Forest-based Algorithms in Natural Language Processing

Liang Huang
Advisors: Aravind Joshi (Penn) and Kevin Knight (USC/ISI)

Dissertation Defense  September 18th, 2008
NLP is all about ambiguities

I saw her duck.
NLP is all about ambiguities

I saw her duck.
NLP is all about ambiguities

I saw her duck.

- how about...
  - I saw her duck with a telescope.
NLP is all about ambiguities

I saw her duck.

- how about...
  - I saw her duck with a telescope.
  - I saw her duck with a telescope in the garden...
Ambiguities and Context in MT
NLP is HARD!

- exponential explosion of the search space
- non-local dependencies (context)
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 $=>$ parsing $=>$ semantics
  - *(open)* need 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 => parsing => semantics
  • *(open)* need efficient algorithms for \( k \)-best search

• **Solution 2**: exact joint search on a much larger space
  • examples: head/parent annotations; often intractable
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
  - **(open)** need efficient algorithms for $k$-best search

- **Solution 2**: exact joint search on a much larger space
  - examples: head/parent annotations; often intractable

- **Solution 3**: approximate joint search **(focus of thesis)**
  - **(open)** integrate non-local information on the fly
Outline

- Forest: Packing Exponential Ambiguities
- Exact $k$-best Search in Forest (Solution 1)
- Approximate Joint Search with Non-Local Features (Solution 3)
  - Forest Reranking (and rescoring)
- Forest-based Translation (Solutions 2+3+1)
  - Forest-based Decoding
  - Forest-based Rule Extraction
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)
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
I-best Viterbi on Forest

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: \( O(n^3) \)
Outline

- Forest: Packing Exponential Ambiguities
- Exact $k$-best Search in Forest (Solution 1)
- Approximate Joint Search with Non-Local Features (Solution 3)
  - Forest Reranking (and rescoring)
- Forest-based Translation (Solutions 2+3)
  - Forest-based Decoding
  - Forest-based Rule Extraction
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: $O(k^2 E)$
**k-best Viterbi Algorithm I**

- **key insight:** do not need to enumerate all $k^2$
  - since vectors $a$ and $b$ 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 $a$ and $b$ 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 $a$ and $b$ 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: $O(E + V k \log k)$
**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 2nd-best” top-down
    - asks for more when need more
# 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>$O( E k \log k )$</td>
<td>$O(k V)$</td>
</tr>
<tr>
<td>Algorithm 2</td>
<td>node</td>
<td>$O( E + V k \log k )$</td>
<td>$O(k V)$</td>
</tr>
<tr>
<td>Algorithm 3</td>
<td>global</td>
<td>$O( E + D k \log k )$</td>
<td>$O(E + k D)$</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

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

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

- measures the potential of real reranking

---

**Graph:**

- **Oracle Parseval score**
- **x-axis:** $k$
- **y-axis:** Oracle Parseval score

**Curves:**

- **our Algorithms**
- **Collins 2000** (turns down DP)
Outline

- Packed Forests and Hypergraph Framework
- Exact k-best Search in Forest (Solution 1)
- Approximate Joint Search with Non-Local Features (Solution 3)
  - Forest Reranking
- Application: Forest-based Translation
  - Forest-based Decoding
  - Forest-based Rule Extraction
Why not $k$-best reranking?

- too few variations (limited scope)
  - 41% correct parses are not in ~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 (Solution 2)
  • dynamic programming, exact, tractable (Taskar et al. 2004; McDonald et al., 2005)

• with non-local features (Solution 3)
  • 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
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 $y$.

Our features are from:
(Charniak & Johnson, 2005)
(Collins, 2000)

Instances of Rule feature:

$$f_{100}(y) = f_{S \rightarrow NP \ VP}(y) = 1$$
$$f_{200}(y) = f_{NP \rightarrow DT \ NN}(y) = 2$$
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 cannot.

```
TOP
  | S
  | NP
  | PRP
  | I
  | saw

NP
  | VBD
  | saw

VP
  | NP
  | DT
  | the
  | NN
  | boy

PP
  | IN
  | with

NP
  | DT
  | a
  | NN
  | telescope
```

- **ParentRule** is non-local.
- **Rule** is local.
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

```
*VP*
  VBD  NP  PP
  |    |    |
  saw DT NN IN NP
  |    |    |
  the DT NN NP
  |    |    |
  a telescope
```

unit instance of ParentRule
feature at VP node

local features factor across hyperedges statically

non-local features factor across nodes dynamically
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
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
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

