Forest-Based Search Algorithms
for Parsing and Machine Translation

Liang Huang
University of Pennsylvania

Google Research, March 14th, 2008
Search in NLP

- is not trivial!

I saw her duck.

Aravind Joshi
Search in NLP

- is not trivial!

I saw her duck.

Aravind Joshi
Search in NLP

- is not trivial!

I eat sushi with tuna.

Aravind Joshi
Search in NLP

- is not trivial!

I eat sushi with tuna.

Aravind Joshi
I saw her duck.
I saw her duck.

• how about...
  
  • I saw her duck with a telescope.
I saw her duck.

- how about...

- I saw her duck with a telescope.
I saw her duck.

- how about...
  - I saw her duck with a telescope.
  - I saw her duck with a telescope in the garden...
I saw her duck.

- how about...
  - I saw her duck with a telescope.
  - I saw her duck with a telescope in the garden...
Parsing/NLP is HARD!

- exponential explosion of the search space
  - solution: locally factored space => packed forest
  - efficient algorithms based on dynamic programming
- non-local dependencies
  - solution: ???

```
S
  NP
    PRP
    VBD
    NP
    PP
      saw
      PRP$
      her
      duck
      IN
      with
      DT
      NN
      a
      telescope
```
Parsing/NLP is HARD!

- exponential explosion of the search space
  - solution: locally factored space => packed forest
- efficient algorithms based on dynamic programming
- non-local dependencies
  - solution: ???

```
eat sushi with tuna
saw her duck with a telescope
```
Key Problem
Key Problem

• How to efficiently incorporate non-local information?
Key Problem

- How to efficiently incorporate non-local information?
- **Solution 1**: pipelined reranking / rescoring
  - postpone disambiguation by propagating $k$-best lists
  - examples: tagging $\Rightarrow$ parsing $\Rightarrow$ semantics
  - need very efficient algorithms for $k$-best search
Key Problem

- How to efficiently incorporate non-local information?
- **Solution 1**: pipelined reranking / rescoring
  - postpone disambiguation by propagating $k$-best lists
  - examples: tagging $\rightarrow$ parsing $\rightarrow$ semantics
  - need very efficient algorithms for $k$-best search
- **Solution 2**: joint approximate search
  - integrate non-local information in the search
  - intractable; so only approximately
  - largely open
Outline

• Packed Forests and Hypergraph Framework
• Exact $k$-best Search in the Forest (for Solution 1)
• Approximate Joint Search (Solution 2) with Non-Local Features
  • Forest Reranking
• Machine Translation
  • Decoding w/ Language Models
• Forest Rescoring
• Future Directions
Packed Forests and Hypergraph Framework
Packed Forests

- a compact representation of many parses
- by sharing common sub-derivations
- polynomial-space encoding of exponentially large set

(Klein and Manning, 2001; Huang and Chiang, 2005)
Packed Forests

- a compact representation of many parses
- by sharing common sub-derivations
- polynomial-space encoding of exponentially large set

(nodes) --hypersets-- (VP1,6) --hyperedges-- (VP1,6)

0) I saw 2) him 3) with 4) a 5) mirror 6)

(Klein and Manning, 2001; Huang and Chiang, 2005)
Lattices vs. Forests

- forest generalizes “lattice” from finite-state world
- both are compact encodings of exponentially many derivations (paths or trees)
- graph => hypergraph; regular grammar => CFG
Weight Functions

- Each hyperedge $e$ has a weight function $f_e$
  - monotonic in each argument
  - e.g. in CKY, $f_e(a, b) = a \times b \times \text{Pr (rule)}$
- optimal subproblem property in dynamic programming
  - optimal solutions include optimal sub-solutions

update along a hyperedge

\[ d(v) = d(v) \oplus f_e(d(u)) \]
Generalized Viterbi Algorithm

1. topological sort (assumes acyclicity)

2. visit each node $v$ in sorted order and do updates

   - for each incoming hyperedge $e = ((u_1, \ldots, u_{|e|}), v, f_e)$
   - use $d(u_i)$'s to update $d(v)$
   - key observation: $d(u_i)$'s are fixed to optimal at this time

\[ d(v) \oplus = f_e(d(u_1), \ldots, d(u_{|e|})) \]

- time complexity: $O(V+E) = O(E)$ for CKY: $O(n^3)$
1-best => k-best

- we need k-best for pipelined reranking / rescoring
- since 1-best is not guaranteed to be correct
- rerank k-best list with non-local features
- we need fast algorithms for very big values of k

I eat sushi with tuna.
k-best Viterbi Algorithm

- straightforward $k$-best extension
  - a vector of $k$ (sorted) values for each node
  - now what’s the result of $f_e(a, b)$?
    - $k \times k = k^2$ possibilities! => then choose top $k$

- time complexity: $O(k^2 E)$
key insight: do not need to enumerate all $k^2$

- since vectors $\mathbf{a}$ and $\mathbf{b}$ are sorted
- and the weight function $f_e$ is monotonic

- $(a_1, b_1)$ must be the best
  - either $(a_2, b_1)$ or $(a_1, b_2)$ is the 2nd-best

use a priority queue for the frontier

- extract best
- push two successors

- time complexity: $O(k \log k E)$
**k-best Viterbi Algorithm I**

- key insight: do not need to enumerate all $k^2$
  - since vectors \( \mathbf{a} \) and \( \mathbf{b} \) are sorted
  - and the weight function \( f_e \) is monotonic
- \((a_1, b_1)\) must be the best
  - either \((a_2, b_1)\) or \((a_1, b_2)\) is the 2nd-best
- use a priority queue for the frontier
  - extract best
  - push two successors
- time complexity: \(O(k \log k E)\)
**k-best Viterbi Algorithm I**

- key insight: do not need to enumerate all \( k^2 \)
  - since vectors \( \mathbf{a} \) and \( \mathbf{b} \) are sorted
  - and the weight function \( f_e \) is monotonic
- \((a_1, b_1)\) must be the best
  - either \((a_2, b_1)\) or \((a_1, b_2)\) is the 2nd-best
- use a priority queue for the frontier
  - extract best
  - push two successors
- time complexity: \( O(k \log k \ E) \)
**k-best Viterbi Algorithm I**

- **key insight:** do not need to enumerate all $k^2$
  - since vectors $\mathbf{a}$ and $\mathbf{b}$ are sorted
  - and the weight function $f_e$ is monotonic
  - $(a_1, b_1)$ must be the best
  - either $(a_2, b_1)$ or $(a_1, b_2)$ is the 2nd-best
- use a priority queue for the frontier
  - extract best
  - push two successors
- **time complexity:** $O(k \log k E)$
**k-best Viterbi Algorithm 2**

- Algorithm 1 works on each hyperedge sequentially
  - $O(k \log k E)$ is still too slow for big $k$
- Algorithm 2 processes all hyperedges in parallel
  - dramatic speed-up: $O(E + V k \log k)$

Diagram:

- Hyperedge connections:
  - PP1,3 → VP1,6
  - VP3,6 → VP1,6
  - PP1,4 → VP1,6
  - VP4,6 → VP1,6
  - NP1,2 → VP2,3
  - VP2,3 → VP1,6
  - PP3,6 → VP3,6

Graph structures:

- Grids and cubes representing hyperedge connections.
**k-best Viterbi Algorithm 2**

- Algorithm 1 works on each hyperedge sequentially
  - \(O(k \log k E)\) is still too slow for big \(k\)
- Algorithm 2 processes all hyperedges in parallel
  - dramatic speed-up: \(O(E + V k \log k)\)

Diagram:
- Hyperedge
- VP\(_1, 6\)
- PP\(_1, 3\)
- VP\(_3, 6\)
- PP\(_1, 4\)
- VP\(_4, 6\)
- NP\(_1, 2\)
- VP\(_2, 3\)
- PP\(_3, 6\)
**k-best Viterbi Algorithm 3**

- Algorithm 2 computes k-best for each node
  - but we are only interested in k-best of the root node
- Algorithm 3 computes as many as really needed
  - **forward-phase**
    - same as 1-best Viterbi, but stores the forest (keeping alternative hyperedges)
  - **backward-phase**
    - recursively asking “what’s your 2\textsuperscript{nd}-best” top-down
    - asks for more when need more
- only 1-best is known after the forward phase
- recursive backward phase
**k-best Viterbi Algorithm 3**

- only 1-best is known after the forward phase
- recursive backward phase

what's your 2nd-best?

```
S1,9

NP1,3  VP3,9  NP1,5  VP5,9  S1,5  PP5,9
```
• only 1-best is known after the forward phase
• recursive backward phase

what’s your 2nd-best?

hyperedge
k-best Viterbi Algorithm 3

- only 1-best is known after the forward phase
- recursive backward phase

what's your 2nd-best?
Summary of Algorithms

- Algorithms 1 => 2 => 3
  - lazier and lazier (computation on demand)
  - larger and larger locality
  - Algorithm 3 is very fast, but requires storing forest

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>locality</th>
<th>time</th>
<th>space</th>
</tr>
</thead>
<tbody>
<tr>
<td>Algorithm 1</td>
<td>hyperedge</td>
<td>$O( E \cdot k \log k)$</td>
<td>$O(kV)$</td>
</tr>
<tr>
<td>Algorithm 2</td>
<td>node</td>
<td>$O( E + V \cdot k \log k)$</td>
<td>$O(kV)$</td>
</tr>
<tr>
<td>Algorithm 3</td>
<td>global</td>
<td>$O( E + D \cdot k \log k)$</td>
<td>$O(E + kD)$</td>
</tr>
</tbody>
</table>

$E$ - hyperedges: $O(n^3)$; $V$ - nodes: $O(n^2)$; $D$ - derivation: $O(n)$
Experiments - Efficiency

- on state-of-the-art Collins/Bikel parser (Bikel, 2004)
- average parsing time per sentence using Algs. 0, 1, 3

\[ O(E + Dk \log k) \]
Reranking and Oracles

- **oracle** - the candidate closest to the correct parse among the $k$-best candidates

- measures the potential of real reranking

![Graph showing Oracle Parseval score against $k$]

- **Collins 2000**
- **our Algorithms**
Outline

- Packed Forests and Hypergraph Framework
- Exact k-best Search in the Forest (Solution 1)
- Approximate Joint Search (Solution 2) with Non-Local Features
  - Forest Reranking
- Machine Translation
  - Decoding w/ Language Models
  - Forest Rescoring
- Future Directions
Why $n$-best reranking is bad?

- too few variations (limited scope)
  - 41% correct parses are not in $\sim 30$-best (Collins, 2000)
- worse for longer sentences
- too many redundancies
  - 50-best usually encodes 5-6 binary decisions ($2^5 < 50 < 2^6$)
Reranking on a Forest?

- with only local features
  - dynamic programming, tractable (Taskar et al. 2004; McDonald et al., 2005)

- with non-local features
  - on-the-fly reranking at internal nodes
  - top $k$ derivations at each node
  - use as many non-local features as possible at each node
  - chart parsing + discriminative reranking

- we use perceptron for simplicity
Generic Reranking by Perceptron

- for each sentence $s_i$, we have a set of candidates $\text{cand}(s_i)$
- and an oracle tree $y_i^+$, among the candidates
- a feature mapping from tree $y$ to vector $f(y)$

```
1: Input: Training examples \{\text{cand}(s_i), y_i^+\}^{N}_{i=1}
2: w ← 0
3: for t ← 1...T do
4:     for i ← 1...N do
5:         \[ \hat{y} = \arg\max_{y \in \text{cand}(s_i)} w \cdot f(y) \]
6:     if $\hat{y} \neq y_i^+$ then
7:         w ← w + $f(y_i^+) - f(\hat{y})$
8: return w
```

(Collins, 2002)
Features

• a feature $f$ is a function from tree $y$ to a real number
• $f_1(y) = \log \Pr(y)$ is the log Prob from generative parser
• every other feature *counts* the number of times a particular configuration occurs in $y$

![Tree diagram]

instances of Rule feature

\[
f_{100}(y) = f_{S \rightarrow NP \ VP}(y) = 1
\]
\[
f_{200}(y) = f_{NP \rightarrow DT NN}(y) = 2
\]

our features are from
(Charniak & Johnson, 2005)
(Collins, 2000)
Local vs. Non-Local Features

- A feature is **local** iff. it can be factored among local productions of a tree (i.e., hyperedges in a forest).
- Local features can be pre-computed on each hyperedge in the forest; non-locals can not.

**Rule** is local

**ParentRule** is non-local

```
(S
  (TOP)
  (S
    (NP (PRP I)
      (VBD saw)
      (DT the)
      (NN boy)
    )
    (PP (IN with)
      (DT a)
      (NN telescope)
    )
  )
)
```
a **WordEdges** feature classifies a node by its label, (binned) span length, and surrounding words

a **POSEEdges** feature uses surrounding POS tags

\[ f_{400}(y) = f_{NP \text{ saw with }}(y) = 1 \]
a **WordEdges** feature classifies a node by its label, (binned) span length, and surrounding words

a **POSEdges** feature uses surrounding POS tags

**WordEdges** is local

\[ f_{400}(y) = f_{\text{NP}} \text{saw with } (y) = 1 \]
**WordEdges** (C&J 05)

- a **WordEdges** feature classifies a node by its label, (binned) span length, and surrounding words
- a **POSEEdges** feature uses surrounding POS tags

**WordEdges** is local

\[ f_{400}(y) = f_{NP \text{ 2 saw with }}(y) = 1 \]
WordEdges (C&J 05)

- a **WordEdges** feature classifies a node by its label, (binned) span length, and surrounding words
- a **POSEEdges** feature uses surrounding POS tags

WordEdges is local

\[
f_{400}(y) = f_{NP \ 2 \ saw \ with}(y) = 1
\]

POSEEdges is non-local

\[
f_{800}(y) = f_{NP \ 2 \ VBD \ IN \ (y) = 1}
\]
WordEdges (C&J 05)

- a WordEdges feature classifies a node by its label, (binned) span length, and surrounding words
- a POSEEdges feature uses surrounding POS tags

WordEdges is local
\[ f_{400}(y) = f_{NP \ 2 \ saw \ with}(y) = 1 \]

POSEEdges is non-local
\[ f_{800}(y) = f_{NP \ 2 \ VBD \ IN}(y) = 1 \]

Local features comprise ~70% of all instances!
Factorizing non-local features

- going bottom-up, at each node
  - compute (partial values of) feature instances that become computable at this level
  - postpone those uncomputable to ancestors

unit instance of ParentRule feature at the TOP node
Factorizing non-local features

- going bottom-up, at each node
  - compute (partial values of) feature instances that become computable at this level
  - postpone those uncomputable to ancestors

unit instance of ParentRule feature at the TOP node
Factorizing non-local features

• going bottom-up, at each node
  • compute (partial values of) feature instances that become computable at this level
  • postpone those uncomputable to ancestors

unit instance of ParentRule feature at the TOP node
an NgramTree captures the smallest tree fragment that contains a bigram (two consecutive words)

unit instances are boundary words between subtrees
an **NGramTree** captures the smallest tree fragment that contains a bigram (two consecutive words)

unit instances are **boundary words** between subtrees

**unit instance of node A**

```plaintext
NP
  | PRP
  | I
  | saw
NP
  | VBD
  | I
VP
  | TOP
  | S

A_{i,k}
B_{i,j}
C_{j,k}

w_i \ldots w_{j-1}
w_j \ldots w_{k-1}
```

Forest Reranking
• an **NGramTree** captures the smallest tree fragment that contains a bigram (two consecutive words)

• unit instances are **boundary words** between subtrees

### NGramTree (C&J 05)

- **TOP**
  - **S**
    - **NP**
      - **PRP**
        - **I**
          - *saw*
      - **VBD**
        - *saw*
    - **VP**
      - **NP**
        - **DT**
          - *the*
        - **NN**
          - *boy*
      - **IN**
        - *with*
      - **NP**
        - **DT**
          - *a*
        - **NN**
          - *telescope*

### Diagram

- **unit instance of node A**
- **B_{i,j}**
- **C_{j,k}**

> $w_i \ldots w_{j-1}$
> $w_j \ldots w_{k-1}$
**NGramTree** (C&J 05)

- an **NGramTree** captures the smallest tree fragment that contains a bigram (two consecutive words)
- unit instances are **boundary words** between subtrees

![Diagram of a forest reranking tree](image)

*unit instance of node A*

Forest Reranking
**NgramTree (C&J 05)**

- an **NgramTree** captures the smallest tree fragment that contains a bigram (two consecutive words)
- unit instances are **boundary words** between subtrees

```
TOP
  S
  NP
  VBD saw
  PRP I
  NP
    DT the
    NN boy

VP
  PP
    IN with
    DT a
    NN telescope
```

```
A_{i,k}
B_{i,j}
C_{j,k}

w_i \ldots w_{j-1} w_j \ldots w_{k-1}
```

*unit instance of node A*
• an **NgramTree** captures the smallest tree fragment that contains a bigram (two consecutive words)

• unit instances are **boundary words** between subtrees
- an NGramTree captures the smallest tree fragment that contains a bigram (two consecutive words)
- unit instances are boundary words between subtrees

```
NP  |  VBD  |  NP  |  PP |
   |        |      |     |
  |  PRP  |  I   |  saw |
     |        |      |     |
     |  I     |  DT  |
       |        |  the |
        |  VBD   |  NN  |
         |        |  boy |
          |  NP    |  IN  |
            |        |  with|
             |  VBD   |  DT  |
               |        |  a   |
                |  VBD   |  NN  |
                  |        |  a   |
                   |  VBD   |  with|
                      |        |  a   |
                        |  VBD   |  telescope|
                          |        |  a   |
```

unit instance of node A

Forest Reranking
• an NGramTree captures the smallest tree fragment that contains a bigram (two consecutive words)

• unit instances are boundary words between subtrees

unit instance of node A

Forest Reranking
NgramTree (C&J 05)

- An NgramTree captures the smallest tree fragment that contains a bigram (two consecutive words).
- Unit instances are boundary words between subtrees.

Unit instance of node A:

$$w_i \ldots w_{j-1} \quad w_j \ldots w_{k-1}$$
• an **NGramTree** captures the smallest tree fragment that contains a bigram (two consecutive words)

• unit instances are **boundary words** between subtrees
Heads (C&J 05, Collins 00)

- head-to-head lexical dependencies
- we percolate heads bottom-up
- unit instances are between the head word of the head child and the head words of non-head children

```
TOP/saw
  S/saw
  NP/I
    PRP/I VBD/saw NP/the VP/saw
      I saw DT/the NN/boy IN/with PP/with NP/a
                  the boy with DT/a NN/telescope
                      telescope
```
Heads (C&J 05, Collins 00)

- head-to-head lexical dependencies
- we percolate heads bottom-up
- unit instances are between the head word of the head child and the head words of non-head children

![Dependency tree diagram]

- TOP/saw
- S/saw
- VP/saw
- NP/I
- PRP/I
- VBD/saw
- NP/the
- DT/the
- NN/boy
- IN/with
- PP/with
- NP/a
- DT/a
- NN/telescope

Penn University of Pennsylvania

35
Heads (C&J 05, Collins 00)

- head-to-head lexical dependencies
- we percolate heads bottom-up
- unit instances are between the head word of the head child and the head words of non-head children

```
TOP / saw

S / saw

NP / I

PRP / I

VBD / saw

I

I

saw

saw

VP / saw

saw

saw

saw - the; saw - with

unit instances at VP node

NP / the

NN / boy

IN / with

PP / with

NP / a

NP / telescope

DT / a

NN / telescope

DT / the

the

boy

with

with

a

the

boy

a

telescope
```
Approximate Decoding

- bottom-up, keeps top $k$ derivations at each node
- non-monotonic grid due to non-local features

\[ \mathbf{w} \cdot f_N(\cdot) = 0.5 \]

<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 + 0.5</td>
<td>4.0 + 5.0</td>
<td>9.0 + 0.5</td>
</tr>
<tr>
<td>1.1</td>
<td>2.1 + 0.3</td>
<td>4.1 + 5.4</td>
<td>9.1 + 0.3</td>
</tr>
<tr>
<td>3.5</td>
<td>4.5 + 0.6</td>
<td>6.5 + 10.5</td>
<td>11.5 + 0.6</td>
</tr>
</tbody>
</table>
Approximate Decoding

- bottom-up, keeps top \( k \) derivations at each node
- non-monotonic grid due to non-local features

\[
w \cdot f_N(\ ) = 0.5
\]
Approximate Decoding

- bottom-up, keeps top $k$ derivations at each node
- non-monotonic grid due to non-local features

$$\mathbf{w} \cdot \mathbf{f}_N() = 0.5$$

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

- bottom-up, keeps top $k$ derivations at each node
- non-monotonic grid due to non-local features
- priority queue for next-best
- each iteration pops the best and pushes successors
- extract unit non-local features on-the-fly
Algorithm 2 \implies Cube Pruning

- process all hyperedges \textit{simultaneously}! significant savings of computation

\textit{bottom-neck}: the time for on-the-fly non-local feature extraction
Forest vs. n-best Oracles

- on top of Charniak parser (modified to dump forest)
- forests enjoy higher oracle scores than n-best lists
  - with much smaller sizes

![Graph showing comparison between forest and n-best oracles](image-url)

- Parseval F-score vs. average # of hyperedges per sentence
- Forest oracle vs. n-best oracle
Main Results

- pre-comp. is for feature-extraction (can be parallelized)
- # of training iterations is determined on the dev set
- forest reranking outperforms both 50- and 100-best

<table>
<thead>
<tr>
<th>features</th>
<th>$n$ or $k$</th>
<th>pre-comp.</th>
<th>training</th>
<th>$F_1$%</th>
</tr>
</thead>
<tbody>
<tr>
<td>local</td>
<td>50</td>
<td>1.4G / 25h</td>
<td>1 x 0.3h</td>
<td>91.01</td>
</tr>
<tr>
<td>all</td>
<td>50</td>
<td>2.4G / 34h</td>
<td>5 x 0.5h</td>
<td>91.43</td>
</tr>
<tr>
<td>all</td>
<td>100</td>
<td>5.3G / 77h</td>
<td>5 x 1.3h</td>
<td>91.47</td>
</tr>
<tr>
<td>local</td>
<td>-</td>
<td>1.2G / 5.1h</td>
<td>3 x 1.4h</td>
<td>91.25</td>
</tr>
<tr>
<td>all</td>
<td>$k=15$</td>
<td>1.2G / 5.1h</td>
<td>4 x 1.1h</td>
<td>91.69</td>
</tr>
</tbody>
</table>

Baseline: 1-best Charniak parser

89.72
Comparison with Others

<table>
<thead>
<tr>
<th>type</th>
<th>system</th>
<th>F₁%</th>
</tr>
</thead>
<tbody>
<tr>
<td>D</td>
<td>Collins (2000)</td>
<td>89.7</td>
</tr>
<tr>
<td></td>
<td>Henderson (2004)</td>
<td>90.1</td>
</tr>
<tr>
<td></td>
<td>Charniak and Johnson (2005)</td>
<td>91.0</td>
</tr>
<tr>
<td></td>
<td>updated (2006)</td>
<td>91.4</td>
</tr>
<tr>
<td></td>
<td>Petrov and Klein (2008)</td>
<td>88.3</td>
</tr>
<tr>
<td></td>
<td><em>this work</em></td>
<td>91.7</td>
</tr>
<tr>
<td>G</td>
<td>Bod (2000)</td>
<td>90.7</td>
</tr>
<tr>
<td></td>
<td>Petrov and Klein (2007)</td>
<td>90.1</td>
</tr>
<tr>
<td>S</td>
<td>McClosky et al. (2006)</td>
<td>92.1</td>
</tr>
</tbody>
</table>

best accuracy to date on the Penn Treebank
Outline

• Packed Forests and Hypergraph Framework
• Exact $k$-best Search in the Forest
• Approximate Joint Search with Non-Local Features
  • Forest Reranking
• Machine Translation
  • Decoding w/ Language Models
  • Forest Rescoring
• Future Directions
Statistical Machine Translation

Spanish/English Bilingual Text

Statistical Analysis

translation model (TM) competency

Spanish

Broken English

English Text

Statistical Analysis

language model (LM) fluency

Spanish/English

Broken English

I am so hungry

Que hambre tengo yo

What hunger have I

Hungry I am so

Have I that hunger

I am so hungry

How hunger have I

...
Statistical Machine Translation

[k]-best rescoring (Algorithm 3)

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

I am so hungry

Que hambre tengo yo

(k)-best rescoring (Algorithm 3)

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

I am so hungry

(Knight and Koehn, 2003)
Statistical Machine Translation

- **Spanish/English** Bilingual Text
- **English** Text

Statistical Analysis

- **Spanish**
- **English**

Phrase-based TM

- **Que hambre tengo yo**
- **I am so hungry**

Syntax-based

- **Integrated decoder**
- Computationally challenging! 😞

n-gram LM

Integrated decoder
Forest Rescoring

- Spanish/English Bilingual Text
  - Statistical Analysis

- English Text
  - Statistical Analysis

- Statistical Analysis
  - packed forest
    - Que hambre tengo yo
      - forest rescoring
        - I am so hungry
  - as non-local info
    - n-gram LM
Syntax-based Translation

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

\[
\begin{align*}
\text{VP} & \rightarrow \text{PP}^{(1)} \text{VP}^{(2)}, \\
\text{VP} & \rightarrow \text{juxing le huitan}, \\
\text{PP} & \rightarrow \text{yu Shalong}, \\
\text{VP}^{(2)} & \text{PP}^{(1)} \\
\text{VP} \rightarrow \text{juxing le huitan, held a meeting} \\
\text{VP} \rightarrow \text{yu Shalong, with Sharon}
\end{align*}
\]
Translation as Parsing

- translation with SCFGs => monolingual parsing
- parse the source input with the source projection
- build the corresponding target sub-strings in parallel

\[
\begin{align*}
\text{VP} & \rightarrow \text{PP}^{(1)} \text{ VP}^{(2)}, \\
\text{VP} & \rightarrow \text{juxing le huitan}, \\
\text{PP} & \rightarrow \text{yu Shalong},
\end{align*}
\]
Translation as Parsing

- translation with SCFGs => monolingual parsing
- parse the source input with the source projection
- build the corresponding target sub-strings in parallel

\[
\begin{align*}
\text{VP} & \rightarrow \text{PP}^{(1)} \text{ VP}^{(2)}, & \text{VP}^{(2)} \text{ PP}^{(1)} \\
\text{VP} & \rightarrow \text{juxing le huitan, held a meeting with Sharon} \\
\text{PP} & \rightarrow \text{yu Shalong, with Sharon} \\
\end{align*}
\]
Adding a Bigram Model

- exact dynamic programming
  - nodes now split into +LM items
  - with English boundary words
- search space too big for exact search
  - beam search: keep at most k +LM items each node
- but can we do better?
**Non-Monotonic Grid**

Non-monotonicity due to LM combo costs

\[
\begin{align*}
\text{(VP} \text{ held } \star \text{ meeting})_{3,6} & : \quad 1.0 \\
\text{(VP} \text{ held } \star \text{ talk})_{3,6} & : \quad 1.1 \\
\text{(VP} \text{ hold } \star \text{ conference})_{3,6} & : \quad 3.5
\end{align*}
\]

<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 + 0.5</td>
<td>4.0 + 5.0</td>
<td>9.0 + 0.5</td>
</tr>
<tr>
<td>1.1</td>
<td>2.1 + 0.3</td>
<td>4.1 + 5.4</td>
<td>9.1 + 0.3</td>
</tr>
<tr>
<td>3.5</td>
<td>4.5 + 0.6</td>
<td>6.5 + 10.5</td>
<td>11.5 + 0.6</td>
</tr>
</tbody>
</table>
Non-Monotonic Grid

non-monotonicity due to LM combo costs

<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 + 0.5</td>
<td>4.0 + 5.0</td>
</tr>
<tr>
<td>(VP$_{3,6}$ held * talk)</td>
<td>1.1</td>
<td>2.1 + 0.3</td>
<td>4.1 + 5.4</td>
</tr>
<tr>
<td>(VP$_{3,6}$ hold * conference)</td>
<td>3.5</td>
<td>4.5 + 0.6</td>
<td>6.5 + 10.5</td>
</tr>
</tbody>
</table>
Algorithm 2 - Cube Pruning

<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></td>
<td></td>
<td></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>
</tbody>
</table>

<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>
Algorithm 2 => Cube Pruning

$k$-best Algorithm 2, with search errors

process all hyperedges simultaneously! significant savings of computation
Phrase-based: Translation Accuracy

- **Speed:** ++
  - ~100 times faster

**Algorithm 2:**
- **Cube pruning**
Syntax-based: Translation Accuracy

speed ++

quality ++

Algorithm 2: cube pruning
Algorithm 3: cube growing
Conclusion so far

- General framework of DP on hypergraphs
  - monotonicity => exact 1-best algorithm
- Exact $k$-best algorithms
- Approximate search with non-local information
  - Forest Reranking for discriminative parsing
  - Forest Rescoring for MT decoding
- Empirical Results
  - orders of magnitudes faster than previous methods
  - best Treebank parsing accuracy to date
These algorithms have been widely implemented in:

- state-of-the-art parsers
  - Charniak parser
  - McDonald’s dependency parser
  - MIT parser (Collins/Koo), Berkeley and Stanford parsers
  - DOP parsers (Bod, 2006/7)

- major statistical MT systems
  - Syntax-based systems from ISI, CMU, BBN, ...
  - Phrase-based system: Moses [underway]
Future Directions
Further work on Forest Reranking

- Better Decoding Algorithms
  - pre-compute most non-local features
  - use Algorithm 3 cube growing
  - intra-sentence level parallelized decoding
- Combination with Semi-supervised Learning
  - easy to apply to self-training (McClosky et al., 2006)
- Deeper and deeper Decoding (e.g., semantic roles)
- Other Machine Learning Algorithms
- Theoretical and Empirical Analysis of Search Errors
Machine Translation / Generation

- Discriminative training using non-local features
- Local-features showed modest improvement on phrase-base systems (Liang et al., 2006)
- Plan for syntax-based (tree-to-string) systems
  - Fast, linear-time decoding
- Using packed parse forest for
  - Tree-to-string decoding (Mi, Huang, Liu, 2008)
  - Rule extraction (tree-to-tree)
- Generation / Summarization: non-local constraints
Thanks!

Questions?

Comments?
Speed vs. Search Quality

tested on our faithful clone of Pharaoh

- log Prob

average model cost

average number of hypotheses per sentence

full-integration
cube pruning

speed ++

quality ++
Speed vs. Search Quality

tested on our faithful clone of Pharaoh

speed ++

quality ++

32 times faster
tested on our faithful clone of Pharaoh

Speed vs. Search Quality

32 times faster
Syntax-based: Search Quality

- Speed ++
- Quality ++
- 10 times faster
Tree-to-String System

- syntax-directed, English to Chinese (Huang, Knight, Joshi, 2006)
- first parse input, and then recursively transfer

synchronous tree-substitution grammars (STSG)
(Galley et al., 2004; Eisner, 2003)

extended to translate a packed-forest instead of a tree
(Mi, Huang, Liu, 2008)
Tree-to-String System

- syntax-directed, English to Chinese (Huang, Knight, Joshi, 2006)
- first parse input, and then recursively transfer

synchronous tree-substitution grammars (STSG) (Galley et al., 2004; Eisner, 2003)

extended to translate a packed-forest instead of a tree (Mi, Huang, Liu, 2008)
• extract features on the 50-best parses of train set

• cut off low-freq. features with count < 5
  • counts are “relative” -- change on at least 5 sentences

• feature templates
  • 4 local from (Charniak and Johnson, 2005)
  • 4 local from (Collins, 2000)
  • 7 non-local from (Charniak and Johnson, 2005)

• 800, 582 feature instances (30% non-local)
  • cf. C & J: 1.3 M feature instances (60% non-local)
Forest Oracle

the candidate tree that is closest to gold-standard
Optimal Parseval F-score

$$y_i^+ \triangleq \arg\max_{y \in \text{cand}(s_i)} F(y, y_i^*)$$

$$F(y, y^*) \triangleq \frac{2PR}{P + R} = \frac{2|y \cap y^*|}{|y| + |y^*|}$$

- Parseval F$_1$-score is the harmonic mean between labeled precision and labeled recall
  - can not optimize F-scores on sub-forests separately
- we instead use dynamic programming
  - optimizes the number of matched brackets per given number of test brackets
  - “when the test (sub-) parse has 5 brackets, what is the max. number of matched brackets?”

$$ora[v]: \mathbb{N} \rightarrow \mathbb{N}$$

$$ora[v](t) \triangleq \max_{y_v:|y_v|=t} |y_v \cap y^*|$$
Combining Oracle Functions

- combining two oracle functions along a hyperedge $e = \langle (v,u), w \rangle$ needs a convolution operator $\otimes$

\[(f \otimes g)(t) \triangleq \max_{t_1 + t_2 = t} f(t_1) + g(t_2)\]

<table>
<thead>
<tr>
<th>$t$</th>
<th>$f(t)$</th>
<th>$t$</th>
<th>$g(t)$</th>
<th>$t$</th>
<th>$(f \otimes g)(t)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>1</td>
<td>4</td>
<td>4</td>
<td>6</td>
<td>5</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>5</td>
<td>4</td>
<td>7</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td></td>
<td>8</td>
<td>6</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

$w$ $\rightarrow$ $v$ $\rightarrow$ $u$
Combining Oracle Functions

- combining two oracle functions along a hyperedge $e = \langle (v,u), w \rangle$ needs a convolution operator $\otimes$

$$ \langle f \otimes g \rangle(t) \triangleq \max_{t_1 + t_2 = t} f(t_1) + g(t_2) $$

<table>
<thead>
<tr>
<th>$t$</th>
<th>$f(t)$</th>
<th>$g(t)$</th>
<th>$(f \otimes g)(t)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>1</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>4</td>
<td>6</td>
</tr>
<tr>
<td>5</td>
<td>4</td>
<td>4</td>
<td>8</td>
</tr>
</tbody>
</table>

- this node matched?

<table>
<thead>
<tr>
<th>$t$</th>
<th>$(f \otimes g) \uparrow(1,0)(t)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>5</td>
</tr>
<tr>
<td>8</td>
<td>6</td>
</tr>
<tr>
<td>9</td>
<td>6</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$t$</th>
<th>$(f \otimes g) \uparrow(1,1)(t)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>6</td>
</tr>
<tr>
<td>8</td>
<td>7</td>
</tr>
<tr>
<td>9</td>
<td>7</td>
</tr>
</tbody>
</table>
Combining Oracle Functions

- combining two oracle functions along a hyperedge $e = \langle (v,u), w \rangle$ needs a convolution operator $\otimes$

$$ (f \otimes g)(t) \triangleq \max_{t_1 + t_2 = t} f(t_1) + g(t_2) $$

**Final answer:**

- $F(y^+, y^*) = \max_t \frac{2 \cdot \text{ora}[\text{TOP}](t)}{t + |y^*|}$

**Table:**

<table>
<thead>
<tr>
<th>$t$</th>
<th>$(f \otimes g)_{(1,0)}(t)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>5</td>
</tr>
<tr>
<td>8</td>
<td>6</td>
</tr>
<tr>
<td>9</td>
<td>6</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$t$</th>
<th>$g(t)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>4</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$t$</th>
<th>$(f \otimes g)(t)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>5</td>
</tr>
<tr>
<td>7</td>
<td>6</td>
</tr>
<tr>
<td>8</td>
<td>6</td>
</tr>
</tbody>
</table>

**Question:**

- This node matched?

- $\text{ora}[w]$
Forest Pruning

a variant of Inside-Outside Algorithm
Pruning (J. Graehl, unpublished)

- prune by marginal probability (Charniak and Johnson, 2005)
  - but we prune hyperedges as well as nodes
- compute Viterbi inside cost $\beta(v)$ and outside cost $\alpha(v)$
- compute merit $\alpha\beta(e) = \alpha(\text{head}(e)) + \sum_{u \in \text{tails}(e)} \beta(u)$
  - cost of the best derivation that traverses $e$
- prune away hyperedges that have $\alpha\beta(e) - \beta(\text{TOP}) > p$
  - difference: a node can “partially” survive the beam
- can prune on average 15% more hyperedges than C&J