(v) \oplus = d(u) \otimes w(u, v)
     \]
   - key observation: \( d(u) \) is fixed to optimal at this time

3. time complexity: \( O(V + E) \)

naive \( k \)-best: \( O(k E) \)

good \( k \)-best: \( O(E + k \log k V) \)

---

Liang Huang 9 Dynamic Programming
Parsing as Deduction

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

\[
\begin{align*}
(B, i, j) & \quad (C, j+1, k) \\
\overline{A \rightarrow B C} & \\
(A, i, k) & \\
\end{align*}
\]
Parsing as Deduction

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

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

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

• Parsing with context-free grammars (CFGs)

  – Dynamic Programming (CKY algorithm)

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

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

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

computationa‌l complexity: \( O(n^3 |P|) \)

\( P \) is the set of productions (rules)
Deduction => Hypergraph

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

\[(\text{NP}, 1, 3) \quad (\text{VP}, 4, 6)\]  
\[\text{Hypergraph}\]

\[\text{S, 1, 6}\]
hypergraph is a generalization of graph

- each hyperedge connects several vertices to one vertex

\[ (\text{NP, 1, 3}) \quad (\text{VP, 4, 6}) \]

\[ (\text{VB, 3, 3}) \quad (\text{PP, 4, 6}) \]

\[ (\text{S, 1, 6}) \]
Deduction => Hypergraph

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

\[
\begin{align*}
(NP, 1, 3) & \quad (VP, 4, 6) \\
(VB, 3, 3) & \quad (PP, 4, 6) \\
(S, 1, 6)
\end{align*}
\]
I saw a boy with a telescope

packed forest

a compact representation of all parse trees

Liang Huang (Penn)
Packed Forest as Hypergraph

I saw a boy with a telescope

packed forest

a compact representation of all parse trees

Liang Huang (Penn)

k-best parsing 13
Packed Forest as Hypergraph

I saw a boy with a telescope

packed forest

a compact representation of all parse trees
Weighted Deduction/Hypergraph

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

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

- \(f\) is the weight function

\[f (p, q) = p \cdot q \cdot Pr (A \rightarrow B C)\]

e.g.: in Probabilistic Context-Free Grammars:
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

A = (S, 1, 5)
B = (NP, 1, 2), C = (VP, 3, 5)

\[ f(b', c) \leq f(b, c) \]

CKY example:

A = (S, 1, 5)
B = (NP, 1, 2), C = (VP, 3, 5)

\[ f(b, c) = b \cdot c \cdot \text{Pr}(S \rightarrow NP \ VP) \]
$k$-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