**Diagram:**

- **S** (Sentence)
  - **NP** (Nominal Phrase)
    - **PRP** (Personal Pronoun)
    - **VBD** (Past Tense Verb)
    - **the** (Determiner)
    - **boy** (Noun)
    - **with** (Preposition)
    - **NP** (Nominal Phrase)
      - **DT** (Determiner)
      - **a** (Determiner)
      - **telescope** (Noun)

- **VP** (Verb Phrase)
  - **I** (Personal Pronoun)
    - **saw** (Past Tense Verb)
  - **NP** (Nominal Phrase)
  - **PP** (Prepositional Phrase)

**Note:**

- **Local features** factor across hyperedges **statically**
- **Non-local features** factor across **nodes dynamically**

**Unit instance** of ParentRule feature at S 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 S node

local features factor across hyperedges statically

non-local features factor across nodes dynamically
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 S node

local features factor across hyperedges \textit{statically}

non-local features factor across nodes \textit{dynamically}
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 TOP node

non-local features factor across nodes *dynamically*

local features factor across hyperedges *statically*
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 TOP node

non-local features factor across nodes dynamically

local features factor across hyperedges statically
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
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

\[ \begin{align*}
&\text{unit instance of node A} \\
&\quad \quad = (A_{i,k}, B_{i,j}, C_{j,k}) \\
&\quad \quad \text{with } w_i \ldots w_{j-1}, w_j \ldots w_{k-1}
\end{align*} \]
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
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
  NP'
    PRP
    VBD

  NP
    DT
    NN
    IN
    NP
      DT
      NN
      NN

  VP

I saw the boy with a telescope
```

unit instance of node A
Approximate Decoding

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

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

<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

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

<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

$$w \cdot f_N(\text{tree}) = 0.5$$

<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>
Algorithm 2 => Cube Pruning

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

$\mathbf{w} \cdot f_N(\cdot) = 0.5$
Algorithm 2 \implies \text{Cube Pruning}

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

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

<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>
Algorithm 2 => Cube Pruning

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

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

<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>2.5</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>
Algorithm 2 => Cube Pruning

- process all hyperedges simultaneously! significant savings of computation

there are search errors, but the trade-off is favorable.
Forest vs. $k$-best Oracles

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

- forest reranking outperforms both 50-best and 100-best reranking
- and can be trained on the whole treebank in ~1 day even with a pure Python implementation!

<table>
<thead>
<tr>
<th>approach</th>
<th>training time</th>
<th>F1%</th>
</tr>
</thead>
<tbody>
<tr>
<td>baseline: 1-best Charniak parser</td>
<td></td>
<td>89.72</td>
</tr>
<tr>
<td>50-best reranking</td>
<td>4 x 0.3h</td>
<td>91.43</td>
</tr>
<tr>
<td>100-best reranking</td>
<td>4 x 0.7h</td>
<td>91.49</td>
</tr>
<tr>
<td>forest reranking</td>
<td>4 x 6.1h</td>
<td>91.69</td>
</tr>
</tbody>
</table>
### Comparison with Others

<table>
<thead>
<tr>
<th>type</th>
<th>system</th>
<th>F$_1$%</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>D</strong></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><strong>91.7</strong></td>
</tr>
<tr>
<td><strong>G</strong></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><strong>S</strong></td>
<td>McClosky et al. (2006)</td>
<td><strong>92.1</strong></td>
</tr>
</tbody>
</table>

*best accuracy to date on the Penn Treebank*
on to Machine Translation...

applying the same ideas of non-locality...
Context in Translation

Left: A sign in Chinese that translates to "Poisonous & Evil Rubbish." It depicts a person throwing trash into a bin.

Right: A sign with a caution symbol and the text "Slip carefully."
Context in Translation

fluency problem
(n-gram)

小心 X  <=>  be careful not to X
(SCFG)

context is important! (but non-local)
fluency problem (n-gram)

context is important! (but non-local)

小心 VP <=> be careful not to VP
小心 NP <=> be careful of NP

(SCFG)
Google:
toxic and harmful waste
carefully slide

fluency problem
(n-gram)

小心翼翼 VP <=> be careful not to VP
小心翼翼 NP <=> be careful of NP

(SCFG)

context is important! (but non-local)
MT decoding: forest rescoring

- Decoding: SCFG + n-gram LM
  - LM for fluency of the output
- n-gram LM as non-local features
- Same idea as in forest reranking
  - Algorithm 2 => cube pruning
  - Orders of magnitudes speed-up
- Also works for phrase-based MT
- Used in almost all major syntax-based systems

小心 X => be careful not to X
MT decoding: forest rescoring

- decoding: SCFG + n-gram LM
- LM for *fluency* of the output
- n-gram LM as non-local features
- same idea as in forest reranking
  - algorithm 2 => cube pruning
  - orders of magnitudes speed-up
  - also works for phrase-based MT
- used in almost all major syntax-based systems
MT decoding: forest rescoring

- Decoding: SCFG + n-gram LM
- LM for fluency of the output
- N-gram LM as non-local features
- Same idea as in forest reranking
  - Algorithm 2 => cube pruning
  - Orders of magnitudes speed-up
- Also works for phrase-based MT
- Used in almost all major syntax-based systems

Bush held a talk with Sharon

| S_0,6 |
| VP_{1,6} |
| PP_{1,3} |
| VP_{3,6} |

Bush with Shalong held a talk

| NP_{0,1} |
| PP_{1,3} |
| VP_{3,6} |

Bushi yu Shalong juxing le huitan bigram

| VP_{3,6} |
| PP_{1,3} |
MT decoding: forest rescoring

- decoding: SCFG + n-gram LM
- LM for fluency of the output
- n-gram LM as non-local features
- same idea as in forest reranking
  - algorithm 2 => cube pruning
  - orders of magnitudes speed-up
- also works for phrase-based MT
- used in almost all major syntax-based systems

Bush held a talk with Sharon

Bushi yu Shalong juxing le huitan bigram

NP0,1 PP1,3 VP3,6

VP1,6

S0,6

Bush with Shalong held a talk

VP3,6

PP1,3

held ... talk with ... Sharon
Outline

- Packed Forests and Hypergraph Framework
- Exact $k$-best Search in Forest (Solution 1)
- Approximate Joint Search with Non-Local Features (Solution 3)
  - Forest Reranking (and rescoring)
- Application: Forest-based Translation
  - Forest-based Decoding
  - Forest-based Rule Extraction
Two Approaches in Syntax MT

Bush held a talk with Sharon

S_0,6

VP_1,6

Bush with Shalong held a talk

NP_0,1 PP_1,3 VP_3,6

Bushi yu Shalong juxing le huitan
Two Approaches in Syntax MT

- string-based (Wu 97; Chiang 05; Galley et al 06)
  - parse the source-language string
  - with a synchronous grammar
  - generate translations accordingly

Bush held a talk with Sharon

\[
S_{0,6} \\
\text{held a talk} \\
\text{with Sharon} \\
\text{VP}_{1,6}
\]

Bush with Shalong held a talk

\[
\text{NP}_{0,1} \\
\text{PP}_{1,3} \\
\text{VP}_{3,6}
\]

Bushi yu Shalong juxing le huitan
Two Approaches in Syntax MT

• **string-based** (Wu 97; Chiang 05; Galley et al 06)
  - parse the source-language **string**
  - with a synchronous grammar
  - generate translations accordingly

• **tree-based** (Quirk et al 05; Liu et al 06; Huang et al 06)
  - start from source-language **parse tree**
  - recursively convert it to the target-language
  - faster decoding; more expressive translation grammar
  - **Problem**: commits to 1-best parse tree! => k-best trees?
Two Approaches in Syntax MT

- **string-based** (Wu 97; Chiang 05; Galley et al 06)
  - parse the source-language string
  - with a synchronous grammar
  - generate translations accordingly

- **tree-based** (Quirk et al 05; Liu et al 06; Huang et al 06)
  - start from source-language parse tree
  - recursively convert it to the target-language
  - faster decoding; more expressive translation grammar
  - **Problem**: commits to 1-best parse tree! => k-best trees?
  - **Idea**: use a parse forest!
Tree-based Translation

- get 1-best parse tree; then convert to English

```
IP
   NP
     NPB     CC     NPB
       Bush   yǔ    Shālóng
     and/with Sharon
   VPB
     VV   AS   NPB
       jǔxíng  le   huìtán

“Bush held a meeting with Sharon”
```
Tree-based Translation

- recursive rewrite by pattern-matching

(Galley et al. 2004; Huang, Knight, Joshi 2006)
Tree-based Translation

- recursive rewrite by pattern-matching

(Galley et al. 2004; Huang, Knight, Joshi 2006)
Tree-based Translation

• recursive rewrite by pattern-matching

(Galley et al. 2004; Huang, Knight, Joshi 2006)
Tree-based Translation

- recursive rewrite by pattern-matching

(Galley et al. 2004; Huang, Knight, Joshi 2006)
Tree-based Translation

- recursively solve unfinished subproblems

(Galley et al. 2004; Huang, Knight, Joshi 2006)
Tree-based Translation

- recursively solve unfinished subproblems

---

(Galley et al. 2004; Huang, Knight, Joshi 2006)
Tree-based Translation

- recursively solve unfinished subproblems

Bush

```
VPB
VV  AS  NPB
jǔxíng  le  huìtán
```

with

```
NPB
Shālóng
```

```
VPB
VV  AS  x₁:NPB
jǔxíng  le  → held  x₁
```
Tree-based Translation

- recursively solve unfinished subproblems

Bush held with NPB

$\text{VPB} \rightarrow \text{NPB}$

$\text{VV} \rightarrow \text{NPB}$

$\text{AS} \rightarrow \text{huìtán}$

$\text{le}$

$\text{jǔxíng}$

(Galley et al. 2004; Huang, Knight, Joshi 2006)
Tree-based Translation

- continue pattern-matching

Bush held with NPB $\text{huìtán}$ with NPB $\text{Shālóng}$

(Galley et al. 2004; Huang, Knight, Joshi 2006)
Tree-based Translation

- continue pattern-matching

Bush held a meeting with Sharon

NPB huìtán

NPB Shālóng

(Galley et al. 2004; Huang, Knight, Joshi 2006)
Tree-based Translation

- continue pattern-matching

Bush held a meeting with Sharon

(Galley et al. 2004; Huang, Knight, Joshi 2006)
Tree-based Translation

- continue pattern-matching

Bush held a meeting with Sharon

**pros**: simplicity, faster decoding, expressive grammar, “extended domain of locality”

no need for binarization, ...

**cons**: commits to 1-best tree!

**idea**: instead of k-best parses,

use a packed parse forest!

(Galley et al. 2004; Huang, Knight, Joshi 2006)
Forest-based Decoding

“and” / “with”
Forest-based Decoding

“and” / “with”
Forest-based Decoding

“and” / “with”
Forest-based Decoding

“and” / “with”
Forest-based Decoding
Forest-based Decoding

pattern-matching on forest complexity: linear in forest size
string-based: cubic in sent. len.

“and” / “with”
Translation Forest
Translation Forest
Translation Forest

“Bush”

“Sharon”

“held a meeting”

Diagram shows the syntactic structure of the sentence "Sharon and Bush held a meeting."
“Bush held a meeting with Sharon”
The Whole Pipeline

input sentence
parser
parse forest

pattern-matching w/ translation rules (exact)

translation forest
cube pruning (approx.)

translation+LM forest

best derivation
l-best translation

k-best Algorithm 3 (exact)

k-best translations

(Huang and Chiang, 2005; 2007; Chiang, 2007)
The Whole Pipeline

input sentence → parser

parse forest

pattern-matching w/ translation rules (exact)

translation forest

cube pruning (approx.)

translation+LM forest

best derivation

1-best translation

k-best Algorithm 3 (exact)

k-best translations

packets forests

(Huang and Chiang, 2005; 2007; Chiang, 2007)
$k$-best trees vs. forest-based
forest as virtual $\infty$-best list

- how often is the $i$th-best tree picked by the decoder?
Where are the rules from?

- source parse tree, target sentence, and alignment
- intuition: contiguous span

GHKM - (Galley et al 2004; 2006)
Where are the rules from?

- source parse tree, target sentence, and alignment
- intuition: contiguous span

GHKM - (Galley et al 2004; 2006)
Where are the rules from?

- source parse tree, target sentence, and alignment
- intuition: contiguous span

GHKM - (Galley et al 2004; 2006)
Where are the rules from?

- source parse tree, target sentence, and alignment
- compute target spans

GHKM - (Galley et al 2004; 2006)
Where are the rules from?

- source parse tree, target sentence, and alignment
- well-formed fragment: contiguous and faithful t-span

GHKM - (Galley et al 2004; 2006)
Where are the rules from?

- source parse tree, target sentence, and alignment
- well-formed fragment: contiguous and faithful t-span

GHKM - (Galley et al 2004; 2006)
Forest-based Rule Extraction

- same cut set computation; different fragmentation

also in (Wang, Knight, Marcu, 2007)
Forest-based Rule Extraction

- same cut set computation; different fragmentation

also in (Wang, Knight, Marcu, 2007)
Forest-based Rule Extraction

- same cut set computation; different fragmentation

also in (Wang, Knight, Marcu, 2007)
Forest-based Rule Extraction

- same cut set computation; different fragmentation

\[
\text{IP}(x_1: \text{NPB}, x_2: \text{VP}) \rightarrow x_1 x_2
\]

also in (Wang, Knight, Marcu, 2007)
Forest-based Rule Extraction

- same admissible set definition; different fragmentation

\[
\text{IP}(x_1:\text{NPB} \; x_2:\text{VP}) \rightarrow x_1 \; x_2
\]

also in (Wang, Knight, Marcu, 2007)
• same admissible set definition; different fragmentation

\[ \text{IP}(x_1: \text{NPB} \ x_2: \text{VP}) \rightarrow x_1 \ x_2 \]
Forest-based Rule Extraction

- same admissible set definition; different fragmentation

\[ \text{IP}(x_1; \text{NPB} \ x_2; \text{VP}) \rightarrow x_1 \ x_2 \]

Also in (Wang, Knight, Marcu, 2007)
Forest-based Rule Extraction

- same admissible set definition; different fragmentation

also in (Wang, Knight, Marcu, 2007)
Forest-based Rule Extraction

- forest can extract smaller chunks of rules
• forest can extract smaller chunks of rules

$IP(x_1:NPB \ x_2:VP) \rightarrow x_1 \ x_2$

$IP$  

$NP$  

$x_4:VPB$

$x_1:NPB$  

$x_2:CC$  

$x_3:NPB$

$\rightarrow x_1 \ x_4 \ x_2 \ x_3$

$VP (x_1:PP \ x_2:VPB) \rightarrow x_2 \ x_1$

also in (Wang, Knight, Marcu, 2007)
Forest-based Rule Extraction

- forest can extract smaller chunks of rules

Also in (Wang, Knight, Marcu, 2007)
Rule Extraction Results

- Forest extraction is twice as fast as 30-best and produces significantly better BLEU score.
- Uses 16% new rules never seen on any 1-best parses.
- Not just changing the distribution of rules.

<table>
<thead>
<tr>
<th></th>
<th>1-best</th>
<th>30-best</th>
<th>forest</th>
</tr>
</thead>
<tbody>
<tr>
<td>Extraction time / 1k sent.</td>
<td>0.24</td>
<td>5.56</td>
<td>2.36</td>
</tr>
<tr>
<td>BLEU</td>
<td>0.2430</td>
<td>0.2488</td>
<td>0.2533</td>
</tr>
<tr>
<td>New rules used in 1-best trans.</td>
<td>-</td>
<td>8.7%</td>
<td>16.3%</td>
</tr>
</tbody>
</table>
Large-Scale Experiments

- FBIS: 239k sentence pairs (7M/9M Chinese/English words)
- Forest in both extraction and decoding
- Forest-forest results is 2.5 points better than 1-best
- And outperforms Hiero

<table>
<thead>
<tr>
<th>rules \ decoding</th>
<th>I-best tree</th>
<th>forest</th>
</tr>
</thead>
<tbody>
<tr>
<td>I-best tree</td>
<td>0.2560</td>
<td>0.2674</td>
</tr>
<tr>
<td>30-best trees</td>
<td>0.2634</td>
<td>0.2767</td>
</tr>
<tr>
<td>forest</td>
<td>0.2679</td>
<td>0.2816</td>
</tr>
<tr>
<td>Hiero</td>
<td></td>
<td>0.2738</td>
</tr>
</tbody>
</table>
Conclusions

- A general framework of DP on hypergraphs
- Exact $k$-best algorithms
- Approximate search with non-local features
  - Forest Reranking for discriminative parsing
  - Forest Rescoring for MT decoding
- Forest-based Translation
  - translates and extract rules on a parse forest
  - as simple as tree-based; as good as string-based
- Future Directions: even faster search with richer info...
Forest is your friend. Save the forest.

Thank you!
Global Feature - RightBranch

- length of rightmost (non-punctuation) path
- English has a right-branching tendency

(Charniak and Johnson, 2005)