$u: a$ 
$w: b$ 
$v: f_1(a, b)$
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 \\
  & \quad f_1 \\
  (VP, 2, 7) & \quad v \quad : f_1(a, b)
\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 | f1 |
| w: b |
| u': c |
| w': d |
```

$v$ : better $(f_1(a, b), f_2(c, d))$
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

$f_1(a, b), f_2(c, d), \ldots$
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

overall time complexity: $O(|E|)$
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

$$
\text{better( better (} f_1(a, b), f_2(c, d), \ldots \text{) )}
$$

overall time complexity: $O(|E|)$
in CKY: $|E| = O(n^3|P|)$
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

\[
\begin{align*}
\text{u: } & a \\
\text{w: } & b \\
\text{f}_1 & \\
\text{v: } & \text{mult}_k(f_1, a, b)
\end{align*}
\]

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

\[
\begin{array}{c|c|c|c|c}
 & .1 & .3 & .4 & .5 \\
\hline
a & & & & .3 \\
\hline
b & .3 & .4 & .3 & .3 \\
\end{array}
\]

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

\[
\begin{array}{c}
u: a \\
w: b \\
\end{array}
\]

\[
\begin{array}{c}
\downarrow \\
f_1 \\
\downarrow \\
v
\end{array}
\]

\[
\text{mult}_k(f_1, a, b)
\]
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: $O(k)$

\[
\begin{array}{c}
\text{u: a} \\
\text{w: b} \\
\text{u': c} \\
\text{w': d}
\end{array}
\]  
\[
\begin{array}{c}
\text{f}_1 \\
\text{f}_2
\end{array}
\]  
\[
\text{v : merge}_k (\text{mult}_k (f_1, a, b), \text{mult}_k (f_2, c, d))
\]
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: \( O(k) \)

\[
\begin{align*}
  u: & \ a \\
  w: & \ b \\
  u': & \ c \\
  w': & \ d
\end{align*}
\]

\[ \begin{array}{c}
  f_1 \\
  f_2 \\
  v
\end{array} \]

: \text{merge}_k(\text{mult}_k(f_1, a, b), \text{mult}_k(f_2, c, d))

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 $k^2$?

<table>
<thead>
<tr>
<th></th>
<th>.1</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>.1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.5</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>.6</td>
<td>.4</td>
<td>.3</td>
<td>.3</td>
</tr>
</tbody>
</table>

a

b
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!

\[
\begin{array}{cccc}
\cdot.1 & & & \\
\cdot.3 & & & \\
\cdot.4 & & & \\
\cdot.5 & & & \\
\cdot.6 & \cdot.4 & \cdot.3 & \cdot.3 \\
\end{array}
\]

\( a \) \( b \)
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!

\[
\begin{array}{cccc}
.1 & .3 & .4 & .5 \\
.6 & .4 & .3 & .3 \\
\end{array}
\]

b

a
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)

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

\[ \begin{array}{cccc}
0.1 &  &  & \\
 & 0.3 &  & \\
 &  & 0.4 & \\
 &  &  & 0.5 \\
 &  &  &  \\
0.6 & 0.4 & 0.3 & 0.3
\end{array} \]
Algorithm 1 (Demo)

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

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

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

\[ f(a, b) = ab \]
use a priority queue (heap) to store the candidates (frontier)

```
\begin{array}{cccc}
\text{a_i} & .6 & .4 & .3 & .3 \\
\hline
.3 & .18 & & & \\
.4 & .24 & .16 & & \\
.5 & .30 & .20 & & \\
\end{array}
```

\(b_j\)
Algorithm 1 (Demo)

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

<table>
<thead>
<tr>
<th></th>
<th>.1</th>
<th>.3</th>
<th>.4</th>
<th>.5</th>
</tr>
</thead>
<tbody>
<tr>
<td>.3</td>
<td>.18</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.4</td>
<td>.24</td>
<td>.16</td>
<td></td>
<td></td>
</tr>
<tr>
<td>.5</td>
<td>.30</td>
<td>.20</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\( b_j \)

\( a_i \)
use a priority queue (heap) to store the candidates \textit{(frontier)}
Algorithm 1 (Demo)

use a priority queue (heap) to store the candidates (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?
starts with an initial heap of the 1-best derivations from each hyperedge
Algorithm 2 (Demo)

pop the best (.42) and …

\[ \begin{array}{ccc}
B_1 \times C_1 &  & B_2 \times C_2 &  & B_3 \times C_3 \\
0.4 & 0.7 & 0.1 & 0.6 & 0.5 & 0.9 & 0.3 & 0.4 & 0.7 & 0.8 & 0.1 & 0.4 & 0.32 & 0.4 \\
0.42 &  & 0.36 &  & .32 & \\
\end{array} \]

item-level heap

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

pop the best (.42) and …

push the two successors (.07 and .24)

\[
\begin{array}{c|c|c}
B_1 \times C_1 & B_2 \times C_2 & B_3 \times C_3 \\ 
\hline 
.24 & .42 & .36 \\ 
.07 & 0.1 & 0.5 \\ 
0.4 & 0.7 & 0.3 & 0.4 \\ 
0.7 & 0.8 & 0.1 \\
\end{array}
\]

item-level heap

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

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

\begin{align*}
B_1 \times C_1 & : \begin{array}{ccc}
0.4 & 0.7 & 0.1 \\
0.24 & 0.42 & 0.6 \\
\end{array} \\
B_2 \times C_2 & : \begin{array}{ccc}
0.3 & 0.4 & 0.5 \\
0.36 & 0.9 & \textbf{0.9} \\
\end{array} \\
B_3 \times C_3 & : \begin{array}{ccc}
0.7 & 0.8 & 0.1 \\
0.32 & 0.4 & \textbf{0.4} \\
\end{array}
\end{align*}

item-level heap

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

\begin{array}{c}
\text{output} \\
\begin{array}{c}
\text{0.42} \\
\text{0.36} \\
\end{array}
\end{array}
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?
Algorithm 3 demo

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) k=2

S (1, 7)
Algorithm 3 demo

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

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

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

NP (6, 7)

S (1, 7)
Algorithm 3 demo

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

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

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

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

S (1, 7)

k=2

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

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
**Algorithm 3 demo**

Recursion goes on to the leaf nodes.
Algorithm 3 demo

and reports back the numbers…

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

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

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

S (1, 7)

k=2

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

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

pop the root of the heap (.30)

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

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

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

now I know my 2nd-best

S (1, 7)

k=2

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

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

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

now I know my 2nd-best

S (1, 7)

k=2

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

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

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

now I know my 2nd-best

S (1, 7)

k=2

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

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

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

now I know my 2nd-best

S (1, 7)

k=2

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

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

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

now I know my 2nd-best

S (1, 7)

k=2

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

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

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

now I know my 2nd-best

S (1, 7)

k=2
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:

\[
\begin{array}{cccccc}
  & .1 & .06 & .04 & .03 & .03 \\
.3 & .18 & .12 & .09 & .09 \\
.4 & .24 & .16 & .12 & .12 \\
.5 & .30 & .20 & .15 & .15 \\
.6 & .4 & .3 & .3 \\
\end{array}
\]
## 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):

$O(|E| + |D| k \log k)$
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>gold standard</th>
<th>accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;correct&quot; parse</td>
<td>100%</td>
</tr>
</tbody>
</table>

1-best

89%
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>&quot;correct&quot; parse</td>
<td>96%</td>
</tr>
<tr>
<td>1-best</td>
<td>91%</td>
</tr>
<tr>
<td>...</td>
<td>89%</td>
</tr>
<tr>
<td>...</td>
<td>78%</td>
</tr>
</tbody>
</table>
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>gold standard</th>
<th>accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>“correct” parse</td>
<td>100%</td>
</tr>
</tbody>
</table>

- oracle: 96%
- $k$-best parses:
  - 91%
  - ...
- 1-best:
  - 89%
  - ...
- 78%
Quality of the $k$-best lists

This work on top of Collins parser

Collins (2000)
Quality of the $k$-best lists

This work with beam width $10^{-4}$
(Collins, 2000)
(Ratnaparkhi, 1997)

This work on top of Collins parser

Collins (2000)
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

Collins (2000) vs this work
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.

7% of the test-set
1. topological sort

2. visit each vertex \( v \) in sorted order and do updates
   - for each incoming edge \( (u, v) \) in \( E \)
   - use \( d(u) \) to update \( d(v) \):
     \[
     d(v) \oplus = d(u) \otimes w(u, v)
     \]
   - key observation: \( d(u) \) is fixed to optimal at this time

3. time complexity: \( O(V + E) \)
**1. topological sort**

2. visit each vertex $v$ in sorted order and do updates
   - for each **incoming** edge $(u, v)$ in $E$
   - use $d(u)$ to update $d(v)$: $d(v) \oplus = d(u) \otimes w(u, v)$
   - key observation: $d(u)$ is fixed to optimal at this time

3. time complexity: $O(V + E)$
k-Best Viterbi Algorithm?

1. topological sort

2. visit each vertex \( v \) in sorted order and do updates
   - for each incoming edge \((u, v)\) in \( E \)
   - use \( d(u) \) to update \( d(v) \):
     \[
     d(v) \oplus = d(u) \otimes w(u, v)
     \]
   - key observation: \( d(u) \) is fixed to optimal at this time

3. time complexity: \( O(V + E) \)
1. topological sort

2. visit each vertex \( v \) in sorted order and do updates
   - for each incoming edge \( (u, v) \) in \( E \)
   - use \( d(u) \) to update \( d(v) \):
     \[ d(v) \oplus = d(u) \otimes w(u, v) \]
   - key observation: \( d(u) \) is fixed to optimal at this time

3. time complexity: \( O(V + E) \)
**k-Best Viterbi Algorithm?**

1. topological sort

2. visit each vertex \( v \) in sorted order and do updates
   - for each incoming edge \((u, v)\) in \( E \)
   - use \( d(u) \) to update \( d(v) \):
     \[ d(v) \oplus = d(u) \otimes w(u, v) \]
   - key observation: \( d(u) \) is fixed to optimal at this time

3. time complexity: \( O(V + E) \)

Algorithm 0: \( O(kE) \)

Algorithm 2: \( O(E + k \log k V) \)
1. topological sort

2. visit each vertex $v$ in sorted order and do updates
   - for each incoming edge $(u, v)$ in $E$
   - use $d(u)$ to update $d(v)$: $d(v) = d(u) \odot w(u, v)$
   - key observation: $d(u)$ is fixed to optimal at this time

3. time complexity: $O(V + E)$

Algorithm 0: $O(kE)$
Algorithm 2: $O(E + k \log kV)$
Algorithm 3: $O(E + k \log kD)$
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
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

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

...
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

...
Translation as Parsing

• translation (“decoding”) => monolingual parsing

• parse the source input with the source projection
  • build the corresponding target sub-strings in parallel

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

Baoweier     yu     Shalong     juxing le huitan
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

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

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

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

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

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

Liang Huang (Penn)

Applications in MT 58
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

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

held a talk with Sharon

\[
\begin{align*}
NP & \quad PP & \quad VP \\
Powell & \quad with Sharon & \quad held a talk \\
Baoweier & \quad yu Shalong & \quad juxing le huitan
\end{align*}
\]
**Language Model: Rescoring**

Spanish/English Bilingual Text

Statistical Analysis

**translation model (TM)**

competency

Spanish

Broken English

**language model (LM)**

fluency

English

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

- Spanish/English Bilingual Text
  - Statistical Analysis
  - synchronous CFG
    - 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
    - Broken English
  - n-gram LM
    - English Text
      - Statistical Analysis

Liang Huang (Penn)
Language Model: Rescoring

Statistical Analysis

Spanish/English
Bilingual Text

English
Text

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
How hunger have I
...
Language Model: Rescoring

Spanish/English Bilingual Text

Statistical Analysis

Spanish

synchronous CFG

Broken English

n-gram LM

English

$k$-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

Have I that hunger

I am so hungry

How hunger have I

...
Language Model: Rescoring

- Spanish/English Bilingual Text
  - Statistical Analysis
  - Synchronous CFG
- English Text
  - Statistical Analysis
  - n-gram LM

$k$-best rescoring

<table>
<thead>
<tr>
<th>Que hambre tengo yo</th>
<th>What hunger have I</th>
<th>I am so hungry</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.7</td>
<td>Hungry I am so</td>
<td>5.4</td>
</tr>
<tr>
<td>3.8</td>
<td>Have I that hunger</td>
<td>7.0</td>
</tr>
<tr>
<td>4.1</td>
<td>I am so hungry</td>
<td>9.8</td>
</tr>
<tr>
<td>4.5</td>
<td>How hunger have I</td>
<td>0.2</td>
</tr>
<tr>
<td>7.2</td>
<td></td>
<td>8.7</td>
</tr>
</tbody>
</table>

Liang Huang (Penn)
**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
- English Text
  - n-gram LM
- Statistical Analysis

Spanish

- (Knight and Koehn, 2003)

Liang Huang (Penn)
Language Model: Integration

Spanish/English Bilingual Text

statistical analysis

synchronous CFG

Que hambre tengo yo

integrated decoder

I am so hungry

English

n-gram LM

I am so hungry

decoder (LM-integrated)

(Knight and Koehn, 2003)
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 over 100 times slower!
The ISI syntax-based translation system is currently the best performing system on Chinese to English task in NIST evaluations. Based on synchronous grammars, the translation model (TM) only achieves a BLEU score of 24.45. Rescoring with trigram LM on 25000-best list increases the BLEU score to 34.58, and trigram integrated decoding further improves it to 38.44. However, this method is over 100 times slower than the others.
Rescoring vs. Integration

![Diagram of rescoring vs. integration on a speed vs. quality plane. Rescoring is shown as the area where speed is fast and quality is poor.](diagram.png)
Rescoring vs. Integration

quality

speed

poor
good

slow

fast

rescoring

integration
Rescoring vs. Integration

- Good quality
  - Integration
- Poor quality
  - Rescoring

? any compromise? (on-the-fly rescoring?)
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 \textbf{approximately}.

process all hyperedges \textbf{simultaneously}!

significant savings of computation
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 NIST evals by 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 the quality of $k$-best lists compared to previous work.](image)
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)

related to STAG (Shieber/Schabes, 90)

tested on 140 sentences slightly better BLEU scores than Pharaoh

Liang Huang (Penn)
Speed vs. Search Quality

The graph depicts the relationship between the average model cost and the average number of +LM items explored per sentence, illustrating the trade-off between speed and search quality for different methods: full-integration, cube pruning, and cube growing. The y-axis represents the average model cost, and the x-axis shows the average number of +LM items explored per sentence.
Speed vs. Translation Accuracy

![Graph showing the relationship between BLEU score and the average number of +LM items explored per sentence. The graph compares different methods: full-integration, cube pruning, and cube growing.]
Cube Pruning

monotonic grid?

(VP_{3,6} \text{ held} \ast \text{ meeting})

(VP_{3,6} \text{ held} \ast \text{ talk})

(VP_{3,6} \text{ hold} \ast \text{ conference})

<table>
<thead>
<tr>
<th></th>
<th>1.0</th>
<th>3.0</th>
<th>8.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.0</td>
<td>2.0</td>
<td>4.0</td>
<td>9.0</td>
</tr>
<tr>
<td>1.1</td>
<td>2.1</td>
<td>4.1</td>
<td>9.1</td>
</tr>
<tr>
<td>3.5</td>
<td>4.5</td>
<td>6.5</td>
<td>11.5</td>
</tr>
</tbody>
</table>
Cube Pruning

non-monotonic grid due to LM combo costs

<table>
<thead>
<tr>
<th>(VP_held * meeting_3,6)</th>
<th>1.0</th>
<th>3.0</th>
<th>8.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>(VP_held * talk_3,6)</td>
<td>1.1</td>
<td>2.1 + 0.3</td>
<td>4.1 + 5.4</td>
</tr>
<tr>
<td>(VP_hold * conference_3,6)</td>
<td>3.5</td>
<td>4.5 + 0.6</td>
<td>6.5 + 10.5</td>
</tr>
</tbody>
</table>
**Cube Pruning**

### Applications in MT

Liang Huang (Penn)

---

#### Non-monotonic Grid Due to LM Combo Costs

<table>
<thead>
<tr>
<th>(VP held * meeting)</th>
<th>(PP along * Sharon)</th>
<th>(PP with * Shalong)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>1.0</strong></td>
<td>2.0 + <strong>0.5</strong></td>
<td>4.0 + 5.0</td>
</tr>
<tr>
<td><strong>1.1</strong></td>
<td>2.1 + <strong>0.3</strong></td>
<td>4.1 + 5.4</td>
</tr>
<tr>
<td><strong>3.5</strong></td>
<td>4.5 + <strong>0.6</strong></td>
<td>6.5 + 10.5</td>
</tr>
</tbody>
</table>

---

<table>
<thead>
<tr>
<th>bigram (meeting, with)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.0</td>
</tr>
<tr>
<td>3.0</td>
</tr>
<tr>
<td>8.0</td>
</tr>
</tbody>
</table>

---

<table>
<thead>
<tr>
<th>PP1,3</th>
<th>VP1,6</th>
<th>VP3,6</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

---

- PP1,3
- VP1,6
- VP3,6
Cube Pruning

non-monotonic grid
due to LM combo costs

<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>PP 1, 3</td>
<td>1.0</td>
<td>3.0</td>
<td>8.0</td>
</tr>
<tr>
<td>(VP 3, 6 held * meeting)</td>
<td>1.0</td>
<td>2.5</td>
<td>9.0</td>
</tr>
<tr>
<td>(VP 3, 6 held * talk)</td>
<td>1.1</td>
<td>2.4</td>
<td>9.5</td>
</tr>
<tr>
<td>(VP 3, 6 hold * conference)</td>
<td>3.5</td>
<td>5.1</td>
<td>17.0</td>
</tr>
</tbody>
</table>

Liang Huang (Penn)
Cube Pruning

\[ k\text{-best parsing} \]

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_{3,6}^\text{held} \ast \text{meeting}))</td>
<td>1.0</td>
<td>2.5</td>
<td>9.0</td>
</tr>
<tr>
<td>((VP_{3,6}^\text{held} \ast \text{talk}))</td>
<td>1.1</td>
<td>2.4</td>
<td>9.5</td>
</tr>
<tr>
<td>((VP_{3,6}^\text{hold} \ast \text{conference}))</td>
<td>3.5</td>
<td>5.1</td>
<td>17.0</td>
</tr>
</tbody>
</table>
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 (1,3)</th>
<th>PP along (1,3)</th>
<th>PP with (1,3)</th>
</tr>
</thead>
<tbody>
<tr>
<td>((VP_{\text{held}} \ast \text{meeting})_{3,6})</td>
<td>1.0</td>
<td>2.5</td>
<td>9.0</td>
</tr>
<tr>
<td>((VP_{\text{held}} \ast \text{talk})_{3,6})</td>
<td>1.1</td>
<td>2.4</td>
<td>9.5</td>
</tr>
<tr>
<td>((VP_{\text{hold}} \ast \text{conference})_{3,6})</td>
<td>3.5</td>
<td>5.1</td>
<td>17.0</td>
</tr>
</tbody>
</table>
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 (_{3,6}) held * meeting)</td>
<td>1.0</td>
<td>3.0</td>
<td>8.0</td>
</tr>
<tr>
<td>(VP (_{3,6}) held * talk)</td>
<td>1.1</td>
<td>2.4</td>
<td>9.5</td>
</tr>
<tr>
<td>(VP (_{3,6}) hold * conference)</td>
<td>3.5</td>
<td>5.1</td>
<td>17.0</td>
</tr>
</tbody>
</table>

Liang Huang (Penn)
# Cube Pruning

items are popped out-of-order
solution: keep a buffer of pop-ups

![Table with data points](image)

<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 2.5 9.0</td>
<td>9.5</td>
<td></td>
</tr>
<tr>
<td>(VP held ⋆ talk)</td>
<td>1.1 2.4 9.5</td>
<td>9.4</td>
<td></td>
</tr>
<tr>
<td>(VP hold ⋆ conference)</td>
<td>3.5 5.1 17.0</td>
<td>12.1</td>
<td></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 inorder:

2.4 2.5 5.1

<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 2.5 9.0 9.5</td>
<td>1.1 2.4 9.5 9.4</td>
<td>3.5 5.1 17.0 12.1</td>
</tr>
</tbody>
</table>

finally re-sort the buffer and return inorder